A Robust, Distributed TGDH-based Scheme for Secure Group Communications in MANETs

by Maria Striki, John S.Baras

TR 2005-97

A Robust, Distributed TGDH-based Scheme for Secure Group Communications in MANETs

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

Abstract—**Securing multicast communications in Mobile Ad Hoc Networks (MANETs) is considered among the most challenging research directions in the areas of wireless networking and security. MANETs are emerging as the desired environment for an increasing number of commercial and military applications, addressing also a growing number of users. Security on the other hand, is now an indispensable requirement for these applications. However, the limitations of the dynamic, infrastructure-less nature of MANETs impose major difficulties in establishing a secure framework suitable for such services. The design of efficient key management (KM) schemes for MANET is of paramount importance, since the performance of the KM functions imposes an upper limit on the efficiency and scalability of the whole secure group communication system.**

In this work, we contribute towards efficient and robust secure group communications for MANETs by extending the TGDH protocol to a novel distributed and topology aware scheme: DS-TGDH. Our aim is to modify TGDH to: *a*) be feasible in the most general resource-constrained flat MANET where no nodes with special capabilities exist, *b*) produce considerably lower overhead for the network nodes involved, *c*) handle disruptions with low cost. We consider the underlying routing protocol in our design, and we apply a distributed TGDH version over a robust schedule, optimizing parameters of interest. We focus on the design and analysis of the "stealthy" TGDH and compare it with the original, w.r.t. this cross-layer consideration. Through our analysis and results we shed more insight on the actual feasibility of these protocols for MANETs and provide more realistic and fair comparison results that more accurately advocate the pros and cons of each protocol over the environment of study.

I. INTRODUCTION

A Mobile Ad Hoc Network (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 wireless multicast services such as cable TV, secure audio and conferencing, military command and control grows, the research on security for wireless multicast 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. So, the existence of a secure, robust KM scheme for multicast communications is essential. However, the characteristics of MANETs constitute the major constraint and challenge for the design of suitable KM schemes. We are dealing with dynamic, infrastructure-less networks of limited bandwidth, unreliable channels, where topology is changing fast. Network nodes may have limited capacity, computational and transmission power. Connections are temporary (mobility, battery drainage) and unreliable. These constraints render most of the existing KM schemes inefficient in MANETs: among other requirements, they need to catch up with a rapidly changing network topology, and handle failures at any time during group key establishment.

Along with the requirement to design secure KM schemes that achieve better performance than existing ones (either for wire-line or wireless networks), the need for the KM schemes to handle successfully and tolerate with low impact network dynamics and failures (robustness) in a network with large number of nodes (scalability) is now equally important.

In an attempt to meet all these objectives, two novel hybrid Octopus schemes MO and MOT [6, 19] were previously introduced and evaluated in addition to the original [1]. The special features of Octopus schemes have motivated our interest to explore and extend them. Hierarchy is supported through the partition of a large key agreement (KA) group to 2^d subgroups of smaller size. Initially, each subgroup agrees on its own subgroup key locally. Then, the subgroup leaders interact among themselves and use the previously generated subgroup keys to agree on a global group key via a core KA protocol, Hypercube [5]. Finally, the subgroup leaders distribute securely the group key to their subgroups. The superiority of MOT has been mainly attributed to the application of TGDH within the subgroup.

The primary focus of prior work was the analysis of the proposed schemes, based on the overhead resulting from the processing and exchange of KM information only. That was done in isolation from underlying backbone or auxiliary network functions (i.e. routing, clustering, leader election). However, since each of the schemes considered relies upon the former functions more or less, or in a different way, our previous evaluation is not totally fair. The consideration of network parameters that can accentuate their individual characteristics and the differences in their performance will provide more realistic results. For example, the hierarchical framework on top of which the three schemes are executed is supported by a clustering mechanism. Although the cost of adding and maintaining this framework adds to the overall cost of each individual scheme, it does not alter the outcome of the comparative evaluation of the schemes. On the contrary, the leader election and the backbone transmission schedule within the subgroup differ. For example, scheduling is not required for the subgroup in the centralized (O), in contrast to MO and MOT that members interact among themselves for the subgroup key generation. A simple schedule based arbitrarily on members' *ID*s is likely to result in unnecessary routing. Our main objective is lowering the overall bandwidth in the frameworks we design.

The need to apply the subgroup schemes in MO and MOT (GDH.2 and TGDH) over a schedule that optimizes metrics of interest comes into the play. The network assumptions made in TGDH [4], (e.g. leader reaches all members via a single broadcast), are too simplistic to apply to a general MANET. This way, the need to integrate the discussed functions in their design is mitigated. In real multi-hop networks, a cross-layer consideration appears to be essential for a concrete KM framework. Previous results have designated MOT as the most efficient for the environment discussed. It would be worth exploring if MOT still prevails after a re-evaluation of the same schemes under more realistic network assumptions.

In the present work, we assume that a path between group members may as well include non-member relays. We simply rely on the redundancy of the routing protocol to ensure that the exchanged messages are delivered in a timely manner. Dividing such group into subgroups, each one corresponding to a fully connected graph of members, reachable by the subgroup leader via a single broadcast, would permit the execution of the original "centralized" TGDH exactly under the assumptions described in [4]. This approach is impractical for a large group: a high number of subgroups will be formed, very sensitive to even subtle mobility changes, and it is quite likely that they will contain very few members, even a single one. Even if such subgroups are in very close proximity they still cannot be merged. The result is a considerable waste in network resources, and an infeasible execution of Hypercube, where the crucial parameter d is likely to be high and unstable. Then, the resulting inter-cluster signaling overhead outweighs the benefits of Hypercube for the inter-cluster communication.

In this paper we present an adaptation of TGDH to meet the requirements of a general MANET. In particular, we modify TGDH so that: *a*) it is made distributed, no single point of failure leader is required, *b*) it is executed under a schedule that optimizes our own defined routing and robustness metrics, under a cross-layer, topologically aware consideration, *c*) it tolerates failures and disruptions with low cost, *d*) it is far more efficient w.r.t. bandwidth and computation overhead. We denote this novel scheme as **DS-TGDH** (*Distributed TGDH with Schedule*) and evaluate both protocols under the new assumptions. Section 2 gives an overview of related work. Section 3 provides an overview of TGDH. Section 4 discusses fault-tolerance issues for our framework. In section 5 we introduce DS-TGDH, in section 6 we analyze its initial and steady state operations. Section 7, 8 present the analytical comparative performance evaluation and the corresponding simulation results. Finally, in section 9 we conclude the paper.

II. Related Work

The KM proposals for secure group communications abound in the literature. From the perspective of **contributory** protocols (equal member contributions for the group key generation), Becker et al. [1], derived lower bounds for contributory key generation systems for the gossip problem and proved them realistic for Diffie-Hellman (DH) protocols. They used the basic DH distribution extended to groups from the work of Steiner [2]. GDH.2 is the most efficient representative of such schemes: it minimizes the total number of message exchanges. **TGDH** by Kim et al. [10], is a hybrid, efficient protocol that blends binary key trees with 2-party DH key exchanges. Becker in [1], introduced Hypercube that requires a minimum number of rounds, and in [5], Asokan added limited fault-tolerant extensions. Becker introduced Octopus that requires minimum number of messages and then 2^d-Octopus that combined Octopus with Hypercube to a very efficient scheme that works for arbitrary nodes.

Centralized (non-contributory) protocols are based on a simple key distribution center. The most fundamental representative is GKMP [9], in which a group leader shares a secret key with each member and uses it to communicate the group key to the associated member. The original Octopus uses a GKMP version within each subgroup. LKH [8], creates a hierarchy of keys for each group member. Each group member is secretly given one of the keys at the bottom of the hierarchy and can decrypt the keys along its path from to the root. Evolution of the latter are: ELK [21], designed rather for a stationary network, and OFT [7] that minimizes the number of bits broadcast to members after a membership change.

Some more recent proposals exist for wireless ad-hoc networks. Even these schemes, do not seem to scale well or handle successfully the network dynamics. Some of these approaches rely on public key cryptography, which is very expensive for resource constrained nodes, or on threshold cryptography [14, 15, 16, 22], which results in high bandwidth, does not scale well, and presents security vulnerabilities, mainly due to the mobility of nodes. A different approach is based on probabilistic key pre-distribution [17, 18], which is a very lightweight method, designed for sensor networks, but has serious vulnerabilities, and requires some basic infrastructure to handle mobility and membership changes (revocations). Amir et al. [12, 13], focus on robust KA to make GDH protocols fault-tolerant to asynchronous network events. However, their scheme designed for the Internet, and requires an underlying reliable group communication service and ordering of messages, so that preservation of virtual semantics is guaranteed. Poovendran et al. [11], attempt to minimize the bandwidth for secure group communications w.r.t. to energy expenditure. They utilize centralized tree key distribution schemes. The network topology is considered static, and there is no provision for adjusting the key tree structure to a dynamically changing network. The optimal solution of their formulation does not scale with group size.

In [6, 19, 20], Octopus-based KA protocols have been designed to provide robust and efficient KM for group communications in MANETs. The primary focus of this work was the analysis and performance evaluation of the proposed schemes, in isolation from network functions that interact with the protocols (e.g. underlying routing).

III. TGDH Overview and cost Evaluation

The original TGDH is well documented in [2] and [4]. Here, we simply address its key points. Also, a detailed analysis on how the associated costs are derived, wherever it has been omitted in [2], [4], can be found in our TR [6].

Overview: Authors in [10] use 2-party DHKEs to compute a binary tree of keys (height $h=\log_2 n$) from the leaves to the root. Each of the n members is associated with a leaf in the tree. Each tree node x is associated with two cryptographic keys, the un-blinded k_x and the blinded key (BK) $k_x' = g(k_x)$, where g is the 2-party DH function. Interior keys are defined by the rule: $k_x = g(g(k_{left(x)}), g(k_{right(x)}))$. The key associated with the **root** serves as the **group key**. Each member knows only the un-blinded node keys on its path to the root, and the BKs of the siblings of the nodes on the same path (co-path). These BKs are sent by the leader. The authors however, for redundancy reasons, assume that the leader communicates all BKs to all members. The member computes the un-blinded keys along its path to the root. If one of the BKs changes and the member gets the new value, it re-computes the keys on the path and finds the new group key. The new values of the BKs that have changed are broadcast by the sponsor to all members (h broadcasts and $2\times h$ exponentiations are required for the sponsor, and between 1 to h exponentiations for the members, since not all BKs change for them). For the case of the initial tree formation, the total number of messages broadcast by the sponsor is $\sum_{i=1}^{h} n/2^{i-1} \leq 2n$, and the exponentiations are $4n$ for the sponsor and h for the member. In the case of membership updates, the sponsor creates a new secret key for itself. In the addition case, it gets the BK of the new member and updates the corresponding path. The new member gets all n BKs.

IV. Fault-Tolerant Extension of TGDH

We wish to design a distributed, efficient transmission **schedule algorithm** on top of which TGDH can be executed, so that the corresponding communication and routing overhead are reduced. The pre-agreed, ID-based schedule of TGDH members is in fact the key generation algorithm, and does not deal with communication regulation. That would be redundant since all the messaging goes through the sponsor. The BK of each member at a given tree level is unicast to the sponsor who waits to collect all BKs of the same level and then combines them to a single broadcast to all the rest of members. This broadcast message signals the advancement in the tree level. Each member is now able to compute the designated secret value for the next level. It then blinds it and unicasts it to the sponsor. The same process is repeated for all levels upwards the tree until the root is reached. Hence, any two members communicate with each other via the sponsor. With this scheme, KA is executed in a centralized manner.

TGDH may perform as claimed in [10] under the constraints of a limited network where any member is able to reach all group members with 1-hop and where the sponsor acquires extra bandwidth and power capabilities. Such a scenario does not reflect the general MANET case, so the resulting performance metrics are not realistic. The most significant **drawbacks** of TGDH as presented in [10] are the following:

(*a*) Any trusted member, with sufficient bandwidth and power capabilities should be ready to assume the duties of a leader. The most important *constraint* for MANETs is the existence of such node(s). What is more, members may not be able to reach the sponsor via a single transmission anyway. So, both directions may introduce excessive multi-hop routing. (*b*) Simultaneous transmissions of BKs at any tree level to the sponsor through multi-hop routing in a limited network area may increase the probability of collisions at the MAC layer, and thus the probability of re-transmissions, deteriorating even more the performance of the scheme. (*c*) Power is considered valuable resource in MANETs, so it is undesirable for members to serve frequently as relays in heavy KM messages. Also, the sponsor is burdened with heavy tasks that consume its residual energy all too fast. (*d*) The sponsor still remains a single point of failure communication-wise. If it fails during key establishment, the protocol is stalled until a new leader emerges, and a number of BKs must be updated.

TGDH has some **considerable advantages** as well: (*a*) it is simple, no sophisticated scheduling is required, (*b*) failure of a member has no impact on the execution of the protocol other than the update of a logarithmic number of BKs, (*c*) every member knows the BKs of all subgroup members and can proactively anticipate dynamic membership changes that would result in the need to reconfigure the key generation algorithm and restore the group key with minimal latency.

Considering the pros and cons of TGDH, we present a more efficient, distributed version that uses a transmission schedule algorithm to mitigate the weak points of the above approach.

V. DS-TGDH Overview

A. Transmission Schedule for DS-TGDH

The sponsor initially collects through members' registration and link state information, all the required information about its subgroup members. We assume **a generic underlying routing protocol with the property that it always finds the minimum path** (i.e. Dijkstra). It provides all nodes with the paths to at least the nearest subgroup members (w.r.t. the number of hops), and finds at least one path connecting two members, as long as both are within the same cluster. The upper limit in the number of hops between members considered immediate "logical one-hop neighbors" is dynamically set. Routing provides members with information about the **robustness** of the paths to "neighbors". For example, if we know the motion parameters of two virtual neighbors (speed, direction, radio propagation range), and their coordinates, we can determine the duration of the time these two nodes will remain connected, denoted as Link Expiration Time (LET) [25]. The **routing path *r*** is characterized by the minimum among all LETs of its links, denoted as Route Expiration Time (*RET*). RET indicates the overall stability of a path: as soon as a single link on a path is disconnected, the entire path is invalidated, as argued in [25]. We as well choose *RET* as a component of our "path robustness metric", generated and communicated to members.

We select the normalized product of the residual energies of all nodes included in the path: $EN = \frac{1}{|r|}\prod_{i\in r} E_{res}(i)$ as an additional metric to characterize path robustness. In this case we do not consider the minimum residual energy as a valid metric, because a node may participate to more than one routing paths. We rather choose to average the residual energies of nodes along the same routing path and we characterize the "**robustness**" of a routing path r by the value: $\boldsymbol{R_r} = a \times EN + \beta \times RET$.

The contribution of each different parameter towards the computation of $\boldsymbol{R_r}$ can be fine tuned through a and β .

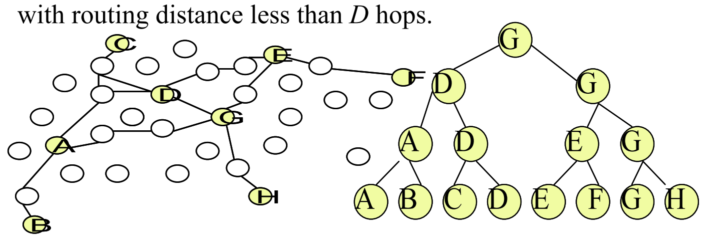
Network nodes are equipped with GPS or other similar devices that allow the computation of their own position, and of distances among them as well. Member x maintains and updates a list $\boldsymbol{L_x}$ with cardinality $|DMx|$, that includes all the "virtual neighbors", and the collected routing metrics for each.

B. Execution Stage Overview

A member j that belongs to the schedule tree $\boldsymbol{T}$, selects one among the available routing paths at level (l-1) that leads to a member $k \in L_j$. Member j generates now the offspring $<j, k>$ for level l. Members j and k are connected with a logical link, one of low cost w.r.t. routing, and are considered siblings at level l. They update all members in their proximity (lists $\boldsymbol{L_j}$, $\boldsymbol{L_k}$) of the new event (TreeFlag(k) is set to "*busy*" mode). Then, they individually proceed to generate offspring for the next level from the lists L_j and L_k. If another member r that did not receive the updates attempts to enlist member j or k in the tree they just "refuse" and r checks its remaining options. The virtual tree expands according to the following simple idea: *The more robust the members, the higher in the tree they will be placed*. A member j arranges L_j w.r.t. the metrics discussed and attempts to use these members in this order one by one as offspring (in successive levels), unless they are already "used" by other members. Whenever a leaf is reached (i.e. a path cannot be expanded anymore), all the information regarding the members that belong to the path traverses up the root. The root then can check if all members are included in the tree (optional). In fact, tree members need only know their immediate parent, grandparent (if any) and the parent's first sibling, in addition to their own siblings. Member j gives priority to this "unselected neighbor" m that satisfies at least one of the following rules in the order they are stated:

Rule#1) $m \in \{L_J \cap L_B / T\}$, where T is the current tree version, and B is the parent (or first sibling) of j. By this rule, we want to ensure that if j becomes faulty then B can take its place in the tree, and become a sibling of all the previous siblings of j, so that the impact of a failure is minimal (no need to prune the tree). To ensure that the selection of m at this stage ensures a relatively strong tree link also, we simultaneously impose that $R_r(j,m) > Th$, where Th is a threshold value. If there is more than one choice, the "shortest path" one is preferred.

Rule#2) $m \in \{L_j / T\}$and $r_{jm} \leq D$, s.t. $\forall x \in \{L_j / T\}$and $r_{jx} \leq D$: $R_r(j, m) > R_r(j, x)$. If rule 1 cannot apply, then rule#2 simply selects this member with the strongest link to j, among those with routing distance less than D hops.



Scheme 1: TGDH schedule formation from arbitrary network graph

Below, we provide the pseudo-code for the offspring selection of member j, to better illustrate the process.

Selection Stage for Member J – Parameters:

Preparation Stage for J: Arrange all members $y \in L_J$ w.r.t. :
a) R_r – in decreasing order and generate list RL_J.
b) $|r|$ – in increasing order and generate list rL_J.

FirstSib(J)=K, Parent(J)=A, Sib(J)=L, Token(J)=1, TreeFlag(*J*)=*1;*

Set $RL_{CMB} = RL_J \cap (L_K \cup L_A)$, in the order of RL_J, and
Set $rL_{CMB} = rL_J \cap (L_K \cup L_A)$, in the order of rL_J.
Set Rev1= LastElem (rL_{CMB}), Rev2=LastElem(RL_{CMB});

Rule#1: search all $y \in rL_{CMB} / RL_{CMB}$ for offspring

```
If ($rL_{CMB} \neq \varnothing$) {
Exit = 0; NoFnd = 0;   y = GetNext ($rL_{CMB}$);
while ( (!Exit ) | | ($y \neq \varnothing$ )) {
If (Status(TreeFlag(y)) == Unknown) {
J ->y: RequestStatus(y);   y -> J: GetTreeFlag(y);   }

if ((TreeFlag (y)) {Update ($L_J$, $RL_{CMB}$, $rL_{CMB}$); // y already used
 y = GetNext ($rL_{CMB}$); if ($y = \varnothing$) NoFnd = 1;   }
// stop here, this is the min. path member fulfilling Rule1
if (!TreeFlag (y)) &&($R_r$(J, y)>Th)) { FoundOpt = y; Exit=1; }
// subopt. $y \in rL_{CMB}$ w/ min. path or max. robust < Th
if ((!TreeFlag (y)) && ($R_r$(J, y)<Th)) {
if ($r_r$(J, y)< $r_r$(J, Rev1)) Rev1=y;  // only min path
if ($R_r$(J, y)> $R_r$(J, Rev2)) Rev2=y; //  max. robust <Th
 y=GetNext ($rL_{CMB}$); if ($y = \varnothing$)   NoFnd = 1;   }  }
```

Rule#2: search all members $y \in rL_J / RL_J$ for offspring

```
If (($rL_{CMB} = \varnothing$ ) | | (NoFnd)) {Exit=0; NoFnd=0;  y=GetNext ($rL_J$);
while ( (!Exit ) | | ($y \neq \varnothing$ ))   {
If (Status (TreeFlag(y)) == Unknown) {
J -> y: RequestStatus(y);   y -> J: GetTreeFlag(y);   }
if ((TreeFlag (y))   {  Update($L_J$, $RL_{CMB}$, $rL_{CMB}$);
 y = GetNext ($rL_J$);    if ($y = \varnothing$ )   NoFnd = 1; }
if (!TreeFlag (y)) {
if (($r_r$(J, y) < $r_r$(J, R1)) && ($R_r$(J, y) > Th)) R1 = y;
if ($R_r$(J, y) > $R_r$(J, R2)) R2 = y; //  max. robustness
 y = GetNext ($rL_J$);    Exit=1; }}
```

```
// Process Results to Decide on Final Siblings
If  Exist (FoundOpt) NewSibl(J)= FoundOpt;   else  {
if  (NoFree($L_J$)) Leaf(J, Status (J, L));  // J stops expanding
else NewSibl(J) = SubOpt_Select(R1,R2,Rev1,Rev2); }  }
```

These rules ensure two basic requirements for robustness and efficiency: *1)* Members with high risk of getting disconnected occupy the fewest possible internal nodes and are pushed towards the leaves. The impact of their "loss" is mitigated as much as possible when pruning the tree, *2)* Members high in the tree are more likely to satisfy rule#1, since they have more available neighbors. Their failure would affect the schedule of a larger members' subset, so it is important to anticipate and remedy such failures with minimum extra cost and latency. At the other extreme, failure of a member associated only with a leaf has no impact to the tree. So, a greedy strategy for the offspring selection appears to be the best to pursue. Members, at all times, attempt to use their best available options (w.r.t. robustness and routing), and keep pushing the "worst candidates" towards the leaves. Also, no member is now a single point of failure. Even if the root A fails, the idea is that its nearest former neighbor, e.g. node B, replaces A in the tree, and routing connects to B all nodes prior connected to A. It is very likely that these nodes remain relatively close to B as well. The other tree nodes remain unaffected.

Applying this algorithm on any network graph, does not necessarily result in a totally balanced transmission tree. This would be desirable if emphasis was placed upon fair resource allocation for building and maintaining a schedule. Our distributed approach indirectly achieves: ***a)*** a minimum delay schedule tree since members are enlisted in the tree in a *first come first served* manner, and ***b)*** producing a relatively balanced tree, since at any level, all members are free to expand, if options are available. So, our approach resembles mainly of breadth-first instead of depth-first search algorithm.

VI. Analysis and Evaluation of DS-TGDH

A. Initial DS-TGDH Schedule Evaluation.

Member j does less than $2L_j$ comparisons of cost $C_{CMP} = O(1)$ each. When j is handed the token, it updates L_j of the status change in its flag (*busy*). While a member y is being tested during the selection process by member x, its flag is set to "*lock*". Those that contact y in the mean time, cannot enlist it as long as its flag remains to "*lock*" mode. After the selection stage ends, x unlocks y's flag by setting it either to: *busy* or *free*. The exchange of ACKs or Status data requires far lower number of bits (K_S) than the key data (K). We obtain the DS-TGDH computation cost C_P, by summing all ramifications of the former algorithm under the worst case scenario. Hence,

$\boldsymbol{C_P} = L_j \times (3C_{CMP} \times K_S)$ (*rule1*) $+ L_j \times (3C_{CMP} \times K_S)$ (*rule2*) $+ (3C_{CMP} \times K_S)$ (*process_results*) $= (2L_j+1) \times (3C_{CMP} \times K_S)$.

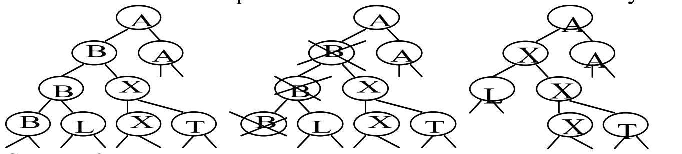
B. Impact of Dynamic Events on the DS - TGDH Schedule.

We want to guarantee a transmission schedule that can anticipate dynamic or membership changes as well. From the security point of view, it is shown in [4, 8] how members' evictions and additions are handled in TGDH to preserve the basic fundamental security properties that escort all secure KM protocols: *Forward*, *Backward* and *Group key secrecy*. Here, we want to show in addition how to resume the TGDH schedule in the event of disruptions of any kind, with the minimum amount of extra overhead and latency, and ensure that the updated schedule is still functional and efficient.

B1. Eviction:

The virtual tree may need reconfiguration after the removal of a member. The idea behind the eviction algorithm is the following: either the parent A or the first sibling X of the evicted member B, substitute B in the tree path it appears. Let B be replaced by A. All former siblings of B must now become A's siblings. For members $x \in (L_B/L_A)$, routing finds the shortest paths to A, and they are all added to L_A. Compared to the rest of B's former siblings, A or X are either more robust or lie closer to B. It is thus quite likely that some of B's former siblings: *a*) already belong to L_A or L_X, and the routing needs not generate extra paths and *b*) do not belong to L_A or L_X, but the paths to be formed are relatively short, since both ends lie in the proximity of the evicted B. These statements are quite likely to hold if the network is relatively dense. Under this simple approach, we reconfigure the tree with little latency and we expect that the new schedule does not result in considerable extra overhead.

The decision of which member will substitute B, is assigned to the **last sibling** chosen by B, denoted as L. L is the least preferred from the point of view of robustness and path length among the remaining available "neighbors" of B placed in the tree. All previous selections are considered more stable. The routing finds the shortest paths from L to both X and A. The shortest one, with robustness is no worse than some threshold, designates which member will substitute B. The reason behind this is that we want the substitute member to accommodate all affected members with low overhead, ensuring as high robustness as possible. It is more likely that the best selection for the least robust sibling is also the best for the rest of them. We could also find the shortest paths to X and A for all siblings, and select the one that accommodates best the majority. The resulting overhead and required coordination, make this solution impractical for the environment of study.



Scheme2: Illustration of schedule maintenance after B's failure, when parent $A = P(B)$ is selected to "replace" B.

Eviction of Member B: - Parameters:
P(B)=A, OFirst(B)=X, OLast(B)=L, Offspring(B)=[X, L];

```
// B's last sibling examines two candidates ( A or X) to substitute B
GetRtPath (($OLast(B)$, $OFirst(B)$));
If ($P(B) \neq \varnothing$ )) GetRtPath (($OLast(B)$, $P(B)$));
If ($P(B) \neq \varnothing$ )) { if (($r_r$ ($OLast(B)$,$P(B)$) < $r_r$ ($OLast(B)$, $OFirst(B)$) && (|$R_r$(($OLast(B)$,$P(B)$) - $R_r$($OLast(B)$, $OFirst(B)$) | < $R$ )) then $P = P(OLast(B)) = P(B)$;
else $P = P(OLast(B)) = $ OFirst($B$); }
if ($P(B)=\varnothing$ )) $P = P(OLast(B)) = $ OFirst($B$);
$\forall y \in L_B$: if ($P(y)= P) \notin L_y$ GetRtPath ($P$); Update($y$);
```

B2. Addition:

The routing finds the shortest paths from the new member T to members in the proximity, and list L_T is created. T will be added as a leaf to this member in L_T to which it is connected via the minimum number of hops and fulfills a robustness criterion at the same time. A modification of *rule1* is used to select the best solutions. Initially, T looks for a member B s.t. B, *Parent*(B)$\in rL_T$. This is necessary to make the tree robust to failures as discussed. If $R_r(T,B)>R$ and $R_r(T,Parent(B))>R$, the solution is optimal, else if only $R_r(T,B)>R$, it is considered suboptimal. If none of the above is true, we apply a version of *rule2* over the members of L_T: T is attached to this member J

with which they share the minimum hop path $r_r(T, J)$, for which robustness is among the highest in L_T. Then, J expands one level and becomes the parent and the first sibling of T.

Add Member T to Subgroup

```
Exit=0; FoundOpt=0; FoundSOpt=0; Set $R_r(T, SOpt)$=0;
Set $Rev1=rL_T$[Last];    $y$ = GetNext ($rL_T$);
$\forall y \in rL_T$ do: while ( (!Exit ) | | ($y \neq \varnothing$ )) {
// Optimal Solution – Rule1
if (($P(y) \in rL_T$ )&&($R_r(T, y)>R$)&&($R_r(T, P(y))>R$))
{ $Opt=y$; FoundOpt=1; Exit=1; }
// SubOpt. Solution – Rule1
if (($P(y) \in rL_T$)&&($R_r(T, y)>R$)&&($R_r(T, P(y))<R$))
{ FoundSOpt=1;  if ($R_r(T, y) > R_r(T, SOpt)$ ) $SOpt= y$; }
//SubOpt.(Rule2)-Select y w/ min. path &threshold robustness
else if (($R_r(T, RL_T[1])$ - $R_r(T, y)$) < Th) {
 if ($r_r(T, y)< r_r(T, Rev1)$)  $Rev1= y$;  } $y$ = GetNext ($rL_T$); }
// Process results:
 if (FoundOpt) { $P[T]=Opt$, FSibl[$T$]=$Opt$; };
else if (FoundSOpt) {$P[T]=SOpt$, FSibl[$T$]= $SOpt$;}
else { $P[T]= Rev1$, FSibl[$T$]= $Rev1$; }
```

C. Analytical Evaluation of Dynamic Events on DS-TGDH.

Eviction/Failure: Let L be the last selected sibling of B. L does only two comparisons to determine which candidate will replace B. Let A be replacing B. Former B's siblings, h_B, that do not already acquire paths to A, use routing to obtain such paths and the associated metric values. However, $h_B<|L_B|$, since a subset of L_B has been prior reserved by other tree members. Also, any $j \notin L_A$ is likely to have been enlisted by B towards the end of its path. Let N_E denote the average number of members that computes such paths. It can be shown that $N_E \approx h_B/2$. The proof is omitted for lack of space.

Addition: About DM_T shortest routing paths are discovered. The total computation cost C_P for T is derived similarly to the eviction case under the worst case scenario. Hence:
$C_P=L_T\times(4C_{CMP}\times K_S)(rule1)+L_T\times(2C_{CMP}\times K_S)(rule2)+(2C_{CMP}\times K_S)$ (*process Results*) $= (3L_T+1)\times(2C_{CMP}\times K_S)$ (bits).
If we impose a limit D on the length of the proximity lists, we obtain the following cost DS-TGDH formulae (per member):

	DS-TGDH Schedule Adj.Costs
Deletion Comm/tion	$N_E\times K_S$ ($> h_B/2$)
Deletion Computation	$2C_{CMP}\times K_S$
Add Comm/tion	$DM_T\times K_S$ ($<D$ paths)
Add Computation	$(3D+1)\times(2C_{COMP}\times K_S)$

VII. DS-TGDH *VS.* TGDH Analytical Evaluation

In TGDH, h rounds are required for the computation of the group key. By then, each member has communicated $h\times K$ bits to the sponsor, and has received a total of $h\times n\times K$ bits, from which the useful information is contained only in $h\times K$ bits. This means that each routing path from the sponsor to any member carries $(n-1)\times K$ bits of "redundant" information per round, which makes the original proactive approach impractical. The total expected number of BKs exchanged is:

$E[BK_S] = (n\times h)$ [members] + $(n^2\times h)$ [sponsor].

In DS-TGDH, the number of pairs that interact in the initial case is reduced to half after each round. This operation is reflected in the way we have constructed the schedule tree. The total number of tree nodes is $2n$. Hence, we have $2\times n$ communication exchanges of key parts over the designated routing paths. Any member must get the BKs of the members of its co-path. The member that participates actively at any step to the key generation knows all the required BKs up to this point. This is true if we consider how the schedule tree is built. If member A is active at step i, it has been active in all previous steps from the start of its path, and has collected and sent all the required BKs (1 per step). Let h_{UA} denote the number of times A appears in the tree ($\neq A$'s path to the root, h_A). During the upward phase, A uses h_{UA} paths and sends a BK over each. It does not send the BK obtained at one round right away to its descendants, but waits first to collect all the keys of the members in its co-path. So, the distribution of the required BKs follows a top-down approach. The number of BKs distributed from a parent A to offspring B is equal to the number of hops B is away from the root. So, A receives a message $(h_A - h_{UA})\times K$ long and sends h_{UA} BKs to its offpring: i.e. for offspring j, at level i, A sends (h_A-i) BKs ($i \leq h_{UA}$).

Members Receive:
BKs (A) = $(h_{UA}-1)_{\text{UPWD}} + (h_A - (h_{UA}-1))_{\text{DNWD}} = h_A$.
Routes Used $\mathbf{RU_R}$(A) = $(h_{UA}-1)_{\text{UPWD}} + 1_{\text{DNWD}} = h_{UA}$.

BKs (*Total*) = $\sum_{i=1}^{n} h_i$, $\mathbf{RU_R}$ **(*Total*)** $= \sum_{i=1}^{n} h_{UI}$.

Assuming $h_{UA} = ½h_A$ in average, the ***expected value of RU*** is:

$$E[RU_R]=E[\sum_{i=1}^{n} h_{UI}]=\sum_{i=1}^{n} E(h_{UI})=½\times\sum_{i=1}^{n} h_i \quad (1).$$

Members Send:
BKs (*A,level i*) = $(h_{UA}-1)_{\text{UPWD}} + (h_A - i)_{\text{DNWD}(i)} = h_{UA} + h_A - 1 - i$.
$\mathbf{RU_S}$**(A) =** $(h_{UA}-1)_{\text{UPWD}}$ (1 BK/route) + $(h_{UA}-1)_{\text{DNWD}}$ (h_A-½ h_{UA}) BKs/route in average).

$\mathbf{RU_S}$**(*total*,twice)**= $(h_{UA}-1)$, **BKs(*total*)** = $(h_{UA}-1+\sum_{i=1}^{h_{UA}-1}(h_A - i))$

$= h_{UA}\times h_A - ½\times h_{UA}\times(h_{UA}-1) = h_{UA}\times(h_A - ½\times(h_{UA}-1))$.

Expected Total Sending Cost values for all Members:

$$\boldsymbol{E[RU_S]}=E[\sum_{i=1}^{n} h_{UI}]=\sum_{i=1}^{n} E(h_{UI}) = ½\sum_{i=1}^{n} h_i \text{ (used twice) } (2).$$

$$\boldsymbol{E[BK_S]}=E(\sum_{i=1}^{n} h_{UI}(h_I - \tfrac{1}{2}(h_{UI}-1)))=E[\tfrac{3}{8}\sum_{i=1}^{n} h_I^2]+$$

$$E[\tfrac{1}{4}\sum_{i=1}^{n} h_I] = \tfrac{3}{8}\sum_{i=1}^{n} E(h_I^{\,2}) + \tfrac{1}{4}\sum_{i=1}^{n} E(h_I) \quad (3).$$

The schedule tree is not necessarily balanced. Any arbitrary **schedule tree** can be derived from a balanced one of the same size, since the following observation is true: to extend a path in the balanced tree from h to $h+1$ (2 nodes are added – property of schedule tree), another path must be abbreviated from h to h-1 (two nodes get eliminated). By induction, we see that for a path to be extended from height h to $h+k$, one or more other paths must be abbreviated by k hops totally. The average individual member participation in unbalanced trees remains the same for both phases. Hence:

$$\sum_{i=1}^{n} h_i = \sum_{i=1}^{n} h = n \times h, \qquad (4),$$

and the previous results of (1), (2), (3) can be revisited.

(1), (2), (4) => $E[RU_R] = \frac{1}{2}(n \times h)$, $E[RU_S] = \frac{1}{2}(n \times h)$.

From (3), (4), with the use of probability theory we obtain:

$$E[BK_S] = \frac{3}{8}\sum_{i=1}^{n} E(h_I^{\,2}) + \frac{1}{4}(n \times h) = \frac{3}{8}\sum_{i=1}^{n} \left((E(h_I))^2 + \sigma_h^2 \right) + \frac{1}{4}(n \times h),$$

where σ_h^2 is the path variance (expected to be low). Hence, $E[BK_S] = \frac{3}{8}(n \times h^2) + \frac{3}{8}(n \times \sigma_h^2) + \frac{1}{4}(n \times h)$.

Total Comm. OH	**TGDH**	**DS –TGDH**
BKs (per member)	h, $n^2 \times h$	$(h - \frac{1}{2}(\frac{1}{2}h - 1)) \times \frac{1}{2} h$
BKeys Total	$(n \times h) + (n^2 \times h)$	$\frac{3}{8} n(h^2 + \sigma_h^2) + \frac{1}{4} n h$
Rt. per member Use /Discover	use 1, n (spons), discovers any	uses $2(h_{UA}-1)$, discovers $2D$
Total Rt. Use/Disc.	use n, discov.$>n$	use $\frac{1}{2}(n \times h)$, discover $2nD$
Redundant Info	$hn(n-1)K$ total	**No**

Table 3: Summary of analytical results of TGDH vs. DS-TGDH.

It is clear that DS-TGDH is substantially more efficient w.r.t. the overall communication, and as will be shown next via simulations, it achieves a much better routing performance.

We have considered a very generic framework to apply and compare the two protocols. Since we assume that not all network nodes belong to the secure subgroup, using broadcast and flooding the network is inefficient. The use of multicast brings about several issues also: (a) the network and members configuration may be such that is not benefited from multicast (i.e. arbitrary relays), (b) underlying multicast should be optimized as well to improve the performance of a key generation scheme, and such a cross-layer consideration is additionally complex, (c) multicast may not be supported by all network nodes that are assumed heterogeneous in general.

VIII. Simulation Results for DS-TGDH, TGDH

i) Simulation Set-Up and Discussion:

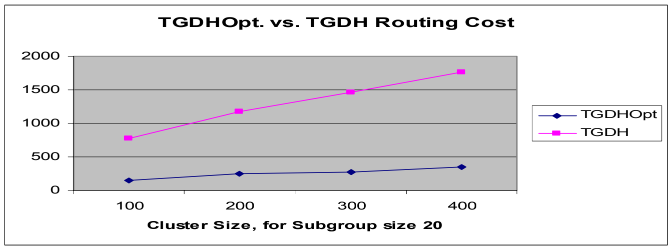
We have conducted a simulation analysis in order to compare the routing cost of DS-TGDH *vs.* TGDH. We use different graphs to generate the secure subgroups and analyze the performance of the two algorithms. Our network graph represents a single cluster area where a single subgroup is deployed. A number of nodes from this graph are randomly selected as subgroup members. We use two methods to select the subgroup leader: either randomly, or pick the member with the largest "member" degree. At the end of the subgroup "registration" period, the sponsor piggybacks the list of the legitimate members into the routing packets. Through the underlying routing, each member obtains the routing path(s) to its closest neighbor(s). We dynamically determine the proximity w.r.t. the number of hops between two members. If no neighbors are found in the proximity, the search diameter (TTL) is gradually expanded until a member is found.

We further assume that while the schedule tree is being formed, the relative placement of members and consequently 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 and probably suboptimal. However, our algorithm is fairly fast. So, it is not too optimistic to assume that the topological changes that occur do not "offset" our solution much from the optimal. Of course, it is expected that the higher the mobility of nodes, the worse the performance of our algorithm is. Even though DS-TGDH is more sensitive to mobility than TGDH, it can still reduce significantly the communication overhead.

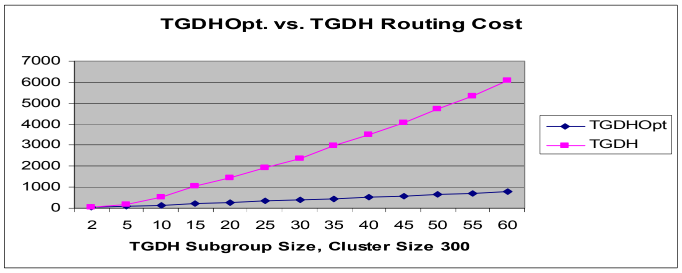
For our evaluation, we generated various random graphs for a given input of the number of nodes n and the number of members m. For the same graph and the same input, we have varied the subgroup configuration, i.e., we have selected the m members in a random manner. For each graph of input $<n, m>$ and subgroup configuration, we have evaluated the total routing cost of DS-TGDH *vs.*TGDH, 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: [100,…500], **Subgroup Size**: [2,…60].

ii) Simulation Results:

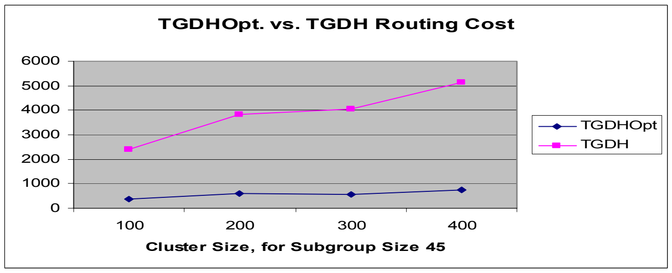
Below we illustrate some indicative results on the routing overhead produced by both DS-TGDH and TGDH, measured in number of hops (relays). The following graphs reflect the routing overhead produced from the key generation in both schemes. The KM messaging is very heavy for the network nodes, so our aim is to reduce the overall number of bits (or packets) required and relieve as many nodes as possible from "relaying" large keying data. Also, we wish to dynamically distribute the KM tasks of a single-point of failure leader to more or potentially all members. Indeed, DS-TGDH results in significant savings in terms of routing overhead, and in most cases the associated ratio is: $\frac{DS\text{-}TGDH(RC)}{TGDH(RC)} < 0.3$. The total savings from the use of DS-TGDH are even more significant if we consider also the amount of redundant keying data, as calculated in our analysis, relayed by nodes during every round of TGDH. To illustrate the above with an example, for a cluster of size 300, and a subgroup of size 35, the averaged relays produced are 457 for DS-TGDH, 2987 for TGDH. This means that the overall redundant data communicated via the relays is: $\frac{2987}{2} \times 34 \times \lceil \log_2 34 \rceil \times 1024$ bits=311,986,170 bits.



Graph1: Total routing overhead computation for original TGDH vs. Decentralized TGDH w.r.t. Cluster Size, for Subgroup Size = 20.



Graph2: Total routing overhead computation for original TGDH vs. Decentralized TGDH w.r.t. SubGroup Size, for Cluster Size = 300.



Graph3: Total routing overhead computation for original TGDH vs. Decentralized TGDH w.r.t. Cluster Size, for Subgroup Size = 45

IX. CONCLUSION

This paper focuses on the design of a decentralized, low cost Fault-Tolerant version of TGDH, denoted as DS-TGDH for a general MANET. DS-TGDH benefits from a cross-layer consideration of the underlying routing and the existing logical TGDH key generation framework to enhance the performance and behavior of the original ancestor scheme. We present a topology aware schedule on top of which DS-TGDH is executed, and we analytically evaluate the overhead of the schedule generation and execution, for the initial and the steady state (under the presence of membership changes and disruptions). Exploring the potential of DS-TGDH through different views (communication, routing, processing overhead), we attempt to provide accurate and spherical evaluation and understanding of both algorithms, and of their strong and weak assets. Through our analytical work and simulation results we show how we can achieve better performance in the environment of interest, with our scheme.

REFERENCES

[1] Klaus Becker, Uta Wille. “Communication Complexity of Group Key Distribution”. Proc.5th ACM Conference on Computer & Comm/tions Security, pages 1-6, San Francisco, CA, November 1998.

[2] Steiner M., Tsudik G., Waidner M. ``Diffie-Hellman Key Distribution Extended to Groups”. 3rd ACM Conference on Computer & Communication Security, ACM Press, 1996.31-37

[3] Maarit Hietalachti. “Key Establishment in Ad-Hoc Networks”. Helsinki University of Technology. Lab. for Theoretical Computer Science.May’01.

[4] A.Perrig. “Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication”. Int’l Workshop on Cryptographic Techniques E-Commerce CryptTEC’99.

[5] N.Asokan and Philip Ginzboorg. “Key-Agreement in Ad-Hoc Networks”. Elsevier Preprint.2000.

[6] M.Striki, J.Baras. “Efficient Scalable Key Agreement Protocols for Secure Multicast Comm/tion in MANETs”. CSHCN TR 2002.

[7] David McGrew. Alan T.Sherman. “Key-Establishment in Large Dynamic Groups Using One-Way Function Trees”. May 1998.

[8] H. Harney, E.Harder. “Logical Tree Hierarchy Protocol”. Internet Draft, Internet Eng. Task Force, April 1999.

[9] H. Harney, C.Muckenhirn. “Group Key Management Protocol (GKMP)”. Specification/Architecture, Int. Eng. Task Force. July 1997.

[10]Y. Kim, A. Perrig, G. Tsudik “Simple & Fault Tolerant Key Agreement for Dynamic Collaborative Groups”.

[11]L.Lazos, R.Poovendran. “Cross-Layer Design for Energy Efficient Secure Multicast Communications in Ad Hoc Networks”. Proceedings of the ICC 2004 Conference, Paris, France, June 2004.

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

[13]Y.Amir, Y.Kim, C.Rotaru, J.Schultz, J.Stanton, G.Tsudik. “Secure Group Communication Using Robust Contributory Key Agreement”. IEEE TPDS, vol. 15, no. 5, pp. 468-480, May 2004.

[14]L.Zhou, Z.Haas. “Securing Adhoc Networks”, IEEE Network Magazine, vol. 13:, no.6, pp. 24-30, Nov/Dec ‘99

[15] J.Kong, P.Zerfos, H.Luo, S.Lu and L.Zhang. “Providing Robust and Ubiquitous Security Support for MANET,” IEEE ICNP 2001, 2001

[16]S.Capkun, L.Buttyan, J.Hubaux,” Self-Organized Public-Key Management for MANETs,” IEEE Transactions on Mobile Computing (TMC) 2002

[17]L.Eschenauer, V.Gligor., “A Key Management Scheme for Distributed Sensor Networks”., ACM CCS’02, pages 41-47, Nov, 2002

[18] H.Chan, A.Perrig, D.Song, “Random Key Predistribution Schemes for sensor networks,” IEEE Symposium on Security & Privacy, CA, May ’03.

[19] M.Striki, J.Baras. “Key Distribution Protocols for Secure Multicast Communication Survivable in MANETs”. Proc’s of the IEEE MILCOM 2003, Boston MA, October 2003.

[20] M.Striki, J.Baras “Fault-Tolerant Hypercube extension for efficient, robust communications in MANETs with Octopus schemes”, CSHCN TR 2005.

[21] A. Perrig, D.Song,. ``ELK, a new protocol for Efficient Large-Group Distribution.” Proc. of IEEE Security and Privacy Symposium, May 2001.

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

[23] A. Hodjat, I.Verbauwhede, “The Energy Cost of Secrets in Ad-Hoc Networks”, IEEE CAS workshop on Wireless Comm/tions & Networking, Pasadena, CA, 2002.

[24] A.Fisiran, R.Lee, “Workload characterization of Elliptic curve Cryptography & other Security algorithms for constrained Environments”, IEEE Int’l Workshop on Workload Characterization, WWC-5, 2002.

[25] S.J.Lee,W.Su,M.Gerla,”Wireless Ad hoc Multicast Routing with Mobility Prediction”, Mobile Networks and Applications 6, pp. 351-360, 2001, Kluwer Academic Publishers.