New Algorithm for the design of topology aware Hypercube in multi-hop ad hoc Networks

Maria Striki, Kyriakos Manousakis,
Telcordia Technologies Inc.,
1 Telcordia Dr, Piscataway,
NJ, 08854, USA

John S. Baras
Institute for Systems Research
University of Maryland, College Park
MD, 20742, USA

Abstract-**Securing group communications in resource constrained, infrastructure-less environments such as Mobile Ad Hoc Networks (MANETs) has become one of the most challenging research directions in the areas of wireless network security. MANETs are emerging as the desired environment for an increasing number of commercial and military applications, addressing also an increasing number of users. Security on the other hand, is an indispensable requirement of modern life for such applications. The inherent limitations of MANETs impose major difficulties in establishing a suitable secure group communications framework. This is even more so for the operation of Key Agreement (KA), under which all parties contribute equally to the group key. The logical design of efficient KA protocols has been the main focus of the related research. Such a consideration however, gives only a partial account on the actual performance of a KA protocol in a multi-hop network as protocols have been evaluated only in terms of the key related messaging in isolation from network functions that interact with the logical scheme (i.e. routing). In recent work, we contributed towards the latter by efficiently extending a number of Diffie-Hellman group KA protocols in wireless multi hop ad hoc networks. In this work, we are extending a scheme that was left out in our previous work: Hypercube. Through analysis and simulations we demonstrate the superiority of the new, enriched H-Cube that takes into account the underlying routing by the use of a topologically aware communications schedule.**

I. INTRODUCTION

A MANET is a collection of wireless mobile nodes, communicating among themselves over possibly multi-hop paths, without the help of any infrastructure such as base stations or access points. As the development of multicast services such as secure conferencing, visual broadcasts, military command and control grows, research on security for wireless multicasting becomes increasingly important. The role of key management (KM) is to ensure that only valid members have access to a valid group key at any time. The operation of Key Agreement (KA) is a subset of the broad KM functionality and imposes that all participants contribute almost equally to the group key establishment, by interacting among themselves in ways designated by the specific protocol. Compared to other tasks classified under KM, KA operates on an inherently more complicated communication regulation mechanism. MANETs constitute the major challenge for the design of suitable KM schemes, with even more severe impact on KA. We are dealing with dynamic, infrastructure-less networks of limited bandwidth, unreliable channels where topology is changing fast. Connections are temporary and unreliable. These constraints turn most of the existing protocols inefficient in MANETs. Along with the continuous quest for the design of more efficient schemes, the need for the new KA schemes to handle successfully network dynamics with low impact is now equally important. Upon failures or disruptions, it is often the case that the group key establishment must start over. Whenever this event occurs, a significant amount of relays get involved in the exchange of large messages, and considerable delay burdens the network. The overall protocol performance degrades even more because of the indirect impact of routing on additional network layers (i.e. QoS deteriorates due to more collisions at the MAC layer, bandwidth usage, resources consumption increase undesirably). For all these reasons, reducing the combined costs resulting from routing and communications becomes essential if we want to apply more sophisticated KA schemes on MANETs. The logical design and analysis of efficient KA protocols has been the main focus of related research to-date. Such a consideration however, gives only a partial account on the feasibility and actual performance of a KA protocol in a wire-less multi-hop network. This is so because the evaluation of protocols is conducted via a logical network abstraction in such a way that essential inseparable operations, such as the underlying routing, are left out.

In [22], we contributed towards efficiently extending a number of KA protocols on wireless multi-hop ad hoc networks (GDH.1-2, ING), and measuring their actual performance over these networks. Initially, we assumed a physical group member graph (each edge is a physical link). We extended the studied protocols by allowing the formation of their communication schedules with respect to routing. The original versions do not exploit members' topological proximity. A pre-agreed schedule is used, based on members' attributes, like their *ID*s. After extending each KA protocol blindly without topological considerations, we observed that the routing structure of each protocol posed a different optimization problem (usually *NP*-complete) for every metric. Given that, we focused on providing efficient approximations that greatly improved the performance of the schemes, under the assumption of a *physical graph*. The work of [22] is extended in [25], where we address far more generic scenarios: now the group member graph represents a *logical topology* (i.e. each edge is a logical link, bounded from the number of hops between two vertices provided by the routing). Our new heuristics achieve now better approximations of the metric functions to optimize. In this work we extend [25] to include Hypercube, as it requires different manipulation from the ones extended so far.

Section 2 gives an overview of related work on KA and section 3 describes the original H-Cube. Section 4 gives an outline of our previous and current work, network model and assumptions. In section 5 we provide a detailed description of our new algorithms and in 6 we present the analysis of our auxiliary framework. In section 7 we present our simulations set-up and results. In section 8 we conclude the paper.

978-0-7695-3552-4/09 $25.00 © 2009 IEEE
DOI 10.1109/ICN.2009.52

II. Related Work

Proposals related to secure group KA protocols abound in the literature, and can be usually found under the broad category of contributory schemes. Most of them correspond to a logical consideration in terms of design, or address wire-line networks and cannot operate as such in MANETs.

Becker et al. [1], derived lower bounds for contributory key generation systems for the gossip problem and proved them realistic for Diffie-Hellman (DH) based protocols. They used the basic DH distribution [3] extended to groups from the work of Steiner et al. [2], where three new protocols are presented: GDH.1-2-3. Ingemarsson et al. [4] presented another efficient DH-based KA scheme, ING, logically implemented on a ring topology. Burmester et al. [5] introduced a new DH protocol, denoted as BD, while Kim et al. [7] introduced a hybrid DH scheme: TGDH. It is an efficient protocol that blends binary key trees with DH key exchanges. Becker in [1], introduced Hypercube, that requires the minimum number of rounds. He also introduced Octopus that requires minimum number messages and derived 2^d-Octopus that combined Octopus with Hypercube to a very efficient scheme that works for arbitrary number of nodes. Related work can be found in [6, 8, 9].

There exist some more recent proposals of KA for wireless ad-hoc networks. Even these, do not seem to scale well or handle successfully the network dynamics [14-19]. Amir et al. [12, 13], focus on robust KA, and attempt to make GDH protocols fault-tolerant to asynchronous network events. Their scheme is designed mainly for the Internet, and requires an underlying reliable group communication service and message ordering to guarantee the preservation of virtual semantics. In [21], Octopus protocols for robust group communications in MANETs are proposed. The focus is on their logical evaluation, in isolation from interacting network functions. In [22] and [25], we study the extension of certain KA protocols (GDH.1-2, ING) over MANETs. We estimate their combined routing-communication costs and then improve the metrics of interest applying a new communications schedule on each protocol, subject to the underlying routing. We distinguished two cases: the group members graph represents a physical topology in [22], and a logical topology in [25].

III. Original Hypercube Scheme (Overview)

Notation_1: $B(x) = a^x$ is the blinding (modular exponentiation *ME* under base *a*) of value *x* and $\varphi(x) = x \bmod n$.

Although the protocol is very well documented in [1, 8], we provide a brief description of the schemes for completeness.

Hypercube (H-cube): 2^d parties agree upon a key within *d* simple rounds performing pair-wise *Diffie-Hellman Key Exchanges* (DHKEs) on the edges of a logical *d*-dimensional cube. Every party is involved in exactly one DHKE per round. Each party uses the intermediate secret key K_{i-1} generated at the end of its DHKE in round (i-1), to compute the intermediate secret key K_i for round *i*: each party processes K_{i-1} and sends its peer in the clear the value $B(\varphi(K_{i-1}))$. The 2^d parties are identified on the *d*-dimensional space $GF(2)^d$ and a basis $b_1, \ldots, b_d \in GF(2)^d$ is selected to be used for the direction of communications per round. In every round, the parties communicate on a maximum number of parallel edges (round *i*, direction b_i). Members that acquire a common key from the previous round use the same "logical" direction of communications in the following round.

1st round: Every member M_i generates a random number N_i and performs a DHKE with member M_k, where $i, k \in GF(2)^d$, for example $k = i \oplus 2^{j-1}$, using values N_i and N_k respectively.

jth round: Every member M_i does a DHKE with member M_k, where $k = i \oplus 2^{j-1}$, where both members use the value generated in round *i*-1 as the secret value for the key establishment.

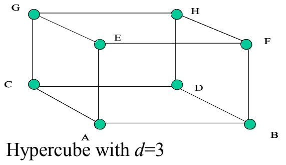


Fig 1.a: Hypercube with *d*=3

IV. Network Model, Specifications and Requirements

In this section we introduce our extensions to H-cube over multi-hop ad hoc networks when the group member graph represents a logical topology. Each edge is a logical link and its weight is bounded by the number of hops between two edges, provided by the routing. Intermediate relays are not necessarily group members. We impose that two nodes that are within each other's radio range have distance of 1-hop. Hence, any direct link has weight 1. The prefixes "*nt*" and "*wt*" abbreviate the extension of KA schemes on a wireless multi-hop network with "no topology" and "with topology" considerations respectively.

Notation_2: Let *n* be the number of members in the secure group, *m* the number of network nodes (size *S*), *D* the diameter *diam*(*G*) of the network graph *G*; that is, the max number of hops between a pair of nodes in *V*. Let $R(N_i, N_{i+1}) = R_{i,i+1}$ be the number of hops in the path between members N_i and N_{i+1}. Let *K* be the bit size of an element in the algebraic group used (where the decision DH problem is assumed hard).

A. Existing Work, Approach, Requirements and Objectives

The performance of H-cube on a logical plane is known and can be found in [1, 8, 19]. In [22] we re-evaluated it, executing it blindly on a multi-hop network, where multi-path routing is required for group members to communicate, and where not all members can be directly reached via single broadcast. We ran it on this framework using a communication schedule based merely on arbitrary member *ID*s. This *nt* approach may lead in overwhelming routing, high communication cost, as seen from Table I of our relevant results. Under the worst case, the resulting group member graph becomes bipartite with links of *D* hops, while under the best case, the physical graph is optimized w.r.t. the underlying routing.

TABLE 1. H-CUBE PERFORMANCE

	Logical *Lt*	Logical *CCost*	*nt-Lt*	*nt-CCost*
HCube	$\log_2 n$	$n\log_2 n$	$D\log_2 n$	$nD\log_2 n$

Table 1: Performance of HCube: (*a*) over logical networks, (*b*) *nt*-extension over multi-hop ad hoc networks under the worst case scenario,

Next, we integrate the underlying routing into the design by the definition of a new *wt* communication schedule that improves the metrics of bandwidth and latency for the logical member graph representation. We assume that each message from a group member is reliably and timely received by all neighbors. Our scheme inherits the security properties of its ancestor. Our main objective is to meet efficiency requirements for the group key establishment during the initial state of key generation. We assume that routing establishes end to end paths, avoiding intermediate link failures and did not consider dynamic cases (i.e. link failure, mobility) under which the network could be partitioned. Under the *nt* schedule, members' placement and the routes formed are random. This arbitrary factor that emerges when we merge the key generation blindly with the underlying routing is what we try to capture, model, and quantify with our analysis. We conclude that H-cube poses two different optimization problems as its routing structure defines a specific optimization function for each of the two metrics of **latency (*Lt*)** and **combined communication cost (*CCost*)**. In [22] we defined these metrics (scaled down by *K*) and in this work we focus on minimizing them:

$$\boldsymbol{CCost} = \sum_{j=1}^{d} \sum_{i=1}^{2^d} R_{i,\varphi(i \oplus 2^{j-1})} \quad (1),$$

$$\boldsymbol{Lt} = \max{}_i \{ \sum_{j=1}^{d} R_{i,\varphi(i \oplus 2^{j-1})} \} \quad (2),$$

We see that the solutions to the metrics that correspond to H-cube are mapped to an *NP-complete Traveling Salesman Problem* (TSP) version with a number of *constraints*. These constraints diversify the solution to H-cube metrics from solutions that correspond to the rest of KA protocols. Finding approximations of the optimal solutions for the latter metrics is the only feasible way to improve the protocols.

V. HCUBE PHYSICAL MEMBER GRAPH: NOVEL HEURISTICS

As shown in [22], we created an auxiliary framework that includes the generation of a tree that spans all *n* members of the secure group, to approximate the simple TSP. This spanning tree (ST) has the following property by definition of the associated network graph *G*: the weight of any link that directly connects any two tree members is 1. In this case, the ST is in fact a minimum ST (MST). This equivalence allows us to use approximations based on the existence of an MST over the group members. We simulate Hamiltonian paths and cycles, by just performing a full walk of the rooted ST. Any of the well-known tree visits traverses every edge exactly twice, resulting in a cost twice the number of tree members. Using these approximations to set up a *td* schedule, the performance of all the studied KA schemes improves by at least a factor of *D* or *n*.

VI. LOGICAL HCUBE MEMBER GRAPH – NOVEL HEURISTICS

A. Logical Group Member Graph case: Approach.

We now allow non-member relays as well in the routing path between two group members. We assume a generic Dijkstra routing protocol that finds the shortest paths between members. Through the underlying routing, each member obtains the routing path(s) to its closest neighbor(s). We dynamically determine the proximity with respect to the number of hops between two members. If no neighbors are found in the proximity, the search diameter (TTL) is gradually expanded until a pre-agreed number of members are found. Assuming that a direct link between two network nodes has weight 1, the virtual link between two virtually connected members may take any value $x \leq Th$ (*TTL*). Hence, we allow the existence of arbitrary weights between two "connected" group members.

In [22] we generated a ST formed strictly by the *n* group members. The value of any link weight was always 1, and an MST would coincide with any ST. Implementing a full walk over an MST guarantees solutions that are better or equal than twice the optimal. The optimal in that case was the size of the secure group *n*, which was indeed what we were after. However, in the current setting, the same approximations would provide us with solutions that are better or equal than twice the size of the virtual ST, or MST with arbitrary weights, which could be the size of the whole network in the worst case. Moreover, the edge weights between two virtually connected tree members depend on the size of the minimum path shared which is most likely greater than 1, unless there is a direct link between two members. In this case, *a ST is no longer equivalent to an MST*. So, *in our new setting we will need to compute an MST* and not just any ST. In fact, *we will need to find the MST of all MSTs originating from every single group member*, if we want to apply an equivalent to our previous method. Using a core framework that computes the MSTs from all group members is an over-kill. This factor discourages us from applying the previous techniques to our new setting.

Next, we introduce a novel algorithm that redefines a new communication schedule and consists of several heuristics. Only one of these heuristics, this designed for the MST generation, has been already introduced in [25]. Although it is described there in detail, we will provide a brief overview in VI.*B* for the sake of completion. All other heuristics are introduced here for the first time, and we are going to provide detailed descriptions and analysis in section VII.

B. Auxiliary Scheme: MST Generation [25]

We generate a MST starting from any member, by applying a distributed version of *Prim's* method [24], which is based on a greedy strategy, captured by a generic algorithm which grows the MST one edge at a time. To implement the algorithm as such, all members must have global information of the link weights of all other members. We adjust the algorithm to our distributed environment, by having each member that joins the current ST instance report its candidate links to the root. At each step, the root determines the next member *J* to join by examining all unused candidate links of all members that belong to the current tree. Then the root sends a *Join Flag* to member *J*, then *J* joins, etc. The improved running time of the algorithm is shown to be: $Lt1 = O(E + V\log_2 V)$.

C. MST Manipulation [25]

Pre-order MST traversal: The *CCost* expression for GDH.1-2 is minimized if inequality $R_{n-1,n} \leq R_{n-2,n-1} \leq .... \leq R_{1,2}$ holds. In GDH.1-2, the messages communicated successively in the up-flow stage (parameter *i*) is incremental. Hence, the routing paths of successive members should be selected to be non-increasing. If we fix the GDH.1 *backward schedule first,* so that the first edge selected in the MST is assigned to relay the maximum number of messages (*n*-1), the second edge to relay (*n*-2) messages, etc., the "non-incremental" requirement is satisfied. We want to identify the *appropriate* traversal method to visit all vertices of a ST and establish the GDH.1 *backward schedule first*. By examining the common traversal methods, we select the *pre-order* tree walk. An intuitive reason for this is that a *pre-order* tree walk visits the root before the values in either sub-tree. So, it uses a greedy approach by adding the "best nodes" first, the earliest possible in the backward schedule. By visiting the vertices indicated by the *pre-order* walk backwards, we obtain the forward GDH.1 schedule.

Heuristic Overview: We start with the following observation: *if the generated MST was in fact a chain, then the desired Hamiltonian path would be directly provided and would result in the same cost as this of the MST*. The more the resulting MST resembles a single chain, the less the cost of the resulting Hamiltonian path is. So, we manipulate the MST using the following *transformation*: During the formation of the MST the *two longest distinct paths* from all group members to the root are located. The group member that marks the end of the longest path becomes now the new root of the transformed MST, and the associations between parents and offspring in the existing MST are sequentially altered to accommodate the transformed tree. This process results in *unfolding the MST* to its longest path or else in *extracting* the largest "path" from the MST (Fig. 2(*b*), 2(*c*)). Next, each member that belongs to the new ST, recursively rearranges its offspring in the order of decreasing distances from their tree leaves (Fig. 2(*d*)). Obviously, the backbone of the tree, which is the previously unfolded path, is accessed first by a pre-order tree traversal. So, if we generate a Hamiltonian path from this ST, *all members that belong to the "unfolded" path will be visited only once*. This modification results in the reduction in the routing overhead for the Hamiltonian path formed. Among siblings, the following invariant is true: the higher the newly assigned *id* of a sibling (parameter *i*), the fewer hops (relays) a message originating from this sibling will go through until the destination is reached.

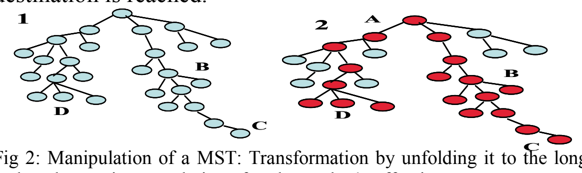


Fig 2: Manipulation of a MST: Transformation by unfolding it to the longest path and recursive re-ordering of each member's offspring.
Fig 2(*a*): Initial MST Prim. Fig 2(*b*): Identification of the largest path to unfold

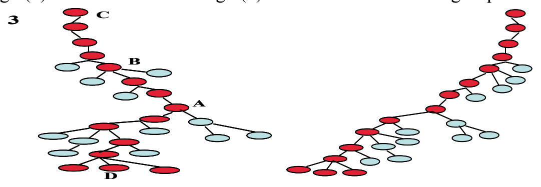


Fig 2(c): MST unfolded to its largest path
Fig 2(*d*): Re-arrange offspring from smaller to largest path

Notation_3: Let $R_{max} = max_{i,j}(R_{j,\ i})$ be the longest virtual link between any two virtually connected members.

VII. *WT*-ADAPTATION OF H-CUBE ON LOGICAL MEMBER GRAPH

In this section, we introduce a new heuristic for generating an efficient *wt*-H-cube. We assume that a starting point is pre-agreed, and as the network operation progresses, the leader election process provides members with new and auxiliary starting points. The knowledge of the starting point is propagated to the rest of group members via a simple broadcast tree. We provide a detailed description of the algorithm that generates the core framework and is used for the assignment of session *id*s to the group members.

We will identify the 2^d parties on the *d*-dimensional space $GF(2)^d$ by selecting such a basis $b_1, \ldots, b_d \in GF(2)^d$ for the direction of communications per round, that results in optimizing the desired metrics of interest *CCost*, *RCost*, *Lt*. The previous method (MST manipulation) does not apply to this case because the group members communicate in pairs. If we used an MST, we would also need to define a deterministic method for extracting pairs out of an MST, which is not straight-forward. Furthermore, the selection of peers for the 1st H-cube round and the selection of a unique basis in $GF(2)^d$, limits the degrees of freedom for the selection of peers for the subsequent rounds. So, similar solutions for the subsequent rounds may not apply in most cases. What is more, the initial selection of peers, even though optimal for the current round, may prove to be very unfortunate for the overhead incurred, after the end of all *d* rounds. Therefore, the use of MSTs as a core framework is highly unlikely to produce efficient H-cube schedules. This solution is abandoned, and we are looking for more lightweight frameworks that can potentially improve the metrics of interest even further. It is clear how complex the problem is. The optimal solution can be found with the exhausting method of trial and error, but this is out of the question, and in particular for the environment of interest.

A. Overview

The goal of our algorithm is to determine the most efficient pairing for the 1st round. In other words, the peers selected for the 1st round must considerably improve the metrics of interest. Towards this end, we use a greedy strategy under which each member makes the best matching selection possible (1st Pass) and then a "corrective operation" is initiated (2nd Pass) that goes around "dead-ends" and ensures that all members obtain a peer. We virtually place each member of a 1st round pair in one of the two columns: *left* or *right*. The selection of these peers is close to the optimal for the given round, as we will show next, irrespectively of its impact to the subsequent rounds. However, we keep the message exchanges between the peers of this round active in every subsequent round, by doing the following modification to the original H-cube: only half of the initial *n* group members, those either in the left or in the right column, participate actively to all subsequent rounds, as designated by the original scheme. The members that belong to the non-active column become passive recipients in the subsequent rounds. Without loss of generality, we assume that the members in the *left* column are the *active* participants and those in the *right* column become the *passive* ones. The *active* members are assigned new sequential session *id*s $\in [0, \frac{n}{2}\text{-}1]$ after the 2nd Pass. Before the start of the following rounds, the active members decide on a unique basis, and execute H-cube for the next (*d*-1) rounds. In our case, we select a basis that designates the new pairs at any given round *j*, where $0 \leq j \leq (d\text{-}2)$, according to the following formula: in round *j*, party *i* interacts with party $i \oplus 2^{j\text{-}1}$. In each such round, each active member communicates the newly calculated blinded value to its corresponding passive peer (member of right column). So, all group members participate indirectly to all rounds, since all receive the intended KM data, and process it as indicated by H-cube. In the end, all members obtain the same group key.

Given the previous observation, we can apply our algorithm on every round and not just the first one. Then, from the active parties of round *j*, only half remain active during round (*j*+1) and the other half become passive. Our greedy algorithm of two passes is applied once again to the currently active members, to determine the lowest weight pairings among them. So, instead of using a pre-agreed common basis for the direction of communications, we dynamically construct such a basis, based on the topological proximity of the active group members. However, this approach only works, if we de-activate half of the active participants of the current round, for all subsequent rounds. However, applying the heuristic as many times as the total number of H-cube rounds is an over-kill. Also, the resulting protocol is quite different from H-cube in nature, with features that may not be desirable for our network. An efficient and flexible solution is to apply the heuristic only for the first few R rounds. Threshold R can be dynamically set.

Given these considerations, we conduct the following analysis by setting $R = 1$, to limit the overhead from the backbone framework. We will show that H-cube performance improves substantially by this modification alone. Having defined our generic algorithm, it is trivial to change R and measure the improvement in the protocol. In this work, we find it sufficient to present only the results collected from $R = 1$, since: (*a*) the results are very satisfactory and the overhead of the auxiliary framework is low, (*b*) by increasing R the overhead of the H-cube application is decreased while this of the backbone framework increases, and (*c*) by increasing R the H-cube nature is shifted from totally distributed to centralized.

1st Pass (Greedy Approach):

This process resembles the construction of an MST in that it follows a greedy strategy as well. The main difference is that only one member or one pair of members (that already belong to the structure being generated) is allowed to select the next candidate that will join the structure generated during the 1st Pass. Ideally, in the case that no corrective process is required (2nd Pass) the outcome of the 1st Pass is the following: a peer for every member is designated, with respect to the proximity among nodes. We denote the outcome of the 1st Pass as Set1. Towards the construction of Set1, a single token is circulated among members, and at each step, the token is passed to one member only. The details of the process follow:

Initially, the starting point is handed the token for the first time. The first time that a member is handed the token, will attempt to make the best selection available and find the closest in terms of number of hops peer. Assume that member A currently possesses the token. If there is such a peer F available, then members A and F will form a valid pair for the process, schematically represented with a horizontal line in the virtual sketch of Set1. If A is the starting point, there is always a peer matched to it, since all selections are available at this point, and also every member is “connected” to a number of members in a “connected” graph. If A is not the starting point, then there is a chance that its nearest neighbors have been reserved by other members, and A cannot find any peer available. We distinguish two cases:

Case (*a*): A finds peer F available: A enlists F as its peer and together form a horizontal line on the virtual sketch of Set1 (Fig. 3). Now both A and F merge their proximity lists (those that contain the members in their proximity) and arrange their elements in increasing order of hop distance. The exchange of merged lists has size $2D \times K_S$, where D is the upper bound of elements in a proximity list, and K_S is the bit size of any of the control messages. Member A still holds the token, and will give it to the first member from the merged and ordered proximity list that is still available (i.e. not already a part of Set1). Hence, A sends a *Join* control Flag to each member in the merged list sequentially, until it finds the first available member (say H), in which case it receives an *Accept* control Flag. Member H broadcasts this *Accept* control Flag and information about its new peer to all neighbors in its own proximity list. This way, neighbors know the status of H beforehand and do not need to query it when they become initiators. Members that are unavailable respond with a *Refuse* control Flag.

Then, member A provides the token to H. Member H becomes the initiator of a similar process, in order to find a peer of its own, and so on and so forth. If however, H does not find a peer, it gives back the token to its ancestor, member A. H becomes “grounded”, and it is represented via a “vertical line” on the sketch of Set1 (Fig. 3). Member A continues scanning its merged list and holds the token until it finds another available member (say member C) that will produce a peer (say member G). Schematically, member C is placed at the end of a vertical link whose other end starts from the middle of the horizontal link formed by members A and F. Members C and G form a horizontal link. Any successor of the merged lists of C and G (grounded or paired) will be placed on a vertical link whose other end starts from the middle of the horizontal link formed by C and G. If the merged list is exhausted and member A has not found a successor to hand the token to, then it recursively gives the token to its predecessor. The same method is followed until all members are visited. As long as the members are connected as defined at the beginning, this method produces no deadlocks, i.e. all members are visited before the process ends. Assume that S_1 is the subset of members that have been visited during the 1st Pass, and S_2 is the subset of members that could not be visited during the 1st Pass. At least one member in S_2 must have a link to one or more members in S_1 otherwise the graph is not connected. Let this member be J. Now, all members in S_1 are part of either a horizontal line or a vertical one. Clearly, no members that belong to a vertical line have a link to member J, otherwise such a member could form a pair with J on a horizontal line. The members that hold the token exhaust their merged lists until they find a successor. If at some point the process stalls, in the sense that no successors can be found, the token recursively goes up the whole structure (Set1). At each step, all elements in the associated merged lists are scanned and the neighbors of all members that belong to Set1 are examined. Then, the link to J is eventually found, and J joins S_1 and is also handed the token for the first time. So, all members are eventually visited.

Case (*b*): Member A does not find any peer available: As stated in the previous case, A provides the token to its predecessor member, which follows the same process and recursively provides the token to its own predecessor under case (*b*) or to a successor under case (*a*). The predecessor has already found its peer, and consequently, it looks through its merged list to find the next potential element to provide the token to. The process ends when all members have been visited, and the token recursively goes back to the starting node.

2nd Pass (Corrective Operation):

The purpose of the 2nd pass is to locally re-arrange the grounded members and the pairs established during the 1st Pass so that all members obtain a peer at the end of this process. This time we access the structure (Set1) obtained after the end of the 1st Pass from down-up and we build the final H-cube structure, denoted as Set2. We start with the horizontal links that have the following property: all members vertically attached to them (if any) are grounded. We denote the horizontal links with this property as *leaf* links. We start by re-arranging all associated members with the leaf horizontal links. Then, we go up one predecessor horizontal line at each step, until we reach the top, i.e. the horizontal link formed by the starting point. All paths starting from all leaf horizontal links eventually end up to the starting point. We denote the members that have been given the token and have also designated a peer for themselves during the 1st Pass as *initiators*. So, we start with leaf initiators and we go up one initiator at each step until we reach the starting point. The initiator is always paired and represented as a vertex on a horizontal link. During this pass however, we re-arrange the initiator, its peer, and all their vertical links, when they become ready. After this step, the initiator may end up either without peer or with a different peer.

Each member that lies at the end of a vertical link (initiator or grounded) uses two control flags: the *ActPass* Flag and the *Arranged* Flag. If for a member *ActPass* = 1, then the member is active at this point during the 2nd Pass process, and actively participates to this step of the algorithm that involves it. If *ActPass* = 0, then the member becomes inactive for all subsequent steps of the process. When *ActPass* is switched to 0 it remains 0 throughout the process. If *Arranged* = 1, then the status of this member is determined for the current step of the algorithm. The status of a member is determined by the value of *ActPass* flag. If *Arranged* =1 and *ActPass* = 1, then the member participates in the step that involves it actively, because no peer has been assigned to it. If *Arranged* =1 and *ActPass* = 0, then the member has been assigned a permanent peer, and consequently it is not active for the step that involves it and any subsequent step. In this case, the status of a member does not depend on the outcome of the execution or previous steps of the algorithm. If however *Arranged* = 0, then the status of the *ActPass* flag is unknown. Then, the member must wait for the outcome of the step that involves the initiator of the previous step, virtually placed below its own initiator. For example, initially, all grounded members participate in the process with the following values in their control flags: *ActPass* = 1, *Arranged* = 1. This is because the status of the grounded members is known – they have no peers assigned – and they will actively participate in the step that involves them. The matched peers of the initiators from the 1st Pass are also going to participate in the step that involves them, and they set their flags to the same values as these of the grounded members. However, the status of the initiators (the matched members that are represented as vertices on the left of the horizontal links) is unknown. Initially, they participate in the step that involves them with the following values in their control flags: *ActPass* = 1, *Arranged* = 0. After getting feedback from initiators that lie below them, they change the values of their control flags to: *ActPass* = {0, 1}, and *Arranged* = 1. We now give a detailed description of a single step of the 2nd Pass algorithm:

At any step of the 2nd Pass, every initiator monitors its own horizontal line until all vertices placed at the other end of the vertical links have set their *Arranged* flag to 1. Then, the status of all associated members (grounded, initiators) is known and the process that will determine the status of the current initiator can now be activated. The initiator scans all these members to see how many have set the value of the control flag *ActPass* to 1. Those with *ActPass*=0, have obtained permanent peers from the previous steps and do not participate to the process any longer. So, they can be ignored by the current initiator.

(*a*) If the number of members with *ActPass* =1 is *odd*, then the current initiator will obtain a permanent peer in this round, and will become inactive for the remaining steps. So, the initiator can set its own control flags (previously set as follows: *ActPass* =1, *Arranged*=0) to the following: *ActPass*=0, *Arranged* =1.

(*b*) If the number of members with *ActPass* =1 is *even*, the current initiator will *not* obtain a permanent peer in this round, and will become active for the remaining steps. Hence, the initiator can set its own control flags (previously set as follows: *ActPass* =1, *Arranged* = 0) to the following values: *ActPass* = 1, *Arranged* = 1. In this case, the initiator gives priority to all the rest of members in its vicinity to obtain peers. This can be done, since the number of these members is even. The peer that was previously matched to this initiator will now be matched to one of the members that lie on the vertical links.

The current initiator, say *A*, determines the matching of its associated members according to the following method: first, it matches its ex-peer, say *B*, with its closest neighbor based on *B*'s proximity list. Then, *if* case (*a*) holds, it matches itself with the closest neighbor available, from its own or the merged ordered proximity list. It matches the vertices (members) on the vertical links, with the following algorithm:

Member *A* scans its own proximity list, and the first two elements available are always matched together. Then, the next two remaining elements are matched together, and so on and so forth. Then, member *A* scans the proximity list of member *B* and matches the first two available elements every time, until all or all except the last elements (if the number of elements is odd) are exhausted. Finally, the two remaining elements on each of the two proximity lists of *A* and *B*, (if any) are matched together. The reason behind this is the following: the two elements that are matched together are always those that are available and have the minimum hop distance from their associated horizontal link. This means that they both have the minimum hop distance either from the initiator or from its ex-peer (except for the last remaining pair). The idea is that these distances set up a threshold for the maximum distance between the newly matched members from the vertical links. This threshold is the min. possible compared to a different matching.

Now all members associated with a given initiator (and the associated horizontal link) are matched. Since the initiator signals its known status (*ActPass*, *Arranged* = 1) to the upper level, the initiator associated with the upper level is ready to perform exactly the same steps and accommodate all members associated with it, following exactly the same approach. At the end, all members are accommodated if the number of members is even. This is so because the process for any given level is not executed unless all associated levels below this are accommodated (i.e. all members in them are paired).

B. Hypercube Structure Manipulation

After the 2nd Pass ends, all members are matched in pairs. This would suffice if we were strictly interested in improving the performance of the 1st H-cube round only. However, we still want to improve the overall performance of the protocol. The H-cube structure becomes equivalent to a ST if we map every pair of members to one tree node. In fact, that is what we do in reality, since in all subsequent rounds after the 1st, only one peer of every pair participates, hence we want to generate a communications schedule only for the members that are active after the 1st round. The other half (inactive after the 2nd round) have been accommodated for the remaining rounds: they remain recipients of the BKs computed by the active senders, during all rounds. Only these active members participate in H-cube structure manipulation and obtain new *ID*s. The resulting H-cube structure does not only provide pairs for the 1st round, but connects all active members via a structure exactly similar to this of MST. We will use the previous methods for the MST manipulation to provide a schedule that improves its performance. The method is similar as before: we apply the MST manipulation to H-cube (fig.3). We unfold the structure to its longest path, recursively arrange the offspring of each node in this structure in decreasing order of hops, and perform a pre-order traversal of the transformed structure to assign the new *id*s to the active members. The improvement is similar to this of the previous schemes. As for the subsequent rounds, there is still improvement compared to an arbitrary assignment of *id*s, but the higher the rounds, the more uncertain this improvement becomes. In reality, the resulting values for these metrics may be much lower, since members may find paths of lower distances than these designated by the transformed structure.

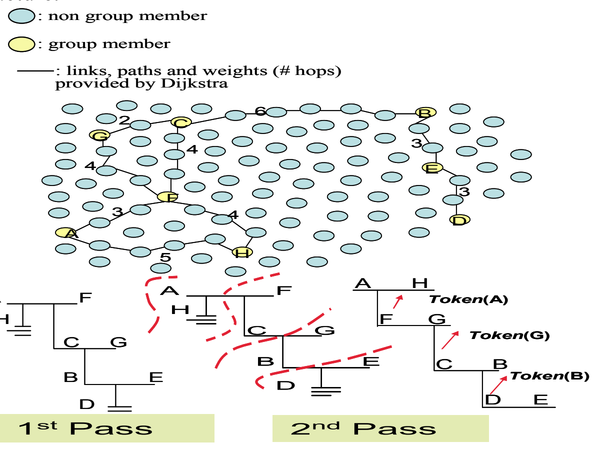


Figure 3. Example of the formation of a H-cube structure from a network graph with arbitrary configuration of the group members: The 1st and 2nd pass are first executed on the network, and if the structure contains more than one path, MST manipulation is triggered.

C. Performance Analysis

1st Pass: All subgroup members except for the peers of the initiators become initiators themselves and attempt to find a peer or a successor initiator. Let the number of subgroup members be n. The peers of initiators are denoted as $x < \frac{1}{2}n$. The *Join* flag is sent only from the initiators to the members in their proximity lists that are scanned. Members that *accept* the request to *join* broadcast their decision and make it known to their proximity lists. By making use of the multicast advantage, the *Accept* control flag packet is transmitted once to their proximity, but goes through all the virtual links reported in their proximity lists. Initiators may scan more than one member, in order to find one to join (because they are using the merged proximity lists, and cannot directly check the status of certain members). Hence, we assume that in average, an initiator scans half of the members reported in its proximity list. Members that get scanned respond by "Refuse" until one that *accepts* is found. The overhead incurred from the exchange of control flags and other data is computed as follows:

Join Flag: sent by (n-x), received by $\frac{1}{2}(n\text{-}x)D$ members at max.

$$CCost\,(Join) < \sum_{i=1}^{n-x} \frac{D}{2} max_j R(i,j) \times K_S.$$

Refuse Flag: transmitted at max. by $\frac{1}{2}(n\text{-}x)\,D$ members.

$$CCost\,(Refuse) < \sum_{i=1}^{(n-x)} (\frac{D}{2}-1)\max_j R(i,j) \times K_S.$$

Accept Flag: broadcast by all members but the starting point.

$$CCost\,(Accept) < \sum_{i=1}^{n-1} D \max_j R(i,j) \times K_S.$$

Exchange of Proximity Lists: The x pairs exchange and merge their proximity lists. Also, let $PK_S = D \times K_S$ denote the bit size of the packet that carries a member's proximity list. Then:

$$CCost\,(List\ Exchange\ LE) = 2\times \sum_{i=1}^{x} \max_j R(i,j) \times PK_S,$$

$$Latency{:}\ Lt < \sum_{i=1}^{x} \frac{D}{2}\max_j R(i,j) \cdot$$

Therefore, the total *CCost* due to 1st Pass becomes:

CCost (*1st Pass*) = *CCost* (*Join* + *Refuse* + *Accept* + *LE*).

$$CCost(1^{st}Pass) < [2\times \sum_{i=1}^{n-x} \frac{D}{2} max_j R(i,j) - \sum_{i=1}^{(n-x)} \max_j R(i,j)$$
$$+ \sum_{i=1}^{n-1} D \max_j R(i,j) + 2\times \sum_{i=1}^{x} D \max_j R(i,j)]\times K_S$$

$$CCost\,(1^{st}Pass) < [D\,(2n{+}x{-}1) + (x{-}n)] \max_{j,i} R(i,j) \times K_S.$$

2nd Pass: Whenever the status of all members locally associated with one paired initiator becomes known, the initiator matches all members that are available in its merged proximity list. It individually notifies each member about its new peer. The merged lists may contain at maximum 2D elements. However, some of the elements are duplicates, or in practice each individual list contains less than D elements. To simplify the case, if we assume that each proximity list contains the same number of active members, then there are $y = \frac{n}{2x} < 2D$ active elements in each. Hence, each initiator notifies (y+1) members of their new peers. The status of an initiator becomes known, when all the members of all initiators that are "schematically" found below the given one, are accommodated. The overall latency of the 2nd Pass is defined by the longest path of initiators, from the starting point to any leaf initiator. We can now compute the metrics related to the 2nd Pass algorithm:

$$CCost(2^{nd}Pass) < \sum_{i=1}^{x} y \max_j R(i,j)\ PK_S =$$
$$\sum_{i=1}^{x} \frac{n}{2x}\max_j R(i,j)\ PK_S < \frac{n}{2}\times D \times \max_{j,i} R(i,j)\, K_S$$

$$Lt\,(2^{nd}\,Pass) = \max_Z \sum_{i=1}^{z} \frac{n}{2x}\max_j R(i,j)\text{, where } z \le x.$$

MST Manipulation: Here, only half the members actively participate. The HCube structure replaces the MST in the previous heuristic and the overhead results from manipulating the structure similarly to an MST. We quote this overhead, which has been computed in [25], and adjust it to the current HCube MST framework (HMST):

$CCost\ (HCube\_aux) = 2\times Wgt(HMST)\times K_S$,

$Lt\ (HCube\_aux) = 2\times Wgt(P_{\max})$, where P_{max} is the max. path in the unfolded HCube structure that consists of $\frac{1}{2}n$ members.

We now compute the overall overhead of the auxiliary scheme:
$CCost(aux) = CCost\ (1^{st} + 2^{nd}\ Pass)\ + CCost\ (HCube\_aux)$.

$$CCost(aux)<[(D(2n+x\text{-}1)+(x\text{-}n))\max_{j,i} R(i,j) + \frac{n}{2}D\max_{j,i} R(i,j) + 2\frac{1}{2}n\max_{i,j} R(i,j)\,]K_S$$

$$CCost\ (aux) < (3D+1)\times n\times \max_{j,i} R(i,j)\times K_S.$$

$$Lt\ (aux) = Lt\ (1^{st}\ Pass) + Lt\ (2^{nd}\ Pass) + Lt\ (HCube\_aux).$$

$$Lt(aux)<n^*(\frac{D}{4y}+\frac{1}{2})^*\max_{i,j} R(i,j) < \mathrm{O}(n^*\mathrm{O}(1))\ ^*\max_{i,j} R(i,j)$$

D. Simulations

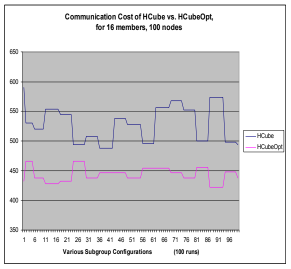
Simulations Set-Up: We compare the routing cost of *wt vs.* original HCube over ad hoc multi-hop networks. We use various topology graphs to generate the secure subgroups and analyze the performance of the heuristics. Our network graph represents a single cluster area where a single secure group is deployed. A number of nodes from this graph are randomly selected as group members. The group leader is randomly selected as well. At the end of the group "registration" period, the sponsor piggybacks the list of the legitimate members into the routing packets. As discussed earlier, we assume that generic Dijkstra routing finds the shortest paths between members. Through the underlying routing, each member obtains the routing path(s) to its closest neighbor(s). The proximity between two members is dynamically determined.

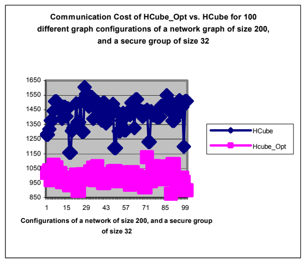
We assume that while the backbone framework is being formed, the relative placement of members and hence the proximity lists do not change significantly. Such a change could result in a different "optimal" solution, and the one currently generated would become outdated. It is expected that the higher the nodes' mobility, the worse the performance of our algorithm is. Even though the *wt*-version is more sensitive to mobility and may be operating sub-optimally, it still outperforms the original. On the other hand, the backbone can be periodically reconfigured to capture all dynamic changes, with frequency that depends on network dynamics, resources available, and our own requirements. For example, the auxiliary framework may be recalculated whenever protocol performance degrades to the median of the best and the average execution of the *nt* scheme.

For our evaluation, we generated various random graphs for a given input of the number of nodes *n* and the number of members *m*. For the same graph and the same input, we have varied the subgroup configuration, i.e., we have selected the *n* members in a random manner. For each graph of input <*n*, *m*> and for each subgroup configuration, we have evaluated the three metrics of interest (*CCost*, *RCost*, *Lt*) of the *wt*-versions *vs.* the *nt*-versions (original), and we have averaged the results for all random graphs with the same inputs <*n*, *m*>. We have tested the following cluster-subgroup scenarios:
Cluster Size: [50 … 800], **Subgroup Size**: [8,…64].

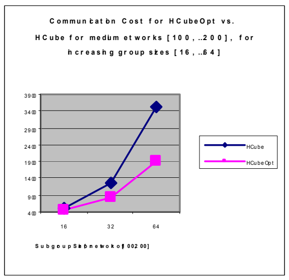
Simulation Results: We illustrate some indicative results on *CCost* and *RCost* produced by the original HCube and its *wt* version, measured the total number of hops required for the protocol to successfully terminate. The following graphs reflect *CCost* of the *wt*-version *vs.* the existing under various scenarios of secure group and network size, scaled by a factor *K*. *CCost* and *RCost* are significantly reduced in all the scenarios captured. Indeed, in most cases the improvement ratio becomes: $R_{COMM} = \frac{CCost(ING\_Opt,n,S)}{CCost(ING,n,S)} < \frac{1}{2}$. In fact: $0.32 < R_{COMM} < 0.7$. To illustrate these with an example, for a subgroup of size 16, and a network of size 200, the average relays produced are 561 for HCube_Opt, and 1604 for HCube, and $R_{COMM} = 0.35$.

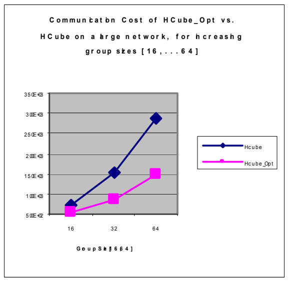
We verify that the costs of the *wt* auxiliary framework of the *wt*-version match our analytical results. Indeed, we re-call the expression that estimates the *CCost* (*aux*) metric. We assume that the maximum number of neighbors of each member is around 10-12, and set $K_S = 8$. Also, we select $max\ (R_{j,\ i}) = 8$, and $D = 16$. Then, we obtain: $CCost\ (aux) < 3200\times|V|$. By computing this expression for a group of 32 members we obtain: $CCost\ (aux,\ V{=}32) < 102{,}400$ bits. Indeed our simulations results verify that this upper bound holds, since some indicative values of *CCost* for group size *n* and network size *S* are the following: $CCost\ (aux,\ n{=}32,\ S{=}200) = 98106$ bits, $CCost\ (aux,\ n=32,\ S=300) = 100725$ bits, $CCost\ (aux,\ n = 32,\ S = 400) = 64136$ bits. The same is the case with the rest of the simulated group sizes. *CCost* (*aux*) increases as the network size grows, until some threshold value is reached. Then, the metric starts decreasing. This happens because two members are "connected" until the hop distance between them reaches a certain threshold. After the threshold is exceeded, the members become disconnected. As the network size increases, the density of group members decreases, and naturally each member's neighbors decrease. The less the neighbors of each node are, the lower the overhead for the auxiliary framework is. We also verify that the metrics associated with the auxiliary framework are kept low and add little to the overall overhead produced by the execution of *wt*-HCube.





Figs 1, 2. *CCost* of HCubeOpt *vs.* HCube for: (1) for 100 different group configurations on a network of 100 nodes (100 different runs) for a group of size 16, (2) for 100 different group configurations on a network of 300 nodes (100 different runs) for a group of size 32. HCube_Opt results in substantially superior performance for all the different configurations, in the two scenarios illustrated (better than this achieved with the previous *wt*- schemes).





Figs 3, 4. *CCost* of HCubeOpt *vs.* HCube w.r.t. the size of the group [16, 32, 64] in a network of (*a*): [100, 200] nodes, (*b*): [200, 500] nodes. HCubeOpt reduces the overhead in half. The performance difference grows even bigger as both the size of the group and network grow. So, HCube_Opt not only presents superior performance, but scales much better as well.

VIII. Conclusions

In this section, we contributed towards the design and analysis of more sophisticated approximation - optimization methods of the communication, routing, and latency functions (metrics) of the Hypercube key agreement protocol. We developed new heuristics for generating a novel topology-aware communication schedule to be used by the protocol. The heuristics achieve significantly better approximations of the metrics we want to optimize. Our comparisons of the new topology oriented and the original Hypercube are done via simulations. The new protocols achieve a dramatic overhead reduction. We believe that there is scope for even more efficient topology oriented approaches and we consider efficient simulations of KA protocols over multi-hop ad hoc networks as an interesting open problem.

References

[1] K. Becker, U. Wille, “Communication Complexity of Group Key Distribution,” *Proc.5th ACM Conference on Computer & Communicatios Security*, pp. 1-6, San Francisco, CA, November 1998.

[2] M. Steiner, G. Tsudik, M. Waidner, “Diffie-Hellman Key Distribution Extended to Groups,” *3rd ACM Conference on Computer & Communication Security*, pp. 31-37 ACM Press, 1996.

[3] W.Diffie, M.Hellman,”New directions in cryptography”, *IEEE Trans. on Information Theory*, 22(1976), 644-654.

[4] I. Ingemarsson, D.Tang, C.Wong. “A Conference Key Distribution System”, *IEEE Trans. on Information Theory*, 28(5): 714-720, Sept. ‘82

[5] M.Burmester, Y.Desmedt. “A Secure and Efficient Conference Key Distribution System”, *Advances in Cryptology–EUROCRYPT’94, Lecture Notes in Computer Science.* Springer – Verlag, Berlin, Germany.

[6] M. Hietalahti. “Key Establishment in Ad-Hoc Networks,” *M.S. Thesis*, Helsinki University of Technology, Dept. of Computer Science and Engineering, May 2001.

[7] A.Perrig, “Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication,” *Int’l Workshop on Cryptographic Techniques and E-Commerce (CrypTEC’99),* pp. 192-202, July 1999.

[8] N.Asokan, P. Ginzboorg, “Key-Agreement in Ad-Hoc Networks,” *Computer Communications*, Vol. 23, No. 18, pp. 1627-1637, 2000.

[9] Y. Kim, A. Perrig, G. Tsudik, “Simple and Fault Tolerant Key Agreement for Dynamic Collaborative Groups,” *Proc. 7th ACM Conf. on Computer and Communication Security (CCS 2000),* pp. 235-244.

[10] J. Katz, M.Yung, “ Scalable Protocols for Authenticated Key Exchange“, *Advances in Cryptology - EUROCRYPT’03, Springer-Verlag*, LNCS Vol 2729, pp. 110-125, Santa Barbara, USA.

[11] J.Katz, R.Ostrovski, A.Smith, “Round Efficiency of Multi-Party Computation with a Dishonest Majority”, *Advances in Cryptology, EUROCRYPT’03*, LNCS Vol. 3152, pp.578-595, Santa Barbara, USA.

[12] Y.Amir, Y.Kim, C.Rotaru, J.Schultz, G.Tsudik, “Exploring Robustness in Group Key Agreement”, *Proc. of the 21th IEEE Int’l Conference on Distr.Computing Systems,* pp. 399-408, Phoenix, AZ, April 16-19, 2001.

[13] Y.Amir, Y.Kim, C.Rotaru, J.Schultz, J.Stanton, G.Tsudik, “Secure Group Comm/tion using Robust Contributory KA”., *IEEE Trans. on Parallel and Distributed Systems*, Vol. 15, no. 5, pp. 468-480, May ‘04.

[14] L.Zhou, Z.Haas, “Securing Adhoc Networks,” *IEEE Network Magazine*, vol. 13, no.6, pp. 24-30, Nov/Dec 1999.

[15] J.Kong, P.Zerfos, H.Luo, S.Lu, L.Zhang, “Providing Robust and Ubiquitous Security Support for Wireless Ad-Hoc Networks,” *Proc. 2001 IEEE Int’l Conf. on Network Protocols (ICNP 2001)*, pp. 251-260.

[16] S.Capkun, L.Buttyan, J.Hubaux, “Self-Organized Public Key Management for MANET,” *IEEE Trans. on Mobile Computing*, Vol. 2, No. 1, pp. 52-64, Jan-Mar. 2003.

[17] L.Eschenauer, V.Gligor., “A Key Management Scheme for Distributed Sensor Networks,” *Proc. 9th ACM Conference on Computer and Communication Security (CCS’02)*, pp. 41-47, Nov, 2002.

[18] H.Chan, A.Perrig, D.Song, “Random Key Predistribution Schemes for Sensor Networks,” *Proc. 2003 IEEE Symposium on Security and Privacy*, pp. 197-213, May 2003.

[19] S.Yi, R.Kravets, “Key Management for Heterogeneous Ad hoc Wireless Networks,” University of Illinois, Urbana-Champaign, CS dept., TR#UIUCDCS-R-2001-2241, UILU-ENG-2001-1748, July 2002.

[20] S.Zhu, S.Setia, S.Xu, S.Jajodia, “GKMPAN: An Efficient Group Re-keying Scheme for Secure Multicast in Ad-hoc Networks”, *IEEE Computer Society, MobiQuitous 2004,* pp. 45-51.

[21] M.Striki, J.Baras "*Efficient Scalable Key Agreement Protocols for Secure Multicast Communication in MANETs*", Collaborative Technologies Alliance (CTA) Symposium, College Park MD, May 2003.

[22] M.Striki, J.Baras, G.DiCrescenzo, “Modeling Key Agreement Protocols in Multi-Hop Ad Hoc Networks”, *Proc. 2006 Int’l Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 39-45, Vancouver, CA, July 2006

[23] R. Gallager, P.Humblet, P.Spira. “A distributed algorithm for minimum weight spanning trees”, *ACM Transactions on Programming Languages and systems (TOPLAS)*, 5(1): pp.66-77, January 1983

[24] T.Cormen, C.Leiserson, R.Rivest,”Introduction to Algorithms”, *MIT Press, McGraw Hill Book Company*, March 1990

[25] M.Striki,J.Baras,K.Manousakis., “New Algorithms for the Efficient design of topology oriented key agreement protocols in multi hop ad hoc networks”, Procs of WiOpt’08, April 1-3, Berlin, Gernany.