Future Generation Computer Systems 24 (2008) 860–869

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A scalable key management and clustering scheme for wireless ad hoc and sensor networks

Jason H. Li [a,*], Bobby Bhattacharjee [b], Miao Yu [c], Renato Levy [a]

[a] Intelligent Automation Inc., 15400 Calhoun Drive, Ste 400, Rockville, MD, 20855, USA

[b] Department of Computer Science, University of Maryland, College Park, MD, 20742, USA

[c] Department of Mechanical Engineering, University of Maryland, College Park, MD, 20742, USA

ARTICLE INFO

Article history:
Received 30 January 2007
Received in revised form
11 January 2008
Accepted 16 March 2008
Available online 12 June 2008

Keywords:
Key management
Clustering
Group communications
Ad hoc networks
Sensor networks

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, poses a significant challenge on the scalability and efficiency on key management research [13,26].

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 [10,14,17–20,24]. 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 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) [25], 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

* Corresponding author. Tel.: +1 301 294 5275; fax: +1 301 294 5201.
E-mail address: jli@i-a-i.com (J.H. Li).

0167-739X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.03.007

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 active 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, DECA protocol renders more robust and energy-efficient clustering. Such efficiency nicely supports those 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.

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

The basic ideas of multi-tiered key management are adapted from a previous work by one of the authors [3], where the cluster size at each tier is bounded between k and $2k - 1$ for some integer k, and better scalability over flat architectures has been achieved. In practice, such a uniform bound across different tiers may not be realistic. 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. Such work will provide practical guidelines for directing deployment, in addition to its generalized theoretical importance. We use simple examples to illustrate concepts and ideas.

2.1. Description of the technical approach

2.1.1. Member hierarchy

Our key distribution scheme creates a member hierarchy. A tier or layer comprises 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).

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 apply to all the subgroups in that layer; different layers can have different sets 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.

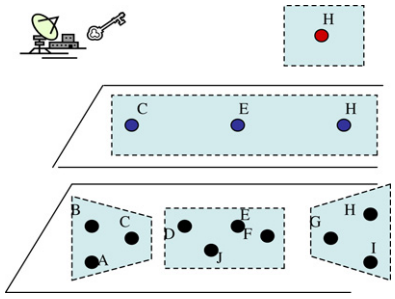

Fig. 1. Initial arrangement of members.

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 a 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. Authentication and access control

2.1.3. 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. a member joins or leaves any layer). A secret subgroup key is associated with each subgroup. Once again, a 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.

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 [15] 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.

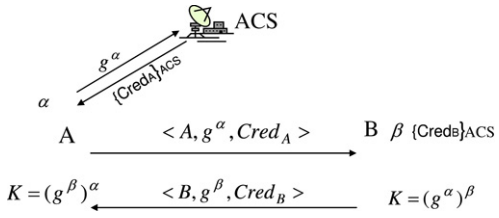

Fig. 2. A variant of Diffie–Hellman protocol.

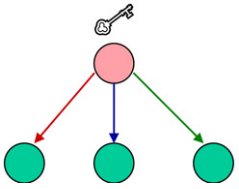

Fig. 3. Subgroup re-keys operations.

Similarly, each subgroup has a secret subgroup key, known to only all the 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.
- 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 $K_p(v)$ represent 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.

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

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

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

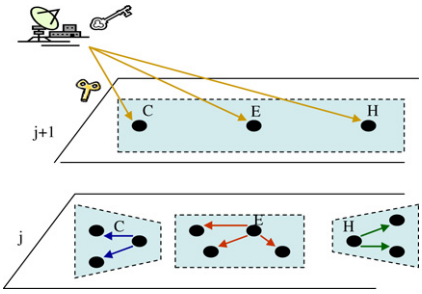

Fig. 4. Layer re-keys operations.

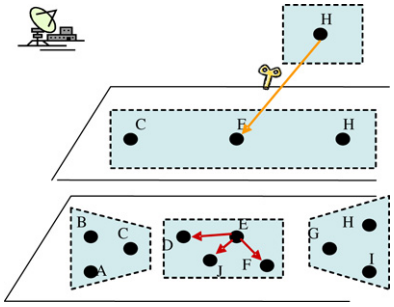

Fig. 5. Selective transmission.

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 layer 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, and in turn re-multicasts the data to their 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. In general, any higher layer leader can select to communicate with its subgroup members in the next lower layer, and so on. 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 encrypts 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 be the leader. Consequently, peer-to-peer data transmissions will be limited within the subgroup; no re-multicast will happen.

2.2. Attack and security analysis

General attack and 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 re-key operations.

Group dynamics security. We refer to *group dynamics information (GDI)* [21] 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 schemes [16,22,23], group members can distinguish the key updating process due to user join and that due to user departure, and the re-key message size is closely related with the group size. As a result, attackers can estimate the number of joins and leaves by examining the re-key processes, and estimate the number of members from the re-key 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 re-keys, the re-keying 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 [3] 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 joining/leaving 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 re-keyed. Let the minimum cluster size be k, and 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* $O(C) * O(\frac{k \log_k n}{n})$.
- *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 joining/leaving 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.

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.

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:

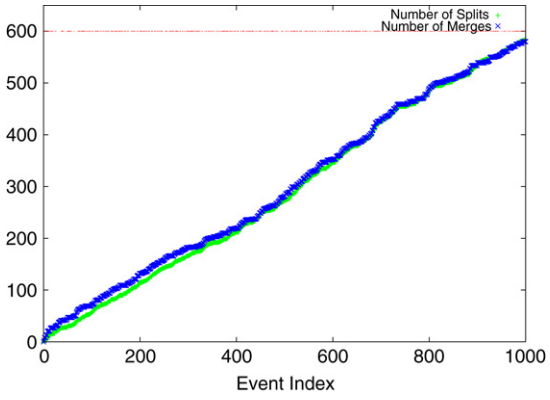

Fig. 6. Maximum cluster size 8.

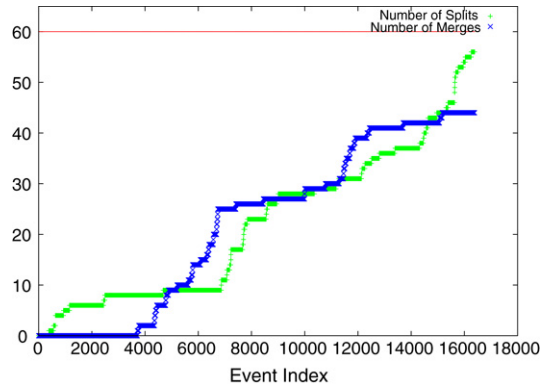

Fig. 7. Maximum cluster size 24.

- 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 the *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 [6]. 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, $T_O \gg T_C$. 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 quickly and execute efficiently in terms of processing complexity and message

Table 1
Batch updates results for member joins and leaves

Batch size	Min. cl. size	Max. cl. size	Num. merges	Merge total	Num. splits	Split total
1	4	8	2396	19 130	2 480	20 925
10	4	8	2218	17 684	2 292	19 466
100	4	8	1310	10 044	1 356	11 738
1000	4	8	282	2 050	386	3 513
1	4	12	188	1 852	246	3 012
10	4	12	219	2 189	276	3 387
100	4	12	165	1 693	233	2 913
1000	4	12	66	658	118	1 538
1	4	16	55	636	94	1 543
10	4	16	59	717	93	1530
100	4	16	38	457	89	1 481
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	12 857	780	13 533
10	8	16	793	13 527	820	14 234
100	8	16	446	7 470	475	8 283
1000	8	16	94	1 456	120	2 108
1	8	24	38	819	60	1 482
10	8	24	50	1 072	70	1 722
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	5 505	86	6 028
10	32	64	29	2 111	37	2 633
100	32	64	23	1 663	30	2 124
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

exchange. Finally, the clustering algorithm must be resistant to moderate mobility (in ad hoc networks) and at the same time render 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. We define the score function at each node as 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 $\sum_{i=1}^{3} w_i = 1$. 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 $D_{\max} = 10$ 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 $T_{\text{stop}} = 1$ 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.

Upon receiving clustering messages, a node needs to check whether the node ID and the cluster ID embedded in the received message are the same; same node 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, the node can mark down the advertised cluster and wait until its scheduled announcement to send its message.

I. START-CLUSTERING-ALGORITHM()

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

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

```
1  if (id == cid)
2     then if (myCid == NULL)
3             then if (score > myScore)
4                     myCid = cid;
5                     elseif (score > myScore)
6                       then if (myId == myCid)
7                               needConvert = true;
8                       else  markBestCluster();
```

III. ACTUAL-ANNOUNCEMENT()

```
1  bcastClstr (myId, myCid, score);
```

IV. FINALIZE-CLUSTERING-ALGORITHM()

```
1  if (needConvert)
2     then if (!amIHeadforAnyOtherNode ())
3             then convtToNewClst ();
4  if (myCid == NULL)
5     then myCid = cid;
6          bcastClstr (myId, myCid, score);
```

If the receiving node currently belongs to some cluster, and the received score is better than its own score, two cases are further considered. First, if the current node receiving a better-scored message is not a clusterhead itself, as an ordinary node, it can immediately mark down the best cluster so far (line 8 in II) and wait until its scheduled announcement. This node will stay in its committed cluster after its announcement. On the other hand, if the current node is a clusterhead itself, receiving a better scored message (due to variant delays and/or synchronization drifts) 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 7 in II) and defer it to the finalizing phase, where it checks to make sure that no other nodes are affiliated with this node in the cluster as the head, before switching can occur. It is noted that the switch process mandates that a node needs to leave a cluster first before joining a new cluster. Further, it is important to point out that since each node announces itself according to the computed score, this second case is really the exception, rather than the normal case. For example, lower scored nodes may transmit earlier when synchronization drifts among nodes are large. We include such exception handling in DECA to achieve better robustness. In this (rare) case, the conversion procedure incurs one more message transmission for the converted node. In normal operations, however, each node transmits only one message.

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.

3.3. Correctness and complexity

The protocol described above is completely distributed. To show the correctness and efficiency of the algorithm, we have the following results.[1]

- Eventually DECA terminates.
- At the end of Phase IV, every node can determine its cluster and only one cluster.
- When clustering finishes, any two nodes in a cluster are at most two hops away.
- In DECA, each node transmits only one message during the operation.
- The time complexity of the algorithm is $O(|V|)$.

3.4. Performance evaluation

In this section, we evaluate the DECA protocol via simulations. We use an in-house simulation tool called the agent-based ad hoc network simulator (NetSim) to implement our protocol and the protocols proposed by Krishna et al. [10], Lin and Gerla [14], and HEED [25] for comparisons. Compared with other network simulators (for instance ns-2), the most important feature of NetSim is its capability of handling massive ad hoc wireless networks and sensor networks.

In general, it is undesirable to create single-node clusters. Single-node clusters arise when a node is forced to represent itself (because of not receiving any clusterhead messages). A cluster may also contain a single node if this node decides to act as a clusterhead

[1] For detailed analysis of the clustering protocol, the readers are referred to our previous work [11,12].

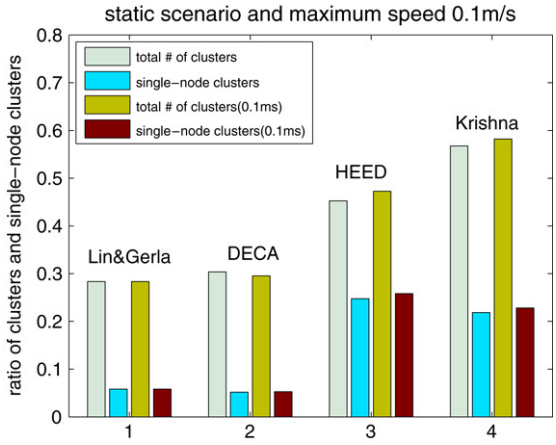

Fig. 8. Ratio of number of clusters.

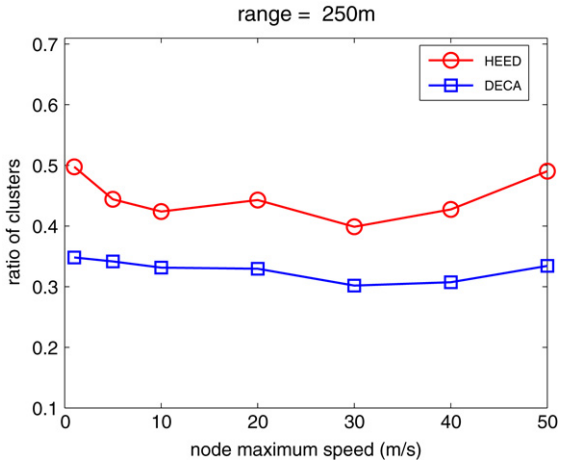

Fig. 9. Ratio of clusters.

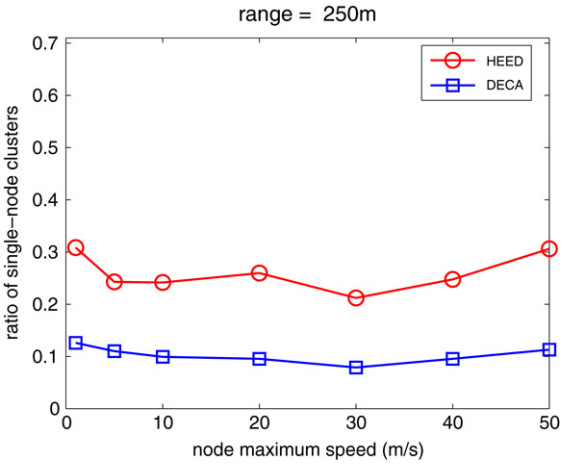

Fig. 10. Ratio of single-node clusters.

and all its neighbors register themselves with other clusterheads. While other protocols will generate lots of single-node clusters as node mobility gets more aggressive, our algorithm shows much better resilience under such situations.

In our simulations, random graphs are generated so that nodes are randomly dispersed in a 1000 m $\times$ 1000 m region and the transmission range of each node is bound to 250 m. We investigate the clustering performance under different node mobility patterns, and the node speed ranges from 0 to 50 m/s. In particular, we simulate the following scenarios with maximum node speed set as 0, 0.1, 1, 5, 10, 20, 30, 40, and 50 m/s. For each scenario, each node takes the same maximum speed and a large number of random graphs are generated. Simulations are run and results are averaged over these random graphs.

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 0.1 m/s. We choose [14] proposed by Lin (LIN) as a representative for those general clustering protocols, and choose Krishna's algorithm (KRISHNA) [10] to represent dominating-set based clustering protocols. For the state-of-the-art, we choose HEED [25] 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.

Both LIN and KRISHNA fail to generate clusters as we increase the maximum node speed. 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 does 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 a 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 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.

Figs. 9 and 10 shows the ratio of the number of clusters and single-node clusters to the total number of nodes in the network. All nodes have the same transmission range of 250 m. In both cases, DECA significantly outperforms HEED, with performance gains around 40% in Fig. 9 and 200% in Fig. 10.

Fig. 11 shows that for DECA, the number of protocol messages for clustering remains one per node, regardless of the node speed. For HEED, the number of protocol messages is roughly 1.7–2 for every node speed. The fact that HEED incurs more message transmissions is due to the possibly many rounds of iterations, where each node in every iteration can potentially send a message to claim itself as the candidate clusterhead [25].

Fig. 12 compares DECA and HEED with respect to the (normalized) average clusterhead energy. Again, DECA outperforms HEED with about twice the clusterhead residual energy. This is in accordance with Fig. 11 where DECA consistently incurs fewer message transmissions than HEED. 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. in IEEE 802.11 MAC.

We extend our simulations to investigate how DECA and HEED perform under different node speeds and transmission ranges. Fig. 13 shows that DECA performs quite consistently in terms of cluster ratio under various node speeds, and the larger the transmission range, the lower the cluster ratio (as expected). Such observations can also be made in Fig. 14, where the cluster ratio curves under different node speeds track each other quite closely, and the ratio of clusters decreases as the transmission range increases. Similar observations have also been made for HEED.

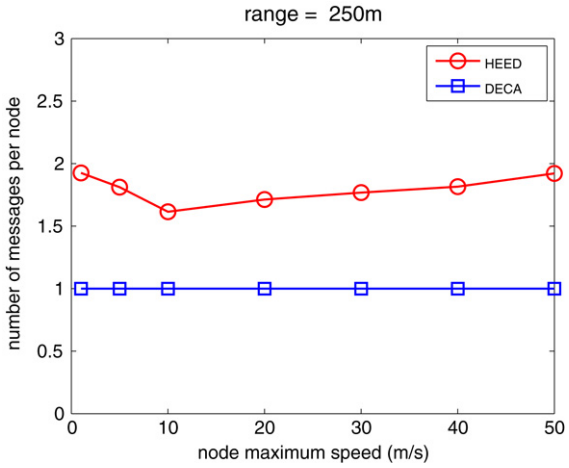

Fig. 11. Number of messages per node.

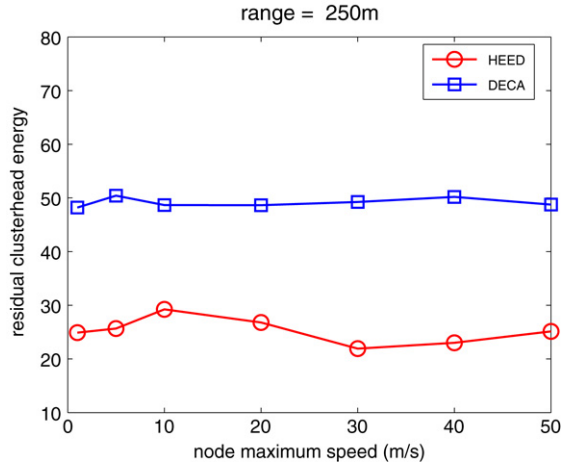

Fig. 12. Residual clusterhead energy.

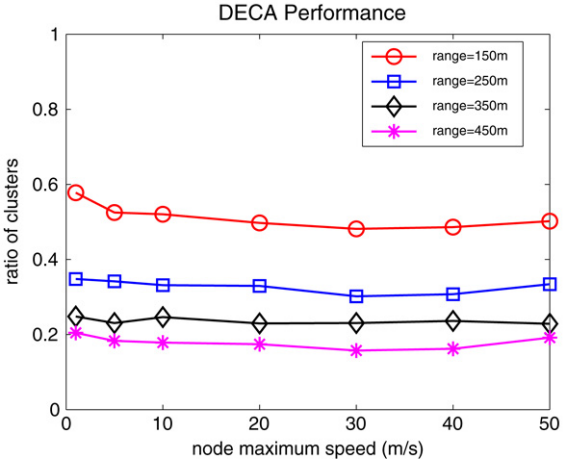

Fig. 13. Cluster ratio versus speed.

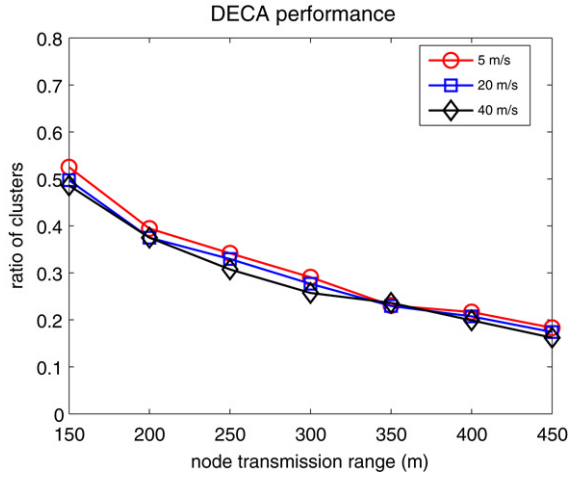

Fig. 14. Cluster ratio versus range.

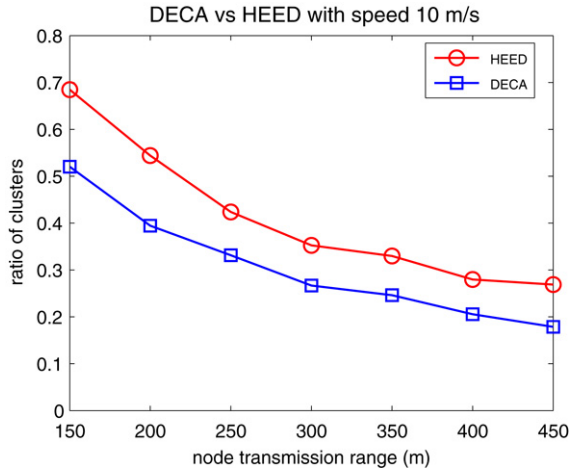

Fig. 15. Cluster ratio of HEED and DECA.

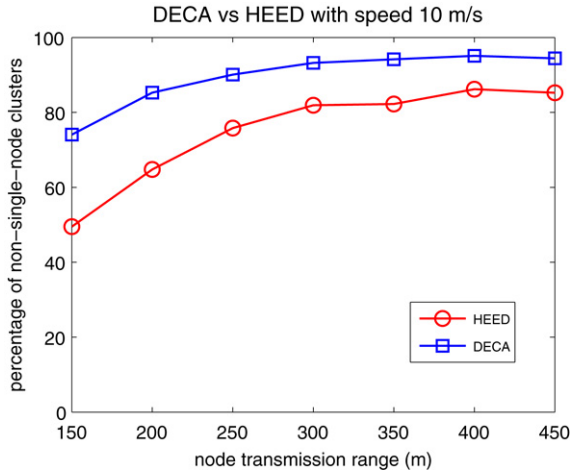

Fig. 16. Non-single-node clusters percentage.

Now we will show that DECA outperforms HEED significantly for every transmission range used in simulations. Given the mobility resilience, we pick 10 m/s node speed as a representative scenario. Figs. 15 and 16 illustrate the simulation results comparing DECA and HEED. It is obvious to see that DECA consistently incurs a smaller ratio of clusters and higher percentage of non-single-node clusters for every transmission range. In addition, it is interesting to observe that the performance gain of DECA over HEED decreases as the transmission range increases. This is particularly evident in Fig. 16.

From Fig. 15 one may conclude that a larger transmission range is more preferable for better clustering performance. However, it is generally undesirable to extend a node's transmission range in multi-hop wireless networks due to energy and interference issues. To tackle this trade-off, we propose the energy-conservative

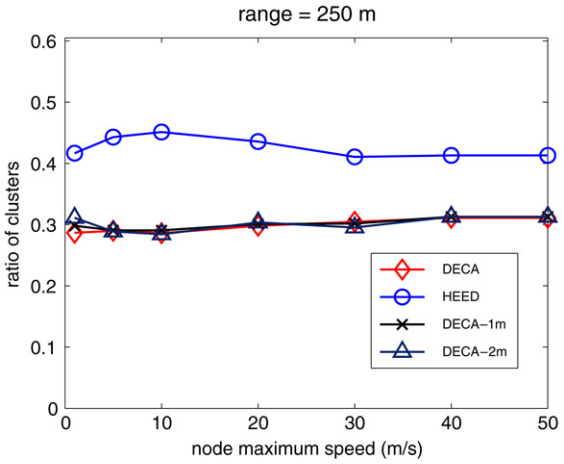

Fig. 17. Protocol resilience against synchronization drifts–cluster ratio.

approach: *select the smallest transmission range that brings about the largest performance improvement*.

For example, the ratio of clusters in Fig. 13 drops from about 0.55 to about 0.35 with range increases from 150 to 250 m. The ratio drops only to about 0.25 if the range increases further to 350 m. As a result, it might not be worthwhile to increase the range over around 250 m. This insight can be further validated by Fig. 14, where we should choose the range with the steepest slope in the figure, indicating the greatest improvement on the clustering performance. Again, we need to choose the range around 250 m. Fig. 16 also indicates that beyond 250 m transmission range, the performance of DECA become "flat" and its gain over HEED gets smaller. This energy-conservative approach is not only of simulation interests; practical deployment of the DECA algorithm should also follow such insights.

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 [7] that synchronization errors can be controlled within 10 μs for sensor nodes. We further relax this time range, and Fig. 17 illustrates the simulation results. We can easily observe that with 1 and 2 ms synchronization error, the protocol performance tracks the case of perfect synchronization in an indistinguishable manner.

4. Related work

4.1. On key management

In Group Key Management Protocol (GKMP) [8], a Group Key Controller is responsible for performing all the security related tasks including member join, deletion, and key generation and distribution. Static keys – pair-wise shared between the Group Key Controller and each group member – are used to establish a group key. In Scalable Multicast Key Distribution (SMKD) [2], a single key encrypting key and a single session encrypting key are distributed: compromise of a single number will disclose these two keys, and compromise the future group communication. So the re-keying problem remains unsolved. Neither GKMP nor SMKD is viable for re-keying large groups with dynamic membership.

A family of protocols has been proposed for key management based on logical trees [16,22,23]. One well known scheme that provably incurred sub-linear group re-keying overheads for each membership change is the Logical Key Hierarchies (LKH) scheme [23]. Essentially, the LKH scheme defines a logical hierarchy of keys distributed between different sets of members. 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. When a member leaves the group, all keys on the path from this member to the root need to be changed.

While LKH is gaining wide popularity and there are various research works based on the virtual key hierarchy (e.g. reducing re-key messages, efficient one-way function tree, etc.), the fundamental principles of LKH lead to several drawbacks. First, there is no hierarchical structure in terms of group member organization. The hierarchy is only for key distribution purposes. 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. In addition, it is noted that such LKH schemes are vulnerable to attacks on the group dynamics information, which can be devastating if leaked to military adversaries. Our scheme is resistant to such attacks.

4.2. On clustering protocols

Among clustering mechanisms in ad hoc and sensor networks, dominating-set-based clustering [17–20] surfaces as one of the most promising approaches. A subset of vertices in an undirected graph is a dominating set if every vertex not in the subset is adjacent to at least one vertex in the subset. Moreover, this dominating set should be connected for ease of the routing process.

Sivakumar et al. [17–19] proposed a series of 2-level hierarchical routing algorithms for ad hoc wireless networks. The idea is to identify a subnetwork that forms a minimum connected dominating set (MCDS). In this approach, a connected dominating set is found by growing a tree T starting from a vertex with the maximum node degree. Then, a vertex v in T that has the maximum number of neighbors not in T is selected. Finally, a spanning tree is constructed and non-leaf nodes form a connected dominating set.

Ref. [24] proposed 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.

Instead of constructing connected dominating sets, Lin and Gerla [14] used node ID numbers to build clusters of nodes that are reachable by two-hop paths. The distributed clustering algorithm is initiated by all nodes that have the lowest ID numbers among their neighbors. If all the lower ID neighbors sent their decisions and none declared itself as a cluster initiator, the node decides to create its own cluster and broadcasts its own ID as the cluster ID. Otherwise, it chooses a neighboring cluster with the lowest ID, and broadcasts such decision.

Similar to [14], Basagni [4] 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 [5] further generalized the scheme by allowing each clusterhead to have at most k neighboring clusterheads and described an algorithm for finding a maximal weighted independent set in wireless networks.

One of the first protocols that use clustering for network longevity is the Low-Energy Adaptive Clustering Hierarchy (LEACH) protocol [9]. In LEACH, a node elects to become a clusterhead randomly according to a target number of clusterheads in the network and its own residual energy, and the energy load get evenly distributed among the sensors in the network. A limitation of this scheme is that it requires all current clusterheads to be able to transmit directly to the sink. Improvements to the basic LEACH algorithms include multi-layer LEACH-based clustering and the optimal determination of the number of clusterheads that minimizes the energy consumption throughout the network.

None of the above algorithms intends to tackle the scenarios where all nodes in the network can potentially move. Our protocol handles such a challenge. The initial ideas were proposed in [11], and a follow-up study focused on clustering performance under lossy channels and synchronization errors [12]. This work complements the previous efforts with enriched results and insights under different node transmission ranges.

5. Conclusions

While many key management schemes suffer from scalability problems, our multi-tiered key management scheme was designed to utilize the parallelism inherent in the multicast topology. Though we have run simulations with 256 and 512 nodes; simulations with more nodes are expected to produce similar results. 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 quickly, 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 partially supported by the Air Force Research Laboratory, USA, grant FA8750-05-C-0161.

References

[1] I.F. Akyildiz, W. Su, Y. Sanakarasubramaniam, E. Cayirci, Wireless sensor networks: A survey, Computer Networks 38 (4) (2002) 393–422.
[2] A. Balladie, Scalable multicast key distribution, RFC 1949, May 1996.
[3] S. Banerjee, B. Bhattacharjee, Scalable secure group communication over IP multicast, in: Network Support for Group Communication, JSAC (2002) (special issue).
[4] S. Basagni, Distributed clustering for ad hoc networks, in: Proc. of the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks.
[5] S. Basagni, D. Turgut, S.K. Das, Mobility-adaptive protocols for managing large ad hoc networks, in: Proc of the ICC 2001, pp. 1539-1543.
[6] B.N. Clark, C.J. Colburn, D.S. Johnson, Unit disk graphs, Discrete Mathematics 86 (1990) 165–167.
[7] L. Elson, L. Girod, D. Estrin, Fine-grained network time synchronization using reference broadcasts, ACM SIGOPS Operating System Review 36 (2002) 147–163.
[8] H. Harney, C. Muckenhirn, Group key management protocol (GKMP) architecture, in: IETF RFC 2094, July 1997.
[9] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy efficient communication protocol for wireless microsensor networks, in: Proc of the 3rd Hawaii International Conference on System Sciences, 2000, pp. 3005–3014.
[10] P. Krishna, N.N. Vaidya, M. Chatterjee, D.K. Pradhan, A cluster-based approach for routing in dynamic networks, ACM SIGCOMM Computer Communication Review 49 (1997) 49–64.
[11] J.H. Li, M. Yu, R. Levy, Distributed efficient clustering approach for ad hoc and sensor networks, in: Proc International Conference on Mobile Ad-hoc and Sensor Networks, 2005.
[12] J.H. Li, M. Yu, R. Levy, A. Teittinen, A mobility-resistant efficient clustering approach for ad hoc and sensor networks, Mobile Computer Communications Review 10 (2) (2006) 1–12.
[13] L. Liao, M. Manulis, Tree-based group key agreement framework for mobile ad-hoc networks, Future Generation Computer Systems 23 (6) (2007) 787–803.
[14] C.R. Lin, M. Gerla, Adaptive clustering for mobile wireless networks, Journal on Selected Areas in Communications 15 (7) (1997) 1265–1275.
[15] B. Schneier, Applied Cryptography, John Wiley and Sons, 1996.
[16] A. Sherman, D. McGrew, Key establishment in large dynamic groups using one-way function trees, IEEE Transactions on Software Engineering 29 (6) (2003) 444–458.
[17] R. Sivakumar, B. Das, V. Bharghavan, The clade vertebrata: Spines and routing in ad hoc networks, in: Proc. of the IEEE Symposium on Computer Communications, ISCC'98.
[18] R. Sivakumar, B. Das, Spine-based routing in ad hoc networks, ACM/Baltzer Cluster Computing Journal 1 (1998) 237–248.
[19] R. Sivakumar, P. Sinha, V. Bharghavan, CEDAR: A core-extraction distributed ad hoc routing algorithm, IEEE Journal on Selected Areas in Communications 17 (8) (1999) 1454–1465.
[20] I. Stojmenovic, M. Seddigh, J. Zunic, Dominating sets and neighbors elimination-based broadcasting algorithms in wireless networks, IEEE Transactions on Parallel and Distributed Systems 13 (1) (2002).
[21] Y. Sun, K.J. Liu, Securing dynamic membership information in multicast communication, in: Proc. of INFOCOM, 2004.
[22] 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) (1999).
[23] D. Wallner, E. Harder, R. Agee, Key management for multicast: issues and architecture. At: ftp://ftp.ietf.org/rfc/rfc2627.txt, 1997.
[24] J. Wu, H. Li, On calculating connected dominating sets for efficient routing in ad hoc wireless networks, Telecommunication Systems 18 (1/3) (2001) 13–36.
[25] O. Younis, S. Fahmy, HEED: A hybrid, energy-efficient,distributed clustering approach for ad hoc sensor networks, IEEE Transactions on Mobile Computing 3 (4) (2004).
[26] X. Zou, Y. Dai, X. Ran, Dual-level key management for secure grid communication in dynamic and hierarchical groups, Future Generation Computer Systems 23 (6) (2007) 776–786.

Jason H. Li received his B.E. and M.S. degrees in Electrical Engineering from the Tsinghua University, Beijing, China in 1993 and 1996 respectively, and his Ph.D. degree in Electrical and Computer Engineering, from the University of Maryland at College Park, USA, in 2002, majoring in computer communication networks.

He is currently a senior research scientist at Intelligent Automation, Inc., Rockville, MD, USA. Prior to that, he was a senior research scientist in Hughes Network Systems. His research interests lie in the general area of computer communication networks, including network protocols, network security, network management and control, and distributed software agents.

Dr. Li is a member of the IEEE.

Bobby Bhattacharjee received his B.E. degree in Computer Science and Mathematics from Georgia College in 1994 and his Ph.D. degree in Computer Science from the College of Computing at Georgia Tech. in 1999.

He is currently an Associate Professor in the Computer Science department at the University of Maryland, College Park. His research interests are in the design and implementations of scalable systems, protocol security, and peer-to-peer systems.

Dr. Bhattacharjee is a fellow of the Sloan Foundation, and a member of the ACM.

Miao Yu obtained her B.S and M.S. degrees in the field of engineering mechanics from the Tsinghua University, Beijing, China, in 1996 and 1998 respectively, and her Ph.D. degree in mechanical engineering from the University of Maryland at College Park, USA, in 2002.

She has been an Assistant Professor in the Department of Mechanical Engineering in the University of Maryland at College Park since January 2005. Her research interests include smart sensors, sensor systems for military, civil, mechanical, electrical, biochemical, and environmental applications, collaborative sensor signal processing, and sensor networks.

Dr. Yu is a member of the OSA, SPIE, ASME, ASEE, and SEM. Her awards include the Invention of the Year Award from the University of Maryland in 2002.

Renato Levy received his BSEE degree in electrical engineering from the Federal University of Rio de Janeiro, Brazil, in 1986, his MBA degree from Institute for Business and market economy, Brazil, in 1992, and his D.Sc. degree in computer science from the George Washington University, USA, in 2004.

He is currently a principal scientist at Intelligent Automation, Inc., Rockville, MD, USA. His research interests include distributed systems, embedded systems, modeling and simulation, software engineering, and wireless networks.

Dr. Levy is a member of the IEEE and ACM.