A Scalable Key Management and Clustering Scheme for Ad Hoc Networks

Jason H. Li, Renato Levy
Intelligent Automation, Inc.
15400 Calhoun Drive, Ste 400
Rockville, MD, 20855, USA
{jli,rlevy}@i-a-i.com

Miao Yu, Bobby Bhattacharjee
Department of Mechanical Engineering
Department of Computer Science
College Park, MD, 20742, USA
mmyu@glue.umd.edu, bobby@cs.umd.edu

Abstract

This paper describes a scalable key management and clustering scheme for secure group communications in ad hoc and sensor networks. The scalability problem is solved by partitioning the communicating devices into subgroups, with a leader in each subgroup, and further organizing the subgroups into hierarchies. Each level of the hierarchy is called a tier or layer. Key generation, distribution, and actual data transmissions follow the hierarchy. The Distributed, Efficient Clustering Approach (DECA) provides robust clustering to form subgroups, and analytical and simulation results demonstrate that DECA is energy-efficient and resilient against node mobility. Comparing with most other schemes, our approach is extremely scalable and efficient, provides more security guarantees, and is selective, adaptive and robust.

1 Introduction

Multicasting, as an efficient communication mechanism for delivering information to a large group of recipients, has led to the development of a range of powerful applications in both commercial and military domains. Key management serves as the crucial foundation to enable such secure group communications. However, the large size of the serving group, combined with the dynamic nature of group changes, pose a significant challenge on the scalability and efficiency on key management research.

Communication between arbitrary endpoints in an *ad hoc network* typically requires routing over multiple-hop wireless paths due to the limited wireless transmission range. Without a fixed infrastructure, these paths consist of wireless links whose endpoints are likely to be moving independently of one another. Given the potentially large number of mobile devices, scalability becomes a critical issue. In particular, *wireless sensor networks (WSNs)* [1] comprise of a higher number of nodes scattered over some region. Sensor nodes are typically less mobile, heavily resource-constrained, irreplaceable, and become unusable after failure or energy depletion. It is thus crucial to devise novel energy-efficient solutions for topology organization and routing that are scalable, efficient and energy conserving in order to increase the overall network longevity.

In our work, the scalability problem is solved by partitioning the communicating devices into subgroups, with a leader in each subgroup, and further organizing the subgroups into hierarchies. Each level of the hierarchy is called a *tier* or *layer*. Key generation, distribution, and actual data transmissions follow the hierarchy. Communications are generally restricted within a subgroup at a tier. Further, we describe an innovative clustering approach to organize devices into subgroups.

Clustering protocols have been investigated for ad hoc and sensor networks [8][10][16]. While these strategies differ in the criteria used to organize the clusters, clustering decisions in each of these schemes are based on static views of the network topology; none of the proposed schemes, even equipped with some local maintenance schemes, is satisfactorily resistant to node mobility beyond trivial node movement. One of the purposes of this work is to propose a clustering protocol that is resilient against mild to moderate mobility where each node can potentially move.

In the hybrid energy-efficient distributed clustering approach (HEED) [17], clusterhead selection is primarily based on the residual energy of each node. The clustering process entails a number of rounds of iterations; each iteration exploiting some probabilistic methods for nodes to elect to become a clusterhead. While HEED is one of the most recognized energy-efficient clustering protocols, we argue that its performance can be further enhanced. In this work, we will present a distributed, energy-efficient clustering approach (DECA). The protocol terminates without rounds of iterations as required by HEED, which makes DECA a less complex and more efficient protocol. In summary, our approach has the following advantages.

Security. We guarantee that neither a passive nor an ac-

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

INFOSCALE '06. Proceedings of the First International Conference on Scalable Information Systems, May 29-June 1 2006, Hong Kong
© 2006 ACM 1-59593-428-6/06/05...$5.00

tive adversary can discover any other subgroup keys that do not belong to them. Further, our scheme can protect the equally sensitive information about *group dynamics*, while most current key management schemes are vulnerable to the group dynamics attacks.

Scalability. The approach addresses the scalability problem by applying the divide-and-conquer principle to organize a multicast group into a hierarchy of subgroups and distribute the functionality of the key management service among the subgroups. The non-overlapping nature of the subgroups ensures that the subgroup multicasts occur in parallel and traverse disjoint parts of the delivery hierarchy, which makes the scheme extremely scalable.

Efficiency. The approach is efficient in terms of complexity of re-keying operation during a member join or leave event and key storage requirement. Further, the DECA protocol renders more robust and energy-efficient clustering. Such efficiency nicely supports those group members that only possess equipments with limited capability.

Selective. Unlike current key management schemes, our approach provides the capacity for selective communication between group members. Such selectivity will apply naturally in many military and commercial situations.

Robust and adaptive. The approach can handle multiple member changes, such as subgroup partition and merge. Moreover, our DECA scheme is robust against mild to moderate node mobilities.

The rest of the paper is organized as follows. We present the multi-tiered key management scheme in Section 2, followed by the description of the efficient clustering protocol in Section 3. We discuss related work in Section 4, and conclude the paper in Section 5.

2 Multi Tiered Key Management (MTKM)

The basic ideas of multi-tiered key management are adapted from a previous work by one of the authors [2], where the cluster size is bounded between k and $2k-1$ for some integer k, and better scalability over flat architectures has been achieved. In this work, we loosen the cluster size constraints and seek for insights on how we should deploy the promising ideas of hierarchies and subgroups to actual military and civilian applications. For completeness, we describe the core ideas and procedures briefly, followed by security and performance analysis. We use simple examples throughout the paper to illustrate concepts and ideas.

2.1 Description of the Technical Approach

2.1.1 Member Hierarchy for Key Distribution

Our key distribution scheme creates a member hierarchy. A tier or layer comprises of a set of members of the secure multicast group in the same level of the hierarchy. Layers are numbered sequentially with the lowest layer of the hierarchy being layer zero (denoted by L_0).

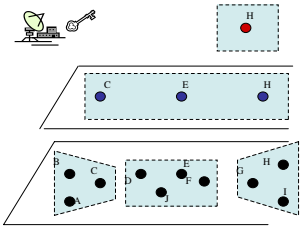

Figure 1. Initial arrangement of members.

A set of members in L_0 can form a subgroup. The size of each subgroup is restricted by a lower bound and an upper bound. Each layer has one lower and upper bound that should be used across all the subgroups in that layer; different layers can have different set of bounds. There can be multiple subgroups in each layer. However, the subgroups need to be disjoint, preferably in the sense of spatial multicast delivery path, at each layer. This will provide the maximum extent of parallelism, making the communications of keying materials and data transmissions extremely scalable.

Within each subgroup, there is a leader that will take the responsibility of key generation and distribution for that subgroup. The subgroup leaders of all the subgroups in layer L_i join layer L_{i+1}. As shown in Fig. 1, all ten members A-J are part of layer L_0, which has been partitioned into three subgroups: [ABC], [DEFJ], and [GHI]. The subgroup leaders, C, E and H join layer L_1. In layer L_1 only single subgroup [CEH] is formed. The leader, H, of the layer L_1 subgroup joins layer L_2 —the highest layer in this example. The procedure terminates when there is only a single member in any layer. Members of each layer of the subgroup hierarchy consist of subgroup leaders from the immediate lower layer. Similarly, when a subgroup leader is demoted in a certain layer L_j, it needs to be removed from all the higher layers, L_i, $i > j$ that it belongs to.

2.1.2 Layer Keys and Subgroup Keys

A secret layer key is associated with each layer of the hierarchy. A group member possesses a layer key for a specific layer if and only if it is a member of a subgroup in that layer. Layer keys are generated, on-demand, by a key server whenever the layer key needs to be changed (e.g. member joins or leaves any layer). A secret subgroup key is associated with each subgroup. Once again, group member possesses a subgroup key for a specific subgroup if and only if it is a member of that subgroup. The leader of each subgroup is responsible for generating the subgroup key for that subgroup. Finally, in all subgroups, a pair-wise key is shared between the subgroup-leader and each subgroup member.

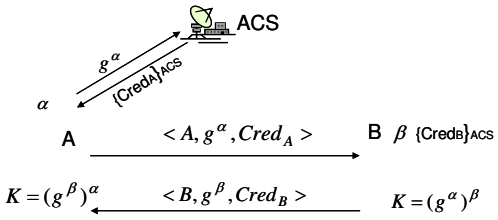

Figure 2. A Variant of Diffie-Hellman Protocol

2.1.3 Authentication and Access Control

Secure multicast typically requires a single entity where access to the group is controlled. We call this entity the Authentication and Access Control Server (ACS).

When a new member, A, joins the secure multicast group, it registers and authenticates itself with the ACS. The ACS maintains the authentication list for all the members of the whole group. As part of the registration, A acquires a time-stamped credential $Cred_A$ from the ACS, which is a digital certificate signed by the ACS. Such a credential should also contain an expiration time, after which A will have to leave the group, or stay via another registration.

Subsequently, when A joins a subgroup with leader B, A and B exchange the credentials to mutually authenticate each other and establish the pair-wise key between them. In this work, the members establish the pair-wise key using a computationally less expensive variant of the Diffie-Hellman key exchange protocol [11] by leveraging the authentication provided by the ACS, as shown in Fig. 2.

2.1.4 Key Distribution Protocol

We assume that the subgroups have been created in some appropriate manner. The key distribution protocol ensures that the layer key is only available to the members joined to that layer. Similarly, each subgroup has a secret subgroup key, known to only all the subgroup members.

I. Notation and terminology

- Members and Member Sets
  - S: The key server for all layer keys.
  - ACS: Authentication and access control server.
  - $SG(u, j)$: Subgroup of layer L_j, to which member u belongs.
  - $LD(u, j)$: Leader of the subgroup in layer L_j to which member u belongs.
  - $LD_g(j)$: Leader of the subgroup g in layer L_j.
  - UB_j: Upper bound of subgroup size in layer L_j. LB_j: Lower bound of subgroup size in layer L_j.
  - u, v: Members of the secure multicast subgroup.

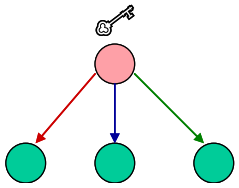

Figure 3. Subgroup Re-keys

- Keys and Messages
  - $K_G(t)$:The secret key of G at time t, where G is a set of members. If G is a subgroup, then this is the subgroup key, if G is a layer, then this is the layer key. If G is a pair of members, then this is a key shared only by these two members.
  - $\{m\}_e$: Message m is encrypted by the key e.
  - $\langle Unicast :: u \rightarrow v : x \rangle$: u sends a unicast message x to v.
  - $\langle Multicast :: u \rightarrow G : x \rangle$: u multicasts message x to a set of members G, where G is either a subgroup or a layer.

II. Distributed re-keying operations

- **Subgroup re-keys.** For a subgroup g in layer j, the leader $LD_g(j)$ obtains a new subgroup key $K_g(t+1)$ and unicasts it to each member of the subgroup encrypted separately by the pair-wise key of the leader with each member, as shown in Fig. 3. Let $K_p(v)$ represents the pair-wise key between a subgroup member v and its leader $LD_g(j)$, and the operation is:

$$\forall v \in g, \langle Unicast :: LD_g(j) \rightarrow v : \{K_g(t+1)\}_{K_p(v)} \rangle. \quad (1)$$

- **Layer re-keys.** The key server S generates a new layer key for layer L_j, and multicasts it to all members of layer L_{j+1}. These are the subgroup-leaders of layer L_j. Each subgroup leader of layer L_j then performs a subgroup multicast to all the members of its subgroup in layer L_j, as shown in Fig. 4.

$$\langle Multicast :: S \rightarrow L_{j+1} : \{K_{L_j}(t+1)\}_{K_{L_{j+1}}(t)} \rangle \quad (2)$$

$$\forall v \in L_{j+1}, \ \langle Multicast :: v \rightarrow SG(v, j) : \{K_{L_j}(t+1)\}_{K_{SG(v,j)}(t+1)} \rangle. \quad (3)$$

Note that $K_{SG(v,j)}(t+1)$ is the most updated subgroup key for the subgroup containing v.

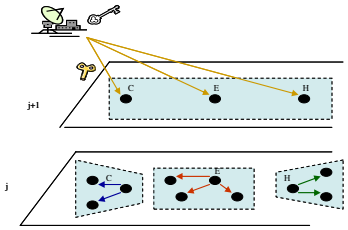

Figure 4. Layer Re-keys

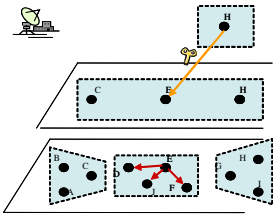

Figure 5. Selective transmission

III. Re-keying algorithm for member joins and leaves

When a new member joins the secure multicast group, it is inducted into some L_0 subgroup. When a member leaves the group, it leaves from all the layers it was joined to. Upon a membership change, the re-keying algorithm does the following: for each affected subgroup, do **Subgroup re-keys**, and then for each affected layer, do **Layer re-keys**.

2.1.5 Data Transmissions

Top-down data transmission Suppose the top leader residing at layer L_j wants to transmit data to the whole group. First, the data gets encrypted using the subgroup key of which it belongs at the immediate lower layer, i.e. L_{j-1}, and then it is multicast to the subgroup. Second, each member of the subgroup in layer L_{j-1} receives the data, which will in turn re-multicast the data to the lower layer subgroups using corresponding subgroup keys. This process continues until everyone receives the data.

In addition, the top layer leader can *selectively* communicate with members at any layer. Such messages can also reach all the layers below, without sending messages to every member in the whole group. For example, as shown in Fig. 5, H can selectively send data only to E and subgroup [DEJF]. Such a scenario naturally matches many military and commercial communication cases.

Peer-to-peer data transmission It is easy to observe that our scheme also supports peer-to-peer communications in the same subgroup. A member in some subgroup simply encrypt the data using the shared subgroup key. Only the members in the same subgroup can correctly receive the data. In this case, the sending party needs not to be the leader. Consequently, peer-to-peer data transmissions will be limited within the subgroup; no re-multicast will happen.

2.2 Security Analysis

General Security Analysis Let A be a *passive adversary*, who is never a subgroup member. We assume A eavesdrops on all traffic in an arbitrary subgroup and receives all the encrypted key information and data packets. A brute-force attack to find the group key takes $O(2^k)$ operations where k is the length of the key–A cannot do better than this. In addition, A cannot construct the pair-wise key, K, by monitoring the network traffic and acquiring the transmitted values g^α and g^β without knowing the values of α and β.

Let B be an *active adversary*, who has been a member of some subgroup during some previous time period. In our protocol, when B joins a subgroup, it cannot derive any previous group key by doing better than exhaustive search, i.e. $O(2^k)$ operations. Now assume B leaves the group and tries to read the subgroup traffic after it has left. B has with it the old pair-wise key with the subgroup leader, and possibly a set of layer keys. However, it cannot read the subgroup traffic at a later time, since the scheme updates all the keys that B previously knows per our re-key operations.

Group Dynamics Security We collectively refer to *group dynamics information (GDI)* [13] as information describing the dynamic membership of a multicast group, such as the number of users as a function of time, and the number of users who join or leave the service during a time interval. In many group communications, group dynamics information is confidential and should not be disclosed.

In most tree-based key distribution scheme [12][14][15], group members can distinguish the key updating process due to user join and that due to user departure, and rekey message size is closely related with the group size. As a result, attackers can estimate the number of joins and leaves by examining the rekey processes, and estimate the number of members from the rekey message size.

In our scheme, however, the GDI can be protected. Since the subgroup leaders establish keys for the subgroup members through pair-wise key exchange, the subgroup members cannot even obtain GDI of its own subgroup, let alone other subgroups at other hierarchy. Our scheme is essentially immune to the GDI attacks because, even with bulk rekeys, the rekeying messages (or their sizes) do not contain any distinguishing information that would divulge GDI to a corrupt insider. Note that the subgroup leaders naturally obtain the dynamic membership information of their subgroup and all subgroups below its layer. However, it can be shown that the probability of an attacker being promoted high in the layer hierarchy is exponentially small.

2.3 Performance Analysis

We generalize the analysis of the previous work [2] to various cluster sizes and summarize our results here without detailed analysis.

Small Cluster Sizes The minimum cluster size that can be used (while still preserving the re-keying guarantees) is two. For very small cluster sizes, the intra-cluster overhead decreases, but the number of layers increases. The increased number of layers results in higher processing at the key server, and may lead to higher overhead in terms of cluster reconfiguration. In the worst case, the minimum cluster size is two, and clusters split as soon as they have more than three members, i.e. the maximum cluster size is three.

Theorem 2.1 For groups with small cluster sizes:

- The number of layers is exactly $\lceil \log_2 N \rceil$.
- The amortized communication cost of a member joining/leaving the group is bounded by $2c+4+O(\frac{log_2 n}{n})$.
- The average number of keys stored by a member is **4.**
- The amortized cost of symmetric key processing due to a member join/leave is $\leq$ **2**. The average asymmetric key processing cost is less than **1.**

Large Cluster Sizes When the minimum cluster size is relatively large (say $\geq$ 10), the number of layers is correspondingly small. Since the large minimum cluster size forces a larger maximum cluster size (e.g. at least 20), there is relatively low inter-cluster overhead from individual joins and leaves. Further, with higher probability, the changes are restricted to a small number of layers. However, the intra-cluster cost is high, since each change to a cluster requires all cluster members to be rekeyed. Let the minimum cluster size be k, and that the largest cluster be of size C, we have:

Theorem 2.2 For large groups:

- The number of layers is at most $\lceil \log_2 N \rceil$.
- The amortized communication cost of a member joining/leaving the group is bounded by $O(C)*O(\frac{klog_k n}{n})$.
- The average number of keys stored by a member is $2 + \frac{C}{k-1}$.
- The amortized cost of symmetric key processing due to a member join/leave is **less than 2** . The average asymmetric key processing cost is **less than 1** .

The cluster reconfiguration cost depends entirely on the dynamic sequence of joins and leaves to the multicast group. Without a closed form description of the join/leave dynamics, it is impossible to analytically quantify the effect of the join/leave dynamics on the protocol overhead. Thus we seek to simulations to analyze such scenarios.

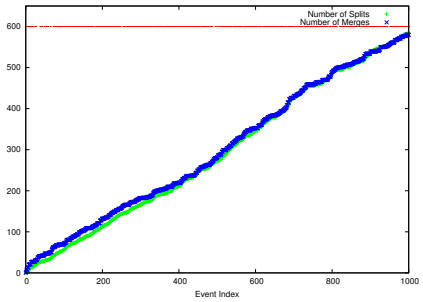

Figure 6. Maximum cluster size 8.

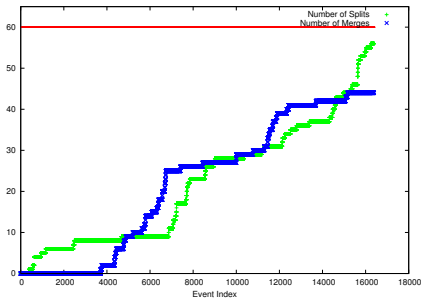

Figure 7. Maximum cluster size 24.

2.4 Simulation Results

2.4.1 Simple Cluster Dynamics

We first experiment with simplistic dynamic join and leave sequences to get a basic understanding of how the protocol behaves when increasing the maximum cluster size. Here, 256 nodes join the group initially, and then we simulate join (and leave) events, with each event chosen uniformly at random. During each event, between 1-4 nodes either join or leave. The clustering is recomputed after each event, and we repeated the experiment for different maximum cluster sizes. We use 4 as the minimum cluster size in all experiments, unless otherwise noted. In Fig. 6, we plot the number of merges and splits for maximum cluster size 8, over 1000 events. With the cluster bounds set to 4 and 8, there is a steady set of cluster merges and splits as nodes join and leave. Note further that the number of merges and splits closely track each other. In Fig. 7, we plot the results from the same experiment when the maximum cluster size is increased to 24. We observe that the number of cluster merges and splits have reduced by more than two orders of magnitude (this run required 16K events to produce around 50 splits and merges whereas with the cluster size set to 8, 1000 events produced over 500 splits and merges!).

2.4.2 Batch Updates

We consider a somewhat more representative scenario in this section. In each experiment, the system has a “batch

size" B. As before, we consider a sequence of join and depart events. Each event is either a join or a leave (uniformly at random), and the joins and leaves are arbitrarily interleaved. After B events, the hierarchy is recomputed.

Table 1. Batch updates.

Batch Size	Min. Cl. Size	Max. Cl. Size	Num. Merges	Merge Total	Num. Splits	Split Total
1	4	8	2396	19130	2480	20925
10	4	8	2218	17684	2292	19466
100	4	8	1310	10044	1356	11738
1000	4	8	282	2050	386	3513
1	4	12	188	1852	246	3012
10	4	12	219	2189	276	3387
100	4	12	165	1693	233	2913
1000	4	12	66	658	118	1538
1	4	16	55	636	94	1543
10	4	16	59	717	93	1530
100	4	16	38	457	89	1481
1000	4	16	9	104	54	920
1	4	32	0	0	22	704
10	4	32	0	0	20	640
100	4	32	0	0	19	612
1000	4	32	0	0	17	544
1	8	16	752	12857	780	13533
10	8	16	793	13527	820	14234
100	8	16	446	7470	475	8283
1000	8	16	94	1456	120	2108
1	8	24	38	819	60	1482
10	8	24	50	1072	70	1722
100	8	24	23	500	40	993
1000	8	24	9	187	27	678
1	8	32	7	190	23	737
10	8	32	3	74	25	803
100	8	32	3	86	19	611
1000	8	32	1	23	18	576
1	32	64	78	5505	86	6028
10	32	64	29	2111	37	2633
100	32	64	23	1663	30	2124
1000	32	64	5	330	12	801
1	32	96	1	72	5	480
10	32	96	1	73	5	480
100	32	96	2	140	5	480
1000	32	96	2	148	5	480
1	32	128	0	0	4	512
10	32	128	0	0	4	512
100	32	128	0	0	4	512
1000	32	128	0	0	4	512

In Table 1, we present the results from our batch experiments. Here, 512 nodes initially join the system (and form the hierarchy), and then 10,000 join (or leave) events are simulated. The batch size is shown in the first column, and the rest of the columns are similar to the previous results. The results motivate the following observations:

- Batch updates clearly reduce the protocol overhead. When the maximum cluster size is relatively small (twice the minimum cluster size), then batching 100 updates can often reduce overhead by 50% or more.
- Batch updates are most useful when the maximum cluster size is small. Larger maximum sized clusters insulate the effects of batch updates since each cluster can "absorb" more joins.
- As the minimum cluster size is increased (and the maximum cluster size is not very close to the twice the minimum cluster sizes), the effect of both batching and increasing cluster sizes is minimal.

Thus, to minimize *dynamic* overhead, the minimum cluster size is a more important parameter, and has more effect than maximum cluster size and batching updates. If the minimum cluster size cannot be increased, then the maximum cluster size should be at least thrice the minimum cluster size, or updates should be batched.

3 Distributed Efficient Clustering Approach

3.1 Problem Statement

An ad hoc wireless network is modeled as a set V of nodes that are interconnected by a set E of full-duplex directed communication links. Each node has a unique identifier and has at least one transmitter and one receiver. Two nodes are neighbors and have a link between them if they are in the transmission range of each other [5]. Nodes within the ad hoc network may move at any time without notice, it is our goal that the clustering protocol can still generate decent clusters under such mobility.

Let the clustering duration T_{C} be the time interval taken by the clustering protocol to cluster the network. Let the network operation interval T_{O} be the time needed to execute the intended tasks. In many applications, $T_{\mathrm{O}} >> T_{\mathrm{C}}$. In general, nodes that travel rapidly in the network may degrade the cluster quality because they alter the node distribution in their clusters and make the clusters unstable, possibly long before the end of T_{O}. However, research efforts on clustering should not be restricted only within the arena of static or quasi-stationary networks where node movements are rare and slow. Rather, for those applications where T_{O} is not much longer than T_{C}, we propose an efficient protocol that generates clusters in *ad hoc networks* with mild to moderate node mobility. One such example is related to fast and efficient command and control in military applications, where nodes can frequently move. In our model for *sensor networks*, though, the sensor nodes are assumed to be quasi-stationary. Nodes are location unaware and will be left unattended after deployment. Recharging is assumed not possible and therefore, energy-efficient sensor network protocols are required for energy conservation and prolonging network lifetime.

For an ad hoc or sensor network with nodes set V, the goal of clustering is to identify a set of clusterheads that cover the whole network. Each and every node v in set V must be mapped into exactly one cluster, and each ordinary

node in the cluster must be able to directly communicate to its clusterhead. The clustering protocol must be completely distributed meaning that each node independently makes its decisions based only on local information. Further, the clustering must terminate fast and execute efficiently in terms of processing complexity and message exchange. Finally, the clustering algorithm must be resistant to moderate mobility (in ad hoc networks) and at the same time renders energy-efficiency, especially for sensor networks.

3.2 DECA Clustering Algorithm

In DECA, each node periodically transmits a Hello message to identify itself, and based on such Hello messages, each node maintains a neighbor list. Define the score function at each node as $\text{score} = w_1E + w_2C + w_3I$, where E stands for node residual energy, C stands for node connectivity, I stands for node identifier, and weights follow $\sum_{\text{i=1}}^{3} w_{\text{i}} = 1$. We put higher weight on node residual energy in our simulations. The computed score is then used to compute the delay for this node to announce itself as the clusterhead. The higher the score, the sooner the node will transmit. The computed delay is normalized between 0 and a certain upper bound $D_{\max}$, which is a key parameter that needs to be carefully selected in practice, like the DIFS parameter in IEEE 802.11. In our simulation, we choose $D_{\max} = 10\text{ms}$ and the protocol works well. After the clustering starts, the procedure will terminate after time T_{stop}, which is another key parameter whose selection needs to take node computation capability and mobility into consideration. In the simulation, we choose $T_{\text{stop}} = 1\text{s}$.

The distributed clustering algorithm at each node is illustrated in the pseudo code fragments. Essentially, clustering is done periodically and at each clustering epoch, each node either immediately announces itself as a potential clusterhead or it holds for some delay time.

On receiving such clustering messages, a node needs to check whether the node ID and cluster ID embedded in the received message are the same; same node ID and cluster ID means that the message has been transmitted from a clusterhead. Further, if the receiving node does not belong to any cluster, and the received score is better than its own score, the node can simply join the advertised cluster and cancel its delayed announcement.

I. START-CLUSTERING-ALGORITHM()

```
1  myScore = w1E + w2C + w3I;
2  delay = (1000 − myScore)/100;
3  if (delay < 0)
4     then broadcastCluster (myId, myCid, myScore);
5     else
6          delayAnnouncement ();
7  Schedule clustering termination.
```

II. RECEIVING-CLUSTERING-MESSAGE(id, cid, score)

```
1  if (id == cid)
2     then if (myCid == UNKNOWN)
3               then if (score > myScore)
4                         myCid = cid;
5                         cancelDelayAnnouncement ();
6                         broadcastCluster (myId, myCid,score);
7  elseif (score > myScore)
8     then if (myId == myCid)
9               then needConversion = true;
10              else
11                    convertToNewCluster ();
```

III. FINALIZE-CLUSTERING-ALGORITHM()

```
1  if (needConversion)
2     then if (!amIHeadforAnyOtherNode ())
3               then convertToNewCluster ();
4  if (myCid == UNKNOWN)
5     then myCid = cid;
6          broadcastCluster (myId, myCid, score);
```

If the receiving node currently belongs to some other cluster, and the received score is better than its own score, two cases are considered. First, if the current node belongs to a cluster with itself as the head, receiving a better scored message means that this node may need to switch to the better cluster. However, cautions need to be taken here before switching since the current node, as a clusterhead, may already have other nodes affiliated with it. Therefore, inconsistencies can occur if it rushes to switch to another cluster. In our approach, we simply mark the necessity for switching (line 9 in Phase II) and defer it to finalizing phase, where it checks to make sure that no other nodes are affiliated with this node in the cluster as the head, before the switching can occur. But if the current node receiving a better-scored message is not itself a clusterhead, as an ordinary node, it can immediately convert to the new cluster, and this is the second case (line 11 in Phase II). It is critical to note that the switch process mandates that a node needs to leave a cluster first before joining a new cluster. In the finalizing phase, where each node is forced to enter after T_{stop}, each node checks to see if it needs to convert. Further, each node checks if it already belongs to a cluster and will initiate a new cluster with itself as the head if not so.

3.3 Correctness and Complexity

The protocol described above is completely distributed, and we have proven the following results[1].

- Eventually DECA terminates.
- At the end of Phase III, every node can determine its cluster and only one cluster.

[1]For a thorough description, please refer to our previous work [9].

- When clustering finishes, any two nodes in a cluster are at most two-hops away.
- Each node transmits only one message in operation.
- The time complexity of the algorithm is $O(|V|)$.

3.4 Performance Evaluation

We evaluate DECA using an in-house simulation tool called agent-based network simulator, *NetSim*. In our simulations, random graphs are generated so that nodes are randomly dispersed in a $1000\text{m} \times 1000\text{m}$ region and each node's transmission range is bound to 250m. We investigate the clustering performance under different node mobility patterns, and the node speed ranges from 0 to 50m/s. For each speed, each node takes the same maximum speed and a large number of random graphs get generated. Simulations results are averaged over these random graphs.

In general, it is undesirable to create single-node clusters. Single-node clusters arise when a node is forced to represent itself . While many other protocols generate lots of single-node clusters as node mobility gets more aggressive, our algorithm shows much better resilience. We have considered the following metrics for performance comparisons: 1) the average overhead (in number of protocol messages); 2) the ratio of the number of clusters to the number of nodes in the network; 3) the ratio of the single-node clusters to the number of nodes in the network; and 4) the average residual energy of the selected clusterheads.

We first look at static scenarios where nodes do not move and the quasi-stationary scenarios where the maximum node speed is bounded at 0.1m/s. We choose [10] proposed by Lin (LIN) as a representative for those general clustering protocols, and choose Krishna's algorithm (KRISHNA) [8] to represent dominating-set based clustering protocols. For energy-aware protocols, we choose HEED [17] to compare with DECA.

Fig. 8 shows that KRISHNA has the worst clustering performance with the highest cluster-to-nodes ratio, while DECA and LIN possess the best performance. HEED performs in between. In addition, all four protocols perform consistently under (very) mild node mobility.

As we increase the maximum node speed, both LIN and KRISHNA fail to generate clusters. This is expected. In LIN, a node will not transmit its message until all its better-scored neighbors have done so; the algorithm will not terminate if a node do not receive a message from each of its neighbors. Node mobility can make the holding node wait for ever. In KRISHNA, in order to compute clusters, each node needs accurate information of the entire network topology, facilitated by network-wide link state update which by itself is extremely vulnerable to node mobility. In contrast, we found that both HEED and DECA are quite resilient to node mobility in that they can generate

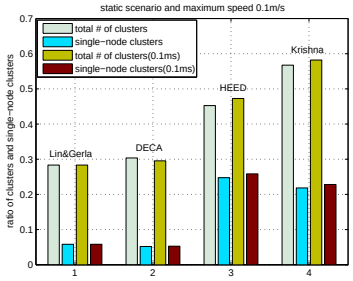

Figure 8. Ratio of number of clusters.

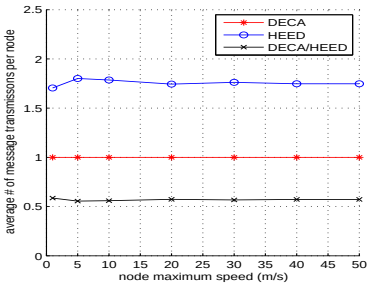

Figure 9. Average number of transmissions.

decent clusters even when each node can potentially move independently of others. The following figures compare the performance of DECA and HEED under node mobility.

Fig. 9 shows that for DECA, the number of protocol messages for clustering remains one per node, regardless of node speed. For HEED, the number of protocol messages is roughly 1.8 for every node speed, and a node running DECA transmits about 56% number of messages as that in HEED. The fact that HEED incurs more message transmissions is due to the possibly many rounds of iterations (especially when node power is getting reduced), where each node in every iteration can potentially send a message to claim itself as the candidate clusterhead. Reducing the number of transmissions is of great importance, especially in sensor networks, since it would render better energy efficiency and fewer packet collisions (e.g. CSMA/CA type MAC in IEEE 802.11).

Further, we compare DECA and HEED with respect to the (normalized) average clusterhead energy in Fig. 10. Again both DECA and HEED perform quite consistently and DECA outperforms HEED with about **twice** the average clusterhead residual energy. This is in accordance with Fig. 9 where DECA consistently incurs fewer message transmissions than HEED. In sensor networks, sending fewer messages by each node in DECA while achieving the intended goal usually means energy-efficiency and longer node lifetime, since transmissions typically consumes orders of magnitude more energy than processing.

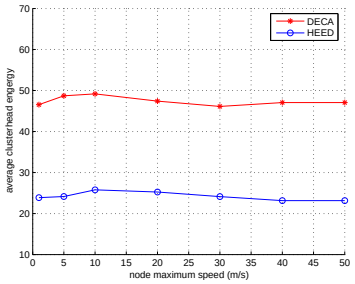

Figure 10. Average clusterhead energy.

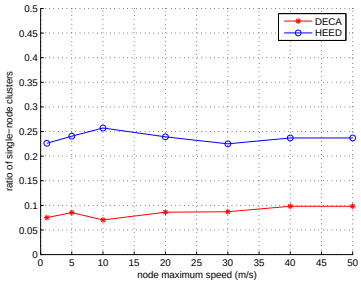

Figure 12. Ratio of single-node clusters.

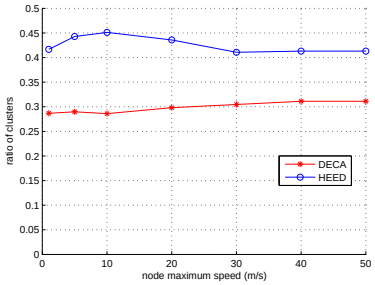

Figure 11. Ratio of clusters.

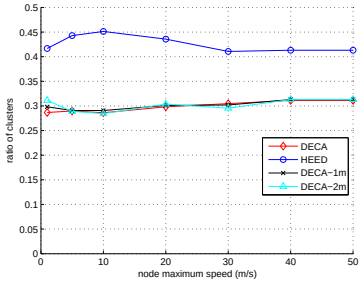

Figure 13. Resilience against sync drifts.

Fig. 11 and Fig. 12 illustrate the ratio of number of clusters and single node clusters to the total number of nodes in network. In both cases, DECA outperforms HEED. Note that both DECA and HEED perform quite *consistently* under different maximum node speed and this is not coincident: a node in both DECA and HEED will stop trying to claiming itself as the potential clusterhead after some initial period (delayed announcement in DECA and rounds of iterations in HEED) and enters the finalizing phase. As a result, the local information gathered, which serves as the base for clustering, is essentially what can be gathered within the somewhat invariant initial period which leads to consistent behaviors under different node mobility.

It can be observed that in DECA the dispersed delay timers for clusterhead announcement assume the existence of a global synchronization system. However, we can show that, our DECA scheme is in fact quite resilient against synchronization drifts. It has been shown in the literature [6] that synchronization errors can be controlled within the range of $10\mu s$ (with minimum efforts) for nodes in sensor networks, which have largely the most stringent computing and communicating resources. We further relax this time range and put upto $2ms$ of errors on the delay timers. Fig. 13 illustrates the simulation results. We can easily observe that with $1ms$ and $2ms$ synchronization error, the protocol performance tracks the case of perfect synchronization in an indistinguishable manner.

4 Relevant Work

A family of well-known protocols has been proposed for key management based on logical trees. One representative scheme is the Logical Key Hierarchies (LKH) scheme [15]. Essentially, the leaf nodes on the tree represent the different members, while the intermediate nodes are only logical and represent the different keys. Each member possesses all the keys on its path to the root, which serves as the group key. While LKH gains wide popularity, the fundamental principles of LKH predetermines that it will not fit well in large networking environment, if applied as a whole framework. First, there is no hierarchical structure in terms of group member organization: the hierarchy is only for key distribution purpose, and all the group members in LKH are of the same leaf level. Second, any member can send messages to the whole group or any selected members. This can incur serious security risks, especially in military environment.

In [16] the authors proposed an efficient localized algorithms that can quickly build a backbone directly in ad hoc networks. This approach uses a localized algorithm called the *marking process* where hosts interact with others in restricted vicinity. This algorithm is simple, which greatly eases its implementation, with low communication and computation cost; but it tends to create small clusters.

Similar to [10], Basagni [3] proposed to use nodes' weights instead of lowest ID or node degrees in clusterhead decisions. Weight is defined by mobility related parameters,

such as speed. Basagni [4] further generalized the scheme by allowing each clusterhead to have at most k neighboring clusterhead and described an algorithm for finding a maximal weighted independent set in wireless networks.

In Low-Energy Adaptive Clustering Hierarchy (LEACH) protocol [7], a node elects to become a clusterhead randomly according to a target number of clusterheads in the network and its own residual energy, and energy load gets evenly distributed among the sensors in the network. In addition, when possible, data are compressed at the clusterhead to reduce the number of transmissions.

5 Conclusions

While many key management schemes suffer from scalability problem, our multi-tiered key management scheme was designed to utilize the parallelism inherent in the multicast topology. Therefore, we expect our algorithms to perform extremely efficiently in practice. In addition, our scheme provides added security in that the group dynamics can also be protected. Furthermore, our scheme provides the capacity for selective group communication.

Our distributed clustering algorithm works with resilience to node mobility and at the same time renders energy efficiency. The algorithm terminates fast, has low time complexity and generates non-overlapping clusters with good clustering performance. Our approach is applicable to both mobile ad hoc networks and energy-constrained sensor networks. Combined together, our scheme is powerful and general, and it can naturally fit into the hybrid military/commercial communication infrastructure.

Acknowledgment

This work was supported by the Air Force Research Laboratory, USA, grant FA8750-05-C-0161.

References

[1] I. F. Akyildiz, W. Su, Y. Sanakarasubramaniam, and E. Cayirci, “Wireless sensor networks: A survey,” Computer Networks, vol. 38, no. 4, pp. 393-422, 2002.

[2] S. Banerjee, B.Bhattacharjee, “Scalable Secure Group Communication over IP Multicast”, JSAC Special Issue on Network Support for Group Communication, 2002.

[3] S. Basagni, “Distributed clustering for ad hoc networks,” in Proc. of the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks.

[4] S. Basagni, D. Turgut, and S. K. Das, “Mobility-adaptive protocols for managing large ad hoc networks,” in Proc of the IEEE International Conference on Communications, ICC 2001, 2001, pp. 1539-1543.

[5] B. N. Clark, C. J. Colburn, and D. S. Johnson, “Unit disk graphs,” Discrete Mathematics, vol. 86, pp. 165-167, 1990.

[6] L. Elson, L. Girod, D. Estrin, “Fine-grained network time synchronization using reference broadcasts”, ACM SIGOPS Operating System Review, vol. 36, pp. 147-163, 2002.

[7] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “Energy efficient communication protocol for wireless microsensor networks,” in Proceedings of the3rd Annual Hawaii International Conference on System Sciences, HICSS 2000, pp. 3005-3014

[8] P. Krishna, N.N. Vaidya, M. Chatterjee and D.K. Pradhan, “A cluster-based approach for routing indynamic networks”, ACM SIGCOMM Computer Communication Review 49 (1997) 49-64.

[9] J. H. Li, M. Yu, and R. Levy, “ Distributed Efficient Clustering Approach for Ad Hoc and Sensor Networks” in Proc International Conference on Mobile Ad-hoc and Sensor Networks, 2005.

[10] C. R. Lin and M. Gerla, “Adaptive clustering for mobile wireless networks,” Journal on Selected Areas in Communications, vol. 15, no. 7, pp. 1265-1275, September 1997.

[11] B. Schneier. Applied Cryptography. John Wiley and Sons, 1996.

[12] A.Sherman, and D. McGrew, “Key Establishment in Large Dynamic Groups Using One-Way Function Trees”, IEEE Transactions on Software Engineering, pp. 444-458, 29(6), May 2003.

[13] Y. Sun, and K. J. Liu, “Securing Dynamic Membership Information in Multicast Communication”, Proceedings of INFOCOM, 2004.

[14] M. Waldvogel, G. Caronni, D.Sun, N.Weiler, B. Plattner, “The VersaKey Framework: Versatile Group Key Management”, IEEE Journal on Selected Areas in Communications, 17(9), August 1999.

[15] D. Wallner, E. Harder, and R. Agee. “Key Management for Multicast: Issues and Architecture”, At ftp://ftp.ietf.org/rfc/rfc2627.txt, 1997

[16] J. Wu and H. Li, “On calculating connected dominating sets for efficient routing in ad hoc wireless networks,” Telecommunication Systems, vol. 18,no. 1/3, pp. 13-36, 2001.

[17] O. Younis, S. Fahmy, “HEED: A Hybrid, Energy-Efficient,Distributed Clustering Approach for Ad Hoc Sensor Networks”, IEEE Trans. ON Mobile Computing, vol. 3, no. 4, 2004