社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－AL－95 (4)
2004／5／21

アドホックネットワークにおける最適証明書分散アルゴリズム

鄭 華[†]、 大村伸吾[†]、 内田次郎[†]、 和田幸一[†]

アドホックネットワークにおける安全な通信を行うために配布される証明書を各利用者に分散させる問題を考える。各利用者に分配される証明書の平均数を分散コストとする。本論文では、効率的に証明書を分散させる2つの証明書分散アルゴリズムを提案する。強連結グラフG=(V, E)と一般の有向グラフH=(V', E')において、D_GをグラフGの直径、pをHの強連結成分の数、d_{max}をHの各強連結成分の直径に対する最大値としたとき、それぞれ分散コストに対する上界$O(D_G+|E|/|V|)$と$O(pd_{max}+|E'|/|V'|)$を証明し、さらに、いくつかのグラフクラスに対してここで提案するアルゴリズムが最適であることも示す。

An Optimal Certificate Dispersal Algorithm for Mobile Ad Hoc Networks

HUA ZHENG[†], SHINGO OMURA[†], JIRO UCHIDA[†] and KOICHI WADA[†]

We focus on the problem that in an ad hoc network, how to send a message securely between two users using the certificate dispersal system. In this paper, we construct two efficient certificate dispersal algorithms. We can prove that for a strongly connected graph G=(V, E) and a directed graph H=(V', E'), new upper bounds on dispersability cost on the average number of certificates stored in one node are $O(D_G+|E|/|V|)$ and $O(pd_{max}+|E'|/|V'|)$ respectively, where D_G is the diameter of G, d_{max} is the maximum diameter of strongly connected components of H and p is the number of strongly connected components of H. Furthermore, we can prove our algorithms are optimal for several graph classes.

I. Introduction

The recent history of the Internet and of computer networks has shown that if security of a given network architecture is not properly designed from the very beginning, then the security breaches will be exploited by malicious users. Security in mobile ad hoc networks is particularly difficult to achieve, notably because of the vulnerability of the links, the limited physical protection of each node, the sporadic nature of connectivity, the dynamically changing topology, the absence of a certification authority, and the lack of a centralized monitoring or management point.

As such, our approach is developed mainly for "open" networks, in which users can join and leave the network without any centralized control. Therefore, it is necessary to establish distributed and secure authentication systems in mobile ad hoc networks. There is much research on secure authentication systems, such as in [5] it provided a system in which, if two nodes meet, they can communicate securely with high probability, and in [12] it guaranteed the probability 1.

The main problem of any public-key based security system is to make each user's public key available to others in such a way that its authenticity is verifiable. In mobile ad hoc networks, this problem becomes even more difficult to solve because of the absence of centralized services and possible network partitions. More precisely, two users willing to authenticate each other are likely to have access only to a subset of nodes of the network (possibly those in their geographic neighborhoods). One of the best known approaches to the public-key management problem is based on public-key certificates [6]. A public-key certificate is a data structure in which a public key is bound to an identity (and possibly to some other attributes) by the digital signature of the issuer of the certificate. The all certificates issued by nodes in a network can be represented by a directed graph, called certificate graph. Each node in a certificate graph represents a mobile user in the network, and each directed edge in a certificate graph represents a certificate. The certificate can be used by any node in the network that knows the public key of one node to further acquire the public key of the other node. The issued certificates are necessary to be dispersed among nodes in the network such that if a node u approaches another node v and wishes to securely send messages to v, then u can obtain the public key of v using certificates stored either in u or v in the certificate graph.

Several papers have investigated the use of certificates to provide security in traditional networks and in ad hoc ones. Architectures for issuing, storing, discovery, and validating certificates in traditional networks are presented in [1,2,4,7,8,9,11,13].

In [5], the authors investigated how to disperse certificates in a certificate graph in a mobile ad hoc network under the following two conditions. First, each node stores the same number of certificates. Second, with high probability, if two nodes meet then they have enough certificates for each of them to compute the public key of the other.

[†]名古屋工業大学
Nagoya Institute of Technology

In [12], it is proved that a lower bound on the number of certificate to be stored in a node is $\sqrt{n}-1$ which satisfies the following conditions: First, each certificate is stored in the same number of nodes. And secondly, the same number of the certificates is stored in each node, where n is the number of nodes in the system.

In [3], another condition, that every certificate must be stored in some node, is considered, and the dispersability cost of a certificate graph G, denoted c.G, is defined as the minimum average number of certificates stored in one node. Under this condition, a dispersal algorithm called Full Tree algorithm is presented, and the upper bound $c.G \leq n-1$ is proved, where n is the number of nodes in the graph. The lower bound $c.G \geq e/n$ is also proved for any certificate graph, where e is the number of edges in the graph. The authors have shown in [3] that there exist some special certificate graphs, such as a fully connected certificate graph such that $c.G=n-1=e/n$. Ring, hourglass and star graphs are also analyzed. The dispersability cost of the ring graph is n-1, which equals to the tight upper bound, (n-1)/n for hourglass and 2(n-1)/n for star graph, which equal to the lower bounds. And Half Tree algorithm, to which Full Tree algorithm is improved, is also presented. However the cost of Full Tree algorithm equals to n-1 for any strongly connected certificate graph. Half Tree algorithm is probably better than Full Tree algorithm, but the cost of Half Tree algorithm is equal to the cost of Full Tree algorithm for strongly connected graphs.

Our work is based on two conditions considered in [3]. First, different nodes may store different number of certificates, but the average number of certificates stored in one node is minimized. Secondly, it is guaranteed (i.e. with probability 1) that if two nodes meet then they have enough certificates for each of them to compute the public key of other.

In this paper, we construct two efficient certificate dispersal algorithms. And new upper bounds on dispersability cost for strongly connected certificate graphs and directed certificate graphs are proved. For a strongly connected graph G and a directed graph H, the upper bounds on the average number of certificates stored in one node are $2D_G+|E|/|V|$ and $2pd_{max}+p-1+|E|/|V|$ respectively, where D_G is the diameter of graph G, d_{max} is the maximum diameter of strongly connected components of H and p is the number of strongly connected components. For a given graph, an algorithm whose complexity of the cost is asymptotically equal to the complexity of the lower bound of the dispersability cost of the graph, is called an optimal algorithm. We can prove our algorithms are optimal for several graph classes, such as Hypercubes, Meshes, complete k-ary trees and de-Bruijn graphs.

II. PRELIMINARIES

For simplicity, we assume that each honest user owns a single mobile node. We consider a mobile ad hoc network, where each node u has a private key pri.u and a public key pub.u. Users themselves create their public and private keys. In this network, in order for a node u to send a message m to v securely, u need to know the public key of v to encrypt the message by pub.v, denoted by pub.v < m >.

If a node u knows the public key pub.v of another node v in the network, then node u can issue a certificate, called a certificate from u to v, that identifies the public key pub.v of node v. The certificate can be used by any node in the network that knows the public key of the node u to further acquire the public key of node v.

A certificate from node u to node v is of the following form: pri.u < u, v, pub.v >.

The certificate is encrypted using the private key pri.u of node u, and it contains three items: The identity of the certificate issuer u, the identity of the certificate subject v, and the public key of the certificate subject pub.v.

Any node who knows the public key of u can use pub.u to decrypt the certificate from u to v for obtaining the public key of node v.

When a node u wants to obtain the public key of another node v, it acquires a path of certificates such that:

1. The first certificate of the path can be directly verified by u, by using a public key that u holds and trusts.
2. Each of remaining certificates can be verified using the public key contained in the previous certificate of the path.
3. The last certificate contains the public key of the target node v.

A. Definitions

In this section, we define some terms used in this paper.

All certificates issued by nodes in a network can be represented by a directed graph, called a certificate graph, denoted by G=(V, E). Each node in the certificate graph represents a mobile user in the network. A directed edge from node u to v in the certificate graph represents a certificate from u to v. Note that according to this definition, a certificate graph is a directed graph that does not have self-loops and multiple edges between two nodes.

The issued certificates are necessary to be dispersed among nodes in the network such that if a node u approaches another node v and wishes to securely send messages to v, then u can obtain the public key of v using certificates stored either in u or v provided that there is a directed path from u to v in the certificate graph.

A directed path $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ in a certificate graph G, where the nodes v_0, v_1, ... and v_k are distinct, is called a certificate path from v_0 to v_k, here, v_0

and v_k are called end nodes. Note that if there exists a path from v_0 to v_k, then, v_0 can get the public key of v_k. For simplicity, we call certificate path as path in this paper. The length of a path $p(v_i, v_j)$ is the number of certificates in it, denoted by $|p(v_i, v_j)|$. A path from v_i to v_j is shortest iff its length is not larger than the length of any other paths from v_i to v_j in the certificate graph. A distance from v_i to v_j is the length of a shortest path from v_i to v_j, denoted by $d(v_i, v_j)$. Let c denote the path (v_0, v_1), (v_1, v_2), ... , (v_{k-1}, v_k). Then each of the paths (v_0, v_1), ... , (v_{j-1}, v_j), where $1 \le j \le k$, is called a prefix of c that ends at node v_j. Also, each of the paths (v_j, v_{j+1}), ... , (v_{k-1}, v_k), where $0 \le j \le k-1$, is called a suffix of c that starts at node v_j. Each of the paths (v_i, v_{i+1}), ... , (v_{j-1}, v_j), where $0 \le i < j \le k-1$, is called a subpath of c.

A graph, in which for any two distinct nodes u and v, there exists a path from u to v, is said to be strongly connected. And a graph is bi-directional if there is an edge from node u to node v then there exists an edge from v to u, and vice versa. For any node v in graph G, we define the eccentricity of v as the length of a longest shortest path from the node v to any other nodes. In a bi-directional graph G, for any node v in G, the minimum value of eccentricity is defined as radius of the graph, denoted by R_G. And in a directed graph G, the maximum length of shortest paths between any two nodes in G is defined as a diameter of the graph, denoted by D_G. A strongly connected component C of a directed graph G=(V, E) is a subgraph C=(V', E') (V'$\subseteq$V and E'$\subseteq$E) which satisfies that no node of G can be added to C such that it is strongly connected. A directed graph can be partitioned into strongly connected components. Note that this partition is unique.

Free tree T is connected undirected graph without cycle. A rooted tree is a free tree in which one of the nodes is distinguished from the others. The distinguished node is called the root of the tree. If the last edge on the path from the root r of a tree T to a node x is (y, x), then y is the parent of x, and x is a child of y. The root is the only node in T with no parent. A node with no children is an external node or leaf. A nonleaf node is an internal node. The number of children of a node x in a tree is called the degree of x. The length of the path from the root to a node x is the depth of x in T. The largest depth of any node in T is the height of T, denoted by h.

For a directed graph G, incoming tree T rooted at node v on G is a graph, whose nodes set are all of the nodes reachable to v in G and there is a unique path from any other node to v on T. A shortest incoming tree rooted at v is an incoming tree in which the path from any other node to v corresponds to a shortest path in G.

For a directed graph G, outgoing tree T rooted at node v on G is a graph, whose nodes set are all of the nodes reachable from v in G and there is a unique path from v to any other nodes on T. A shortest outgoing tree rooted at v is an outgoing tree in which the path from v to any other node corresponds to a shortest path in G.

Given a graph V=(V, E) and a distinguished source node v, breadth-first search is one of the algorithms for searching a graph that systematically explores the edges of G to discover every node that is reachable from v. It produces a breadth-first tree with root v that contains all such reachable nodes. For any node u reachable from v, the path in the breadth-first tree from v to u corresponds to a shortest path from v to u in G.

B. Certificate Dispersal Problem [3]

A certificate dispersal algorithm F is an algorithm that takes a certificate graph G as an input and outputs a subset of the certificates, which is stored in each node in G, denoted F.(G, v), such that the following two certificate dispersal conditions hold;

1) Connectivity:
   For every distinct pair of nodes u and v in G, if there is a path from u to v in G, then the certificates on the path are in the set $F.(G, u) \cup F.(G, v)$.
2) Completeness:
   For every certificate in G, there exists a node v in G such that this certificates is in F.(G, v).

Let G be a certificate graph, and let F be a certificate dispersal algorithm. The cost of F to disperse the certificates in G, which denoted by c.(F, G), is defined as follows:

$$c.(F, G) = \frac{1}{n} \sum_{v \in V} | F.(G, v) | ,$$

where n is the number of nodes in G, and |F.(G, v)| denotes the number of certificates in the set F.(G, v) assigned by F to node v. Note that c.(F, G) is the average number of certificates assigned by F to a node in G.

The dispersability cost of a certificate graph G, denoted c.G, is defined as follows:

$$c.G = \min_{F} \{c.(F, G)\} .$$

III. New Upper Bounds

In this section, we introduce a procedure Pivot first. Using Pivot procedure, we construct CPivot algorithm for strongly connected certificate graphs and GPivot algorithm for directed graphs later.

A new upper bound $O(D_G)$ for a strongly connected graph G and $O(pd_{max})$ for a directed graph H are proved, where D_G is the diameter of graph G, d_{max} is the maximum diameter of strongly connected components of H and p is the number of strongly connected components of H. In section IV, CPivot algorithm will be proved to be an optimal algorithm for Hypercubes, Meshes, complete k-ary trees and de-Bruijn graphs.

A. Pivot

Procedure Pivot takes a strongly connected directed graph G as an input. We sketch an outline of Pivot.

Arbitrarily, select one node as pivot node p. And then, for every node x ($x \neq p$), all of the certificates on two shortest paths in both directions between x and p are stored in x. Pivot is formulated as follows:

Procedure Pivot (G=(V, E))
1. Select an arbitrary node as pivot node p
2. For every node v in V – {p}
   2.1 Compute the shortest path from v to p, p(v, p) and the shortest path from p to v, p(p, v)
   2.2 Store $p(v, p) \cup p(p, v)$ to v

Lemma 1: Pivot satisfies one of the certificate dispersal conditions, connectivity, and its computation time is O(|E|).
Proof: Let G be a strongly connected graph, which is an input of Pivot. Since G is strongly connected, for every node $v(\neq p)$ in G, there exist paths p(v, p) and p(p, v). In Pivot, we stored all of the certificates on p(v, p) and p(p, v) to v. That means, for every distinct pair of nodes u and v, there exists a path from u to v via p in G, and the certificates on the path are in the sets of the certificates stored in u and v.

In the following, we prove the computation time. For computing two shortest paths p(v, p) and p(p, v), incoming and outgoing breadth-first trees rooted at pivot node p are needed. It completes in time of O(|V|+|E|). And the main loop consumes time of O(|V|). Since G is strongly connected, we have $|E| \geq |V|$. So the total computation time of Pivot is $O((|V|+|E|)+|V|) = O(|E|)$. □

B. Certificate dispersal algorithm CPivot

In order to construct certificate dispersal algorithm CPivot, we make a slight change to the Pivot. From Lemma 1, we know that Pivot satisfies connectivity. For satisfying the second condition, completeness, we store all unused certificates to the pivot node after running Pivot Procedure. Through this addition, Pivot can be changed to a certificate dispersal algorithm, which satisfying both of two conditions, connectivity and completeness. We call this algorithm as CPivot.

Lemma 2: CPivot is a certificate dispersal algorithm.
Proof: In CPivot algorithm, after running Pivot procedure, we stored all remaining certificates in pivot node p. That means all of the certificates are stored in some nodes, which satisfies completeness. And Pivot procedure satisfies connectivity. We can know that CPivot satisfies both of two conditions. □

Theorem 1: For any strongly connected certificate graph G, $c.G \leq 2D_G+|E|/|V|$.
Proof: CPivot consists of two phases, Pivot procedure and the additional storing operation. In Pivot procedure, it is clear that even if which node is selected as pivot node p, for any v, $|p(v, p) \cup p(p, v)| \leq 2D_G$. Therefore, at the end of Pivot procedure, the total number of certificates stored in all of the nodes is at most $2|V|D_G$. In the time of starting the additional storing operation, at most |E| certificates are remained. So, the cost of CPivot is $2D_G+|E|/|V|$. □

C. Pivot selection

How to select a pivot node p in Pivot procedure, determines the cost of the CPivot. We can choose a special node as pivot node p to decrease the cost. In this subsection, we introduce two efficient methods for selecting pivot node p, for strongly connected graphs and bi-directional graphs, respectively.

For a strongly connected graph, for each node x, compute incoming breadth-first tree T_{in} rooted at x and outgoing breadth-first tree T_{out} rooted at x. Select node x with the minimum value of $\sum_{y \in V(T_{out})} d(x, y) + \sum_{z \in V(T_{in})} d(z, x)$ as pivot node p.

Lemma 3: Pivot selection for strongly connected graphs is computed with $O(|V| \cdot |E|)$ time.
Proof: For any node incoming and outgoing breadth-first trees rooted at the node is computed in O(|V|+|E|) time. For G is strongly connected, we have $|E| \geq |V|$. So the total computation time is $O((|V|+|E|) \cdot |V|) = O(|V| \cdot |E|)$. □

For a bi-directional graph, for each node x, compute outgoing breadth-first tree T_{out} rooted at x. Select node x with the minimum value of $\sum_{y \in V(T_{out})} d(x, y)$ as pivot node.

Lemma 4: Pivot selection for bi-directional graphs is computed with $O(|V| \cdot |E|)$ time.
Proof: Outgoing breadth-first tree for any node is computed in time of O(|V|+|E|). Since G is bi-directional, we have $|E| \geq |V|$. So the total computation time is $O((|V|+|E|) \cdot |V|) = O(|V| \cdot |E|)$. □

Theorem 2: For any bi-directional certificate graph G, $c.G \leq 2R_G+|E|/|V|$.
Proof: In Pivot selection, we select a special node x as pivot node p, such that for any node y, $d(x, y) \leq R_G$. In Pivot procedure, for any node v, $|p(v, p) \cup p(p, v)| \leq 2R_G$. Therefore, at the end of Pivot procedure, the total number of certificates stored in all of the nodes is at most $2|V|R_G$. In the time of starting the additional storing operation, at most |E| certificates are remained. So, the cost of CPivot is $2R_G+|E|/|V|$. □

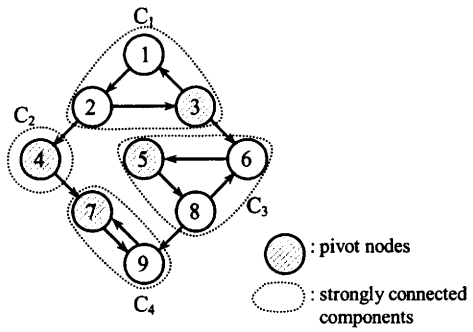
D. Certificate dispersal algorithm GPivot

In this subsection, GPivot algorithm, which takes a directed graph as an input, is presented.

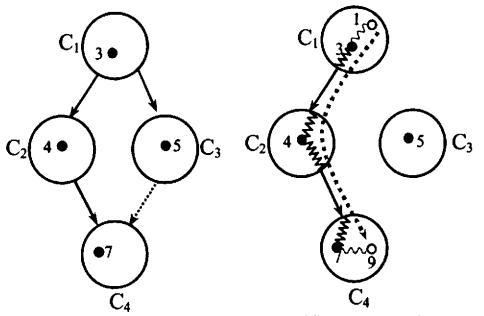
Algorithm GPivot formulated is as follows.

Algorithm GPivot (G=(V, E))

1. Partition G into strongly connected components $C_i=(V_i, E_i)$, i=1,2,...,p. Let p_i be pivot node of C_i.
2. For each C_i do Pivot(C_i)
3. Construct a graph H s.t. $V(H)=\{C_1,C_2,...,C_p\}$ and $E(H)=\{(C_i, C_j) \mid (v, w)\in E, v\in V(C_i)$ and $w\in V(C_j)\}$ Note that H contains no cycles (H is a directed acyclic graph).
4. For each node $C_i \in V(H)$
   4.1 Compute a shortest outgoing tree T_i rooted at C_i on H
   4.2 For each node $v\in V(C_i)$
      4.2.1 For each edge $(C_j, C_k)\in E(T_i)$
      Store the certificates on the shortest path $p(p_j, p_k)$ in G into v
5. Store all unused certificates to an arbitrary node



(a) an input graph G



(b) T_1 on H

(c) certificates stored in nodes 1 and 9

Fig.1. An example of GPivot. (a): an input graph G and pivot node of each strongly connected component. (b): induced graph H and a shortest outgoing tree rooted at C_1. (c): the certificates on the path p(1, 9) by concatenating p(1, 3), p(3, 4), p(4, 7) and p(7, 9).

An example of GPivot is shown in Fig.1.

A directed graph G is given as an input of GPivot. We partition G into four strongly connected components C_1, C_2, C_3 and C_4. For any reachable nodes pair, following certificates are stored in. As an example, we focus on the nodes 1 and 9 in different components. In step 2, through Pivot procedure, the certificates (1, 2), (2, 3) and (3, 1) are stored in node 1, and the certificates (9, 7) and (7, 9) are stored in node 9. And the induced graph H from G is shown in Fig.1 (b). A shortest outgoing tree T_1 rooted at C_1 on H is shown in Fig.1 (b), in which the edges of T_1 are denoted by solid line. A path from p_1 in C_1 to p_4 in C_4, which constructed in step 4 is shown in Fig.1 (c). The certificates (3, 1), (1, 2), (2, 4) and (4, 7) on the path p(3, 7) are stored in node 1. After running of GPivot(G), the certificates (1, 2), (2, 4), (4,7) and (7, 9) are stored in nodes 1 and 9, which satisfies connectivity.

Lemma 5: For any directed graph G, GPivot is a certificate dispersal algorithm, and its computation time is $O(p\cdot(|V|+|E|))$.

Proof: To prove that GPivot is a certificate dispersal algorithm, we show connectivity first. For any reachable pair u and v in G, we consider two different cases, whether u and v are in the same strongly connected component, or not. In the case which u and v are in the same component C_i, since C_i is strongly connected, there must exist a path from u to v. After running Pivot(C_i), all of the certificates on $p(u, p_i)$ and $p(p_i, v)$ are stored in u and v respectively, which satisfies connectivity. In the other case which reachable pair u and v are in different strongly connected components, assume that $u\in C_i$, $v\in C_j$ ($i\neq j$). The certificates on $p(u, p_i)$ is stored in u and the certificates on $p(p_j, v)$ is stored in v in step2. And in step 4.1, we construct a shortest outgoing tree T_i rooted at C_i. Since u and v are reachable, there must exist a path from C_i to C_j in T_i. Furthermore, in step 4.2 certificates on the shortest path from p_i to p_j are stored in u. We can know that for any reachable pair u and v, there exist a path p(u, v) which concatenating $p(u, p_i)$, $p(p_i, p_j)$ and $p(p_j, v)$, and all of the certificates on p(u, v) are stored in u and v, this satisfies connectivity. We show completeness second. In step 5, we store all remained certificates in an aribitrary node. That means all of the certificates are stored in some nodes, which satisfies completeness. Therefore, GPivot is a certificate dispersal algorithm.

For the proof of computation time, we consider the time needed for each step. In step 1, G can be partitioned into strongly connected components by Tarjan's algorithm [10] in $O(|V|+|E|)$ time. It takes $O(|V|+|E|)$ time for Pivot procedure in step 2, and $O(|E|)$ time is needed for constructing graph H in step 3. In step 4.1, we need $O(p+|E(H)|)$ time for computing shortest outgoing tree for each component, and in step 4.2, for constructing the shortest paths between two reachable pivot nodes, we need to construct shortest outgoing tree rooted at each pivot node, which needs $O(|V|+|E|)$ time. Therefore, we

can finish the computation in step 4 in time of $O(p\cdot(p+|E(H)|+|V|+|E|)) = O(p\cdot(|V|+|E|))$. In step 5, at most |E| certificates are remained, that means we need at most $O(|E|)$ time for step 5. Hence, the total computation time for GPivot is $O(p\cdot(|V|+|E|))$. □

Theorem 3: For any directed graph G, $c.(GPivot, G) \le 2pd_{max}+p-1+|E|/|V|$, where p is the number of strongly connected components of G and d_{max} is the maximum diameter of strongly connected components of G.

Proof: Let C_1, C_2,... and C_p be strongly connected components, d_i be diameter of C_i and $d_{max}=\max\{d_1,d_2,\ldots, d_p\}$. We consider the number of certificates stored in each step. In step 2, for each node v in C_i, at most $2d_i$ certificates are stored in v by Pivot procedure. In step 4, since T_i is a tree and $|V(H)| = p$, there are at most p–1 edges in T_i. For any edge (C_i, C_j) in T_i, $d(p_i, p_j)$ is at most $2d_{max}+1$. Therefore, $|GPivot.(G, v)|$ satisfies an inequality below:

$$|GPivot.(G,v)| = 2d_i + \Big|\bigcup_{(C_j,C_k)\in E(T_i)} p(p_j,p_k)\Big|$$

$$\le 2d_{max} + \sum_{(C_j,C_k)\in E(T_i)}(2d_{max}+1) \le 2d_{max}+(p-1)(2d_{max}+1)$$

$$\le 2pd_{max}+p-1$$

This inequality holds for any node v in G. In step 5, at most |E| certificates are stored in an arbitrary node. Therefore, the cost of G.Pivot is $2pd_{max}+p-1+|E|/|V|$. □

IV. Lower Bounds for Special Graph Classes

In this section, we prove some lower bounds of the dispersability cost for special graph classes, n-dimensional Hypercube, n-dimensional Meshs, complete k-ary trees and de-Bruijn graphs. And our CPivot algorithm is shown to be optimal algorithm for these graph classes.

Lemma 6: Let G = (V, E) be a strongly connected graph, V_1 and V_2 be subsets of V such that $V_1 \cap V_2 = \phi$. Let $f: V_1 \to V_2$ be an injective function and $P=\{p(u, f(u)) | u\in V_1$ and there exists a path from u to f(u)$\}$. Then

$$c.G \ge \max\left(\frac{|E|}{|V|}, \frac{1}{|V|}\sum_{p(u,v)\in P} d(u,v)\right).$$

Proof: For satisfying completeness condition, at least |E| certificates have to be stored. This means the average number of certificates stored in each node is not less than |E|/|V|. On the other hand, we assume that there exists a certificate dispersal algorithm F with $c.(F,G)<\sum_{(u,v)\in P} d(u,v)/|V|$. The total number of certificates stored in all of the nodes of G is c.(F, G)|V|. Let us consider the path set P defined in Lemma 6. For any path $p(u, f(u))\in P$, we need at least d(u, f(u)) certificates. f is an injection from V_1 to V_2, therefore, for any two distinct nodes u and v in V_1, f(u) and f(v) are different. So for satisfying connectivity, we need at least $\sum_{(u,v)\in P} d(u, v)$ certificates. Although, we only have c.(F, G)|V| certificates in G, that means there exists node x such that at least one certificate on path p(x, f(x)) that is not stored in any node. Here, F.(G, x) ∪F.(G, f(x)) is not enough to satisfy connectivity. Contradiction occurs. □

Hereafter we prove lower bounds on dispersability cost for some special graph classes by using Lemma 6.

A. Hypercubes

An n-dimensional Hypercube H_n (n-Hypercube) is defined as follows: the n-Hypercube H_n is an bi-directional graph with 2^n nodes which labeled with a set of binary strings of length n and there exist bi-directional edges between any two nodes, iff there is only one bit differ in their labels.

Theorem 4 : For any integer n, $c.H_n \ge n$.

Proof: Since in H_n, $|V|=2^n$ and $|E|=n2^n$, according to Lemma 6, $c.H_n \ge |E|/|V| = n$. □

Corollary 1: CPivot is an optimal algorithm for n-Hypercube.

Proof: From Theorem 2 and 4, we know that $c.(CPivot, G) \le 2R_G+|E|/|V|$ and $c.H_n \ge n$. For n-Hypercube, $R_G=n$ and $|E|/|V|=n$, so $n \le c.H_n \le 3n$. We get this corollary. □

B. Meshes

An n-dimensional Mesh M_k^n with side k (called (n, k)-Mesh) is defined as follows: the (n, k)-Mesh M_k^n is a bi-directional graph whose node set is strings of length n on alphabet of k integers 0, 1, ..., k–1 and there is an edge from $(a_1, a_2, \ldots, a_n)$ to $(b_1, b_2, \ldots, b_n)$ when there exists i such that for all $j\neq i$, $a_j=b_j$ and $a_i=b_i\pm1$. In (n, k)-Mesh, $|V|=k^n$ and $|E|=2nk^n-2nk^{n-1}$.

Theorem 5: For any integer n and k, $c.M_k^n \ge (k-1)^2n/4k$.

Proof: In the case that k is an odd number; Let $V_1=\{(a_1,a_2,\ldots,a_n)|a_1\in\{0,\ldots,(k-3)/2\},a_2,\ldots,a_n\in\{0,\ldots,k-1\}\}$, $V_2=\{(a_1,a_2,\ldots,a_n)|a_1\in\{(k-1)/2,\ldots,k-1\},a_2,\ldots,a_n\in\{0,\ldots,k-1\}\}$. We define an injective function $f: V_1\to V_2$ as follows. $f((a_1,a_2,\ldots,a_n))=(a_1+(k-1)/2, a_2+(k-1)/2 \bmod k, \ldots, a_n+(k-1)/2 \bmod k)$. It is clear that f is an injection from V_1 to V_2. For any path $p(x, f(x))\in P$, $P=\{p(x, f(x)) \mid x\in V_1\}$, $d(x, f(x))= n(k-1)/2$. Note that $|P|=k^{n-1}(k-1)/2=((k-1)/2k)|V|$.

In the case that k is an even number; Let $V_1=\{(a_1,a_2,\ldots,a_n)|a_1\in\{0,\ldots,k/2-1\}, a_2,\ldots,a_n\in\{0,\ldots,k-1\}\}$, $V_2=\{(a_1,a_2,\ldots,a_n)|a_1\in\{k/2,\ldots,k-1\}, a_2,\ldots,a_n\in\{0,\ldots,k-1\}\}$. We define an injective function $f: V_1\to V_2$ as follows. $f((a_1, a_2, \ldots, a_n))=(a_1+k/2, a_2+k/2 \bmod k, \ldots, a_n+k/2 \bmod k)$. It is clear that f is an injection from V_1 to V_2. For any path $p(x, f(x))\in P$, $P=\{p(x, f(x))|x\in V_1\}$, $d(x, f(x))=nk/2$. Note that $|P|=|V|/2$.

In both cases, since $|P| \ge ((k-1)/2k)|V|$, $d(x, f(x)) \ge n(k-1)/2$ and from Lemma 6,

$$c.M_k^n \ge \sum_{p(x,f(x))\in P} \frac{d(x,f(x))}{|V|} \ge \frac{k-1}{2k}\cdot\frac{n(k-1)}{2} = \frac{(k-1)^2 n}{4k}$$ □

Corollary 2: CPivot is an optimal algorithm for (n, k)-Mesh.

Proof: From Theorem 2 and 5, we know that c.(CPivot, G) $\le 2R_G+|E|/|V|$ and $c.M_k^n \ge nk/2$. For the (n, k)-Mesh, $R_G=nk/2$, $|E|=2nk^n-2nk^{n-1}$ and $|V|=k^n$, so $(k-1)^2n/4k \le c.M_k^n \le nk+2n-2n/k$. We have this corollary. □

C. complete k-ary tree

complete k-ary tree T_k is a free tree in which all leaves have the same depth and all internal nodes have degree k. We make all of the undirected edges in T_k bi-directional ones.

Theorem 6: For any integer k, let T_k^h be a complete k-ary tree of height h, $c.T_k^h \ge h-1$.

Proof: We number the nodes in the complete k-ary tree with breadth-first-order and sort the leaves in ascending order, say 1, 2, ..., kh. Note that there are kh leaves in a complete k-ary tree. $V_1=\{x\in T_k^h \mid 1 \le x \le \lfloor k^h/2 \rfloor\}$, $V_2=\{x\in T_k^h \mid \lfloor k^h/2 \rfloor+1 \le x \le k^h\}$

We define an injective function f: $V_1 \to V_2$ as follows. $f(x)=x+\lfloor k^h/2 \rfloor$. It is clear that f is an injection from V_1 to V_2. For any $p(x, f(x))\in P$, $P=\{p(x, f(x))|x\in V_1\}$, d(x, f(x))=2h.

From lemma 6, $|V|=(k^{h+1}-1)/(k-1)$ and $|P|=\lfloor k^h/2 \rfloor$,

$$c.T_k^h \ge \sum_{p(x,f(x))\in P} \frac{d(x,f(x))}{|V|} = \frac{k^h}{2}\cdot 2h\cdot\frac{k-1}{k^{h+1}-1} = \frac{h\left(k^{h+1}-k^h\right)}{k^{h+1}-1}$$
$$\ge h-1 = \Omega(h)$$ □

Corollary 3: CPivot is an optimal algorithm for complete k-ary tree.

Proof: From Theorem 2 and Theorem 6, we know that $c.(CPivot, G) \le 2R_G+|E|/|V|$ and $c.T_k^h \ge h-1$. For complete k-ary tree, $R_G=h$ and $|E|/|V|=2(|V|-1)/|V|\le 2$, so $h-1 \le c.T_k^h \le 2h+2$. We have this corollary. □

D. de-Bruijn Directed Graph

We need to prepare some notations before defining de-Bruijn directed graph. For two sets A and B, let $A\times B=\{(a,b) \mid a\in A, b\in B\}$. Let $Z_n=\{0, ..., n-1\}$, and for any positive integer k, $Z_n^{k+1} = Z_n\times Z_n^k$.

A de-Bruijn directed graph B(n, d) (called (n,d)-de-Bruijn) is a graph whose nodes set is the strings of length d on an alphabet of n integers 0,1,...,n–1, i.e. $V(B(n, d)) = Z_n^d$. For any x, $y\in V(B(n, d))$, there is an edge from x to y if the d-1 first letters of y are equal to the d-1 last letters of x. That is, there is an edge from $(x_1, x_2, ..., x_D)$ to all the vertices $(x_2, ..., x_D, p)$, where p is any letter of the alphabet (shifting property). It is easy to check that its diameter is d (from a given node, all nodes are attained after d shifting and there exists a pair of nodes away from d). Its number of nodes is n^d and its number of edges is n^{d+1}. For any node $v=v_1, v_2, ..., v_d$ in B(n, d), let $s_i(v)=v_{d-i+1}, ..., v_d$, $p_i(v)=v_1v_2...v_i$. Define that $k^{u,v} = \max\{i \mid s_i(u)=p_i(v)\}$ and $(u^{-1})^+=u_d^+u_{d-1}^+...u_1^+$ and $u_i^+=u_i+1$.

Lemma 7: For any two nodes $u,v\in Z_n^d = V(B(n, d))$, the distance from u to v is $d-k^{u,v}$.

Proof: Let $u = u_1u_2...u_d$, $v = v_1v_2...v_d$ and d(u, v) = s. Shifting $d-k^{u,v}$ times, u's id can be changed to v's id. So $d(u, v) \le d-k^{u,v}$. u's id changes to the form $u_{s+1}u_{s+2}...u_dx_1x_2...x_s$, where x_i is an arbitrary letter after shifting s times. Since $u=v$, $u_{s+1}u_{s+2}...u_d=v_1v_2...v_{d-s}$, that is $s_{d-s}(u) = p_{d-s}(v)$. Because $k^{u,v}$ is maximum, $d-s \le k^{u,v}$. Therefore, $s=d(u,v)=d-k^{u,v}$. □

Lemma 8: For any node $u = u_1u_2...u_d \in Z_{n-1}^d \subset Z_n^d = V(B(n, d))$, $d(u, (u^{-1})^+)=d$.

Proof: We show that for any $k(1\le k \le d)$, $s_k(u)\ne p_k((u^{-1})^+)$. If k=1, then $s_1(u)=u_d$ and $p_1((u^{-1})^+)=u_d^+$, since $u_d \ne u_d^+$. For any k larger than 1, $s_k(u) = u_{d-k+1}u_{d-k+2}...u_d$, $p_k((u^{-1})^+)=u_d^+u_{d-1}^+...u_{d-k+1}^+$. If $u_d \le u_{d-k+1}$, then since $u_d\le u_{d-k+1}<u_{d-k+1}^+$, the last letters differ. If $u_{d-k+1}< u_d$, then $u_{d-k+1}<u_d<u_d^+$, the first letters differ. □

Theorem 7: For any integer n and d, $c.B(n,d)\ge d(n-1)/2n$.

Proof: From lemma 8, for any node in Z_{n-1}^d, there exists a node away from d. So we take $V_1=\{v\in Z_{n-1}^d \mid (v^{-1})^+\notin Z_{n-1}^d\}$. V_1 can be computed through an algorithm below:

1. $V_1 = \varnothing$, $V=Z_{n-1}^d$
2. Repeat
   Take a node v from V arbitrary
   $V_1=V_1\cup\{v\}$
   If $(v^{-1})^+ \in V$ then $V=V-\{(v^{-1})^+\}$
   Until $V = \varnothing$ or $|V|=1$
   Then it is clear that $|V_1| \ge |Z_{n-1}^d|/2$.

From lemma 6,

$$c.B(n,d)\ge \frac{1}{|V(B(n,d))|}\sum_{v\in V_1} d(v,(v^{-1})^+) = \frac{d(n-1)}{2n} = \Omega(d)$$ □

Corollary 4: CPivot is an optimal algorithm for (n, d)-de-Bruijn graph.

Proof: From Theorem 1, we know that $c.(CPivot, G) \le 2D_G+|E|/|V|$. And from Theorem 7, we know that $c.B(n, d) \ge d(n-1)/2n$. Since for de-Bruijn graph, $D_G=d$ and $|E|/|V|=n$, so $\max(d(n-1)/2n, n) \le c.B(n, d) \le 2d+n$. We have this corollary. □

V. Conclusions

In this paper, we proposed certificate dispersal problem under two conditions connectivity and completeness. New upper bounds on the cost of the certificate dispersability for strongly connected graphs and directed graphs are given. CPivot algorithm is

constructed for strongly connected graphs, and GPivot is constructed for directed graphs. For some special graph classes such as n-dimensional Hypercube, n-dimensional Mesh, complete k-ary tree and de-Bruijn graph, our algorithms are proved to be optimal.

Further research is needed on the upper bounds using other parameters. The upper bounds and the lower bounds of certificate dispersability cost for some other graph classes are necessary to be studied. It will be possible to construct some other certificate dispersal algorithms with lower cost for general directed graphs. We are also interested in the problem that what kind of certificate graphs have lower dispersability cost.

REFERENCES

[1] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "SPKI certificate theory," RFC 2693, 1999.

[2] D. Clarke, J.-E. Elien, C. Ellison, M.Fredette, A. Morcos, and R.L. Rivest, "Certificate chain discovery in SPKI/SDSI, "*Journal of Computer Security*, vol. 9, no. 4, pp. 285-322, 2001.

[3] Eunjin Jung, Mohamed G. Gouda, "Certificate Dispersal in Ad Hoc Networks," *Proceedings of the 24th annual International Conference on Distributed Computing Systems* (2004)

[4] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti, "dRBAC: distributed role-based access control for dynamic coalition environments," in *Proceedings. 22nd International Conference on Distributed Computing Systems*, 2002, pp. 411-420.

[5] Jean-Pierre Hubaux, Levente Buttyan, and Srdan Capkun, "The quest for security in mobile ad hoc networks," in *Proceedings of the 2001 ACM International Symposium on Mobile ad hoc networking & computing*. 2001, pp. 146-155, ACM Press.

[6] L.M. Kornfelder, "Toward a Practical Public-Key Cryptosystem," bachelor's thesis, Dept. Electrical Eng., Massachusetts Inst. of Technology, Cambridge, 1978.

[7] M. Blaze, J. Feigenbaum, J. Ioannididis, and A. Keromytis, "The keynote trust-management system version 2," RFC 2704, 1999.

[8] M. Myers, R. Ankney, A Malpani, S. Galperin, and C. Adams, "X.509 internet public key infrastructure online certificate status protocol – OCSP," RFC 2560, 1999.

[9] Ninghui Li, William H. Winsborough, and John C. Mitchell, "Distributed credential chain discovery in trust management: extended abstract," in *Proceedings of the 8th ACM conference on Computer and Communications Security*. 2001, pp. 156-165, ACM Press.

[10] Robert E. Tarjan, "Depth first search and linear graph algorithms," SIAM Journal on Computing, 1(2):146-160, 1972

[11] Ronald L. Rivest and Butler Lampson, "SDSI – A simple distributed security infrastructure," Presented at CRYPTO'96 Rumpsesseion, 1996.

[12] Srdjan Capkun, Levente Buttyan, and Jean-Pierre Hubaux, "Self-organized public-key management for mobile ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 2, no. 1, pp. 52-64, 2003.

[13] S. Boeyen, T. Howes, and P. Richard, "Internet X.509 public key infrastructure operational protocols – LDAPv2," 2559, 1999.