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

by Maria Striki, John S.Baras

CSHCN TR 2005-9
(ISR TR 2005-97)

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

Abstract—**Securing multicast communications in Mobile Ad Hoc Networks (MANETs) is now 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 group communications. The design of efficient key management (KM) schemes for MANET is of paramount importance, since the performance of the KM functions (e.g. group key generation, entity authentication) imposes an upper limit on the efficiency and scalability of the whole secure group communication system. In this work, we contribute towards efficient, robust and scalable secure group communications for MANETs by extending the TGDH scheme to a novel distributed and topology aware protocol: DS-TGDH. Our aim is to modify TGDH so that: *a*) it is feasible in the most general resource-constrained flat MANET where no nodes with special capabilities may exist, *b*) it produces considerably lower overhead for the network nodes involved, *c*) it handles disruptions with low cost. To meet our objectives we consider in our design the underlying routing protocol, and we apply a distributed version of TGDH over a robust schedule, optimizing parameters of interest. We assume that members have already been authenticated and we focus on the design and analysis of the "reinforced" DS-TGDH. We compare our scheme 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 interest.**

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, visual broadcasts, military command and control grows, the research on security for wireless multicasting becomes increasingly important. The role of KM is to ensure that only valid members have access to a valid group key or can be reached and updated with a valid 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. Nodes within the network may have limited capacity, computational and transmission power. Connections among nodes are temporary (mobility changes, battery drainage, etc.) and unreliable. These constraints render most of the existing KM schemes inefficient in MANETs: among other requirements, they need to catch up with the rapidly changing network topology, and deal with 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. MOT has been designated by far to be the most efficient one, suitable for a resource-constrained network. Hierarchy is supported through the partition of a large 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 is mainly attributed to the application of TGDH within the subgroup. This efficient tree protocol is used in such a way in Octopus schemes, that it boosts their performance even further.

The primary focus of prior work was the analysis and performance evaluation of the proposed schemes, based on the overhead resulting from the processing and exchange of KM information only, in isolation from underlying network functions, i.e. routing, clustering, leader election, that constitute the backbone or simply support their correct operation. Although some of the protocols compared appear promising (MOT), their evaluation under these assumptions is not totally "fair" because each scheme considered, relies upon the former functions more or less, or in a different way. So, the consideration into the design and analysis of the KA schemes discussed, of at least those network parameters that would further point out their individual characteristics and would accentuate the differences in their performance, might shed more insight on their actual feasibility for MANETs, and provide more realistic results.

For example, all three protocols operate on a group that is already divided into d subgroups. A clustering mechanism maintains the above hierarchical framework, on top of which the schemes are executed. Although the cost of adding and maintaining the necessary hierarchical 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 subgroup leader election process and the backbone transmission schedule within the subgroup differ. For example, failure of a centralized heavy duty single point of failure leader has considerable impact on the subgroup state, and the overhead required for the subgroup to resume after the election of a new leader is substantial as well. (O) is based on a centralized subgroup leader to execute a subgroup key distribution scheme, MOT and MO use a centralized leader that has a rather coordinating role to facilitate the execution of subgroup KA. Contrary to MO and MOT, scheduling is not needed for the subgroup in (O). In MO and MOT however, members interact among themselves for the generation of the subgroup key. Ideally, any member could transmit its contribution to the rest, and then process only the parts dictated by the key generation algorithm. This approach yields O(n^2) communication complexity. A simple schedule based arbitrarily on members' *ID*s is likely to result in unnecessary routing. Our main objective is lowering the overall communication overhead in the frameworks we design. Thus, the need to apply GDH.2 and TGDH protocols over a transmission schedule that optimizes desired metrics of interest comes into the play. The authors of TGDH make so simplistic assumptions about the network, that the need to integrate the discussed functions in their design is mitigated. Their assumptions however (e.g. all members are reached from the leader via a single broadcast) do not correspond to a general resource constrained MANET. Obviously, a cross-layer consideration is needed for the design of a concrete KM framework, in a real network. Therefore, to accurately evaluate the three Octopus-based protocols in a way that makes sense, the former issues cannot be ignored. Previous results designated MOT as the most efficient overall for the environment discussed. It would be worth exploring if MOT still prevails and in which cases, after a re-evaluation of the schemes, under more realistic network assumptions.

In the present work, we assume that any arbitrarily selected network node could potentially belong to the secure group. Thus, a path between members may as well include non-member relays. All group members are assumed to be honest. No assumptions are made about secure routing. We simply rely on the redundancy of the routing protocol to ensure that the exchanged messages are delivered in a timely manner. Dividing such a group into subgroups, so that each subgroup corresponds to a fully connected graph of members only, that could be reached by the elected subgroup leader with a single broadcast, would permit the execution of the original "centralized" TGDH exactly under the assumptions described in [4]. This is an impractical approach for a large number of group members: a much higher number of subgroups will be created, very sensitive to even subtle mobility changes, and it is quite likely that such subgroups may contain very few members, even a single one. Even if such subgroups are in very close proximity with each other, 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 very high and unstable. If *d* grows too high, then the resulting inter-cluster signaling overhead outweighs the benefits of Hypercube for the inter-cluster communication in Octopus schemes.

Working towards the direction discussed under the above assumptions, we present in this paper an adaptation of TGDH to the requirements of our Octopus based framework, in a general dynamic infrastructure-less network. In particular, we modify TGDH so that: *a*) it is made distributed, and no single point of failure leader is required for the protocol's upright operation, *b*) it is executed under a schedule that optimizes our own defined "routing" and "robustness" metrics, under this new cross-layer, topologically aware consideration, *c*) it tolerates failures and disruptions with low cost, *d*) it is much more efficient w.r.t. the resulting communication 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 the issues of fault-tolerance for our framework. In section 5 we introduce and describe the DS-TGDH scheme, in section 6 we analyze the DS-TGDH for the initial and steady state operations. Section 7 presents the analytical comparative performance evaluation, and section 8 the simulation results for DS-TGDH *vs.* TGDH. Finally, in section 9 we conclude the paper.

II. Related Work

There exist several KM proposals for secure group communications in the literature. From the perspective of **contributory** protocols (equal member contributions required 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]. From this work, GDH.2 is the most efficient representative of group DH 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 the minimum number of rounds. In [5], Asokan et al. added limited fault-tolerant extensions to Hypercube. Becker introduced Octopus as one that requires minimum number of total 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 secret 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 the leaf 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. The number of keys broadcast to the group in this case, and the computational efforts of the members are logarithmic in the number of members.

There exist some more recent proposals 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 communication overhead, 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 security vulnerabilities, and requires some basic infrastructure to handle mobility and membership changes (revocations). Y.Amir et al. [12, 13], focus on robust KA, and attempt to make GDH protocols fault-tolerant to asynchronous network events. However, their scheme is 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 communication overhead 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. It is shown that 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 [6].

Overview: Authors in [10] use 2-party DHKEs to compute a binary tree of keys from the leaves to the root. Each tree node x is associated with two cryptographic keys, the un-blinded key k_x and the blinded key (BK) $k_x' = g(k_x)$, where g is the 2-party DH one-way function. Each member is associated with a leaf in the tree. Interior keys are defined by the rule: $k_x = g(g(k_{left(x)}), g(k_{right(x)}))$. The **key** associated with the **root** of the tree serves as the **group key**. For the correct operation of this protocol it suffices that each member knows only the un-blinded node keys on the path from its leaf 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 to all members all group BKs. 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 leader to all members. A broadcast of h keys is required to update all members of the BKs that have changed.

Any trusted, robust member, with sufficient computational, storage, and communication capabilities (if there exists any) should be ready to become "sponsor" and assume the duties of a leader. The most important *constraint* for MANETs is the existence of robust nodes, able to become leaders. Thus, such protocols operate as intended under the assumption that the network size is limited and all nodes are able to reach all group members within 1-hop. The latter constraint can be relaxed if there exist a path of group members only, leading to any member, at the expense of extra communication cost however. Such considerations are not made for TGDH execution, and the routing and communication costs invoked by this scheme do not reflect real MANET scenarios.

Initial Member/Sponsor Computation/Communication: At every step, a member gets the BK of its sibling, raises it to the power of its own secret key and blinds the new key (2 exp/s, 1 broadcast). The sponsor associated with an internal node broadcasts the BK to all members, computes the secret key of the parent node and blinds it. Therefore, a single operation on one node corresponds to 1 broadcast and 2 exp/s. The total number of broadcast messages is: $\sum_{i=1}^{\log_2 n=h} n/2^{i-1} \leq 2n$. The total number of exp/s is approximately $4n$. Each sponsor/member does $\log_2 n = h$ broadcasts, $2 \times h$ exp/s at max.

Add/Delete Sponsor Computation/Communication: In both cases the sponsor generates a new secret key for itself. In the addition case, it gets the BK of the new member. It computes the secret and BK of the parent, generates a new secret key, etc. All updated keys in the sponsor's path from the leaf to the root are calculated similarly. So, a sponsor does $2 \times h$ exp/s, and sends to all members the updated h BKs. The new member gets all n Bs, and does h exp/s in the addition case.

Add/Delete Member Computation: The new member does h exp/s (using the BKs of its co-path) to get the group key. One to (h-1) exp/s are done by the rest of members to compute the group key, since not all BKs change for them: i.e. $n/2$ members do only 1 exp.,…, $n/2^h$ do h exp/s. In average, a member does: $\frac{1}{n}\sum_{i=1}^{\log_2 n=h} \frac{n}{2^i} \times i \approx 2$ exp/s.

IV. FAULT-TOLERANT EXTENSION OF TGDH

Our **objective** is to generate a **distributed, efficient transmission schedule algorithm** on top of which TGDH can be executed, so that communication and routing overheads are lowered. The pre-agreed, ID-based schedule, TGDH members are provided is in fact the key generation algorithm, and does not have to do with communication regulation. That would be redundant since all members receive all BKs from the sponsor anyway. In fact, all the key generation messaging goes through the sponsor. The BKs of each member at a given tree level are unicast to the sponsor. The sponsor waits to collect all these values and then broadcasts them all to all members. Each member gets the broadcast message and selects only those values required at that level for computing the final group key. 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 virtual tree until the root is reached. Therefore, any two members communicate with each other via the sponsor. With this scheme, KA is executed in a centralized manner. This approach appears suitable for a restricted wire-line network, where the sponsor must acquire extra bandwidth and power capabilities, since all communication for group key generation must go through it. Its most significant **drawbacks** are the following:

(*a*) This approach is suitable only for a limited network size, where the sponsor is 1-hop away from all members. As we discussed, it is hard to find nodes that acquire the appropriate bandwidth to reach all members directly. 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 and undesirable relay nodes. (*b*) The simultaneous transmissions of the BKs at any tree level to the sponsor through multi-hop routing in a relatively restricted 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 a valuable resource in MANETs for all nodes, and it is undesirable for them to serve frequently as relays in heavy KM messages. In addition, the sponsor is burdened with heavy tasks that consume its residual energy all too fast, which is also undesirable. (*d*) The sponsor remains still a single point of failure communication-wise at least. Upon failure of the current sponsor during key establishment, the protocol is stalled until a new leader emerges, all the communication exchanges of that level must be repeated, and a number of BKs must be updated by the newly elected subgroup leader.

TGDH has some **considerable advantages** as well: (*a*) it is simple, and 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 certain dynamic member changes that would result in the need to reconfigure the key generation algorithm (use BKs of different members than before) 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

A1. *Preparation Stage*

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). The routing 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. So, it provides each member with paths and information about at least its "logical one-hop" neighbors. The upper limit in the number of hops between members considered immediate "neighbors" is dynamically set. The higher this limit, the higher the degree *DM* of each member is likely to be.

The routing provides members with information about the robustness of the paths to "neighbor" members. 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, D_t, denoted also as the Link Expiration Time (LET) [25]. The routing path *r* is characterized by the minimum LET among all LETs of the links of the path, i.e. MIN_LET= $\min_{e_{ij}\in r} LET_{ij}$, denoted as Route Expiration Time (*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. An averaging of the residual energies of nodes along the same routing path seems to be a more valid metric. So, we characterize the "robustness" of a routing path *r* by the value: $\boldsymbol{R_r} = a \times EN + \beta \times RET.$ (1).

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

It is assumed that network nodes are equipped with GPS or other similar devices (indoor or outdoor), which allow the computation of their own position, permitting consequently the computation of distances among nodes as well, etc. These values are updated at regular intervals and are communicated to subgroup members via the underlying routing protocol.

Member *x* maintains and updates a list, denoted as $\boldsymbol{L_x}$ with cardinality $|DMx|$, that includes all the "virtual neighbors", and the collected routing metrics for each.

```
Preparation Stage for Subgroup Member 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 $rL_J$.
Set: TreeFlag(J)=0;  Token(J)=0;
```

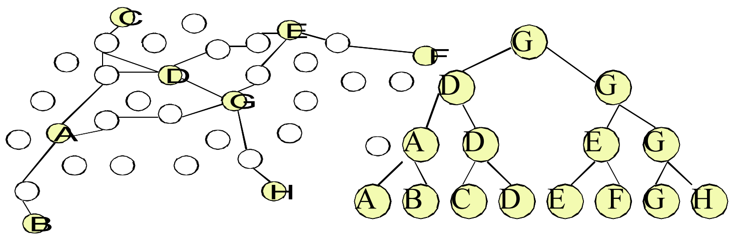
A2. *Execution Stage Overview*

Each member *j* in this tree ***T***, selects one among the available routing paths leading to a member in list L_j, denoted as *k*. Member *j* generates now the following offspring for the next level: <*j*, *k*>. Therefore, 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* in the virtual tree. 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 member *r* checks its remaining options.

The virtual tree is expanded 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 one by one as offspring (in successive tree levels) in this order, unless they are already "used" in the tree 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 it up to the root. This way the root verifies that all members are included in the tree (optional step). In fact, tree members need only know their immediate parent, grandparent (if they exist) 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:

Rule1) $m \in \{L_J \cap L_B / T\}$, where T is the current tree version, and member 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, (m is one of those siblings), so that the impact of a failure is minimal (no need to prune the tree after all). To ensure that the selection of m at this stage ensures a relatively strong tree link also, we simultaneously impose that $R_r(j,m) > Thr\_R$, where Thr_R is a threshold value. If there is more than one choice, the "shortest path" one is preferred.

Rule2) $m \in \{L_j / T\}$ s.t. $\forall x \in \{L_j / T\}$: $R_r(j, m) > R_r(j, x)$, (i.e. max. $R_r(j, m)$), and $r_{jm} \leq D$.



Scheme 1: Graphic illustration of TGDH Tree Schedule generation from an 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: *FirstSibl(J)=K; Parent(J)=A; CurrentSibl(J)=L; Token(J)=1;* TreeFlag(*J*)*=1;* *J->L$_J$*: TreeFlag(*J*); *A->J*: L_A ; *K->J*: L_K ; // *J -> K, A*: L_J ;

```
Set $RL_{COMB} = RL_J \cap (L_K \cup L_A)$, in the order of $RL_J$, and
Set $rL_{COMB} = rL_J \cap (L_K \cup L_A)$, in the order of $rL_J$.
Set Rev1= LastElem ($rL_{COMB}$);  Set R1= LastElem ($rL_j$);
Set Rev2=LastElem($RL_{COMB}$);   Set R2= LastElem ($RL_j$);

//Rule_1: search all $y \in rL_{COMB} / RL_{COMB}$ for offspring
If ($rL_{COMB} \neq \varnothing$ ) {
  Exit = 0; Notfound = 0;   $y$ = GetNext ($rL_{COMB}$);
  while ( (!Exit ) | | ($y \neq \varnothing$ )) {
    If (Status(TreeFlag($y$)) == Unknown) {
      $J$->$y$: RequestStatus($y$);  $y$ -> $J$: GetTreeFlag($y$);   }
    if ((TreeFlag ($y$)) // $y$ already used in the tree
    { Update ($L_J$, $RL_{COMB}$, $rL_{COMB}$);   $y$ = GetNext ($rL_{COMB}$);
      if ($y = \varnothing$ ) Notfound = 1;  }
    // stop here, this is the min. path member fulfilling Rule1
    if (!TreeFlag ($y$)) && ($R_r(J, y)$>Thr_$R$)) {
      FoundOpt = $y$; Exit=1; }

    // subopt. $y \in rL_{COMB}$ w/ min. path or max. robust < Thr
    if ((!TreeFlag ($y$)) && ($R_r(J, y)$<Thr_$R$)) {
      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 <Thr
      $y$=GetNext ($rL_{COMB}$); if ($y = \varnothing$ )   Notfound = 1;  }  }
// Rule#2: search all members $y \in rL_J / RL_J$ for offspring
if (($rL_{COMB} = \varnothing$ ) | | (NotFound)) {
  Exit = 0;   NotFound = 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($L_J$, $RL_{COMB}$, $rL_{COMB}$);
      $y$ = GetNext ($rL_J$);   if ($y = \varnothing$ )  Notfound = 1; }

    if (!TreeFlag ($y$)) {
      if (($r_r(J, y) < r_r(J, R1)$) && ($R_r(J, y) > Thr\_R$)) 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 no sibling found, then J stops expanding (leaf)
  If  (NoFree($L_J$)) Leaf($J$, Status ($J$, $L$));
  // If suboptimal solutions – decide which to use
  If (Free($L_J$)) {
    if (Rev1 - R1>$D$) $y\_mdist$ = R1; else $y\_mdist$ = Rev1;
    if (R2 - Rev2 >$R$)  $y\_rob$ =R2; else $y\_rob$ = Rev2;
    NewSibl($J$)=$F_{SELECT}$ ($y\_mdist$, $y\_rob$);   }  }  }

Expansion Stage - Member J – Sibling (J, L), Offspring (J, F)
$J$ ->$L_J$:  NewSibl($J$)=$F$, TreeFlag($F$)=1; $J$->$F$: Token($F$)=1;
//Optional: Leaf J updates sponsor of all obtained siblings
 If Leaf($J$, Status ($J$, $F$))   $J$ -> Root:  Update(SibList($J$));
```

These rules ensure two basic requirements for robustness and efficiency: 1) The most isolated members (with the highest risk of getting disconnected), are pushed towards the tree leaves, occupying the fewest possible internal nodes. This way, the impact of their "loss" is mitigated as much as possible when pruning the tree. Also, the higher a member is placed in the tree, the more densely connected it is to the subgroup, 2) Members high in the tree are more likely to satisfy rule (a), since the number of available 1-hop neighbors is higher. Failure of such members, affects the transmission schedule of a larger members subset, and 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 at all to the current schedule. So, a "greedy algorithm" for the offspring selection appears to be the best strategy to pursue. Members, at all times, attempt to use their "best available choices" (w.r.t. robustness and routing cost), and keep pushing the "worst candidates" towards the leaves, which is exactly what we want to achieve. Also, no member is now a single point of failure. Even if the root A fails, the idea is that its nearest former tree neighbor, e.g. node B, replaces A in the tree, and routing connects with *B* all nodes prior connected to *A*, and it is very likely that these nodes are relatively close to *B* as well, in terms of routing. 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 members to build and maintain 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)*** limiting the height of the tree, or producing a relatively balanced tree, since at any level, all members are free to expand, provided that they have choices available, so that the resulting approach resembles mainly of breadth-first search instead of depth first search algorithm.

VI. Analysis and Evaluation of DS-TGDH

A. *Initial DS-TGDH Schedule Evaluation.*

Each member *j* is required to do less than $2\times L_j$ comparisons. When *j* is handed the token, it must update all the members in L_j of the status change in its flag (*busy*). A member first ensures that its candidate has not already been selected in the tree. 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*. Each member must contact $2\times L_j$ members at max., exchanging ACKs and Flag Status data that require far lower number of bits (K_S) than these of the key data (*K*). The processing cost for these operations is practically negligible and set to: $L_j\times(3\times C_{COMP}\times K_S)$(*rule_1*)+ $L_j\times(3\times C_{COMP}\times K_S)$ (*rule_2*) + $(3\times C_{COMP}\times K_S)$(*process_results*)= $(2\times L_j+1)\times(3\times C_{COMP}\times K_S)$, where C_{COMP} = O(1). Hence, the total computation cost becomes: $(2\times L_j+1)\times(3\times C_{COMP}\times K_S)$ (bits). The storage cost for a member is upper limited by $7\times L_j\times K_S$ (bits). If we impose a limit ***D*** on the length of the proximity lists, we obtain the following cost DS-TGDH formulae (per member):

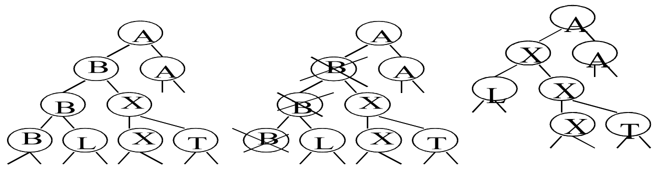
Per Tree Member Costs	**TGDH Schedule Init Costs**
Communication	$2\times D\times K_S$
Computation	$(2\times D+1)\times(3\times C_{COMP}\times K_S)$
Storage	$7\times D\times K_S$

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

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

A. Eviction:

The removal of a member results in the need to reconfigure the virtual tree. The idea behind the eviction algorithm is the following: The parent *A* or the first sibling *X* of the evicted member *B*, substitute *B* in the path that it occupies in the tree. Assume that *B* is replaced by *A*. All members that have been included in the tree as *B*'s siblings must now become *A*'s siblings. For members *x: s.t.* $x\in(L_B/L_A)$, the routing finds the shortest paths *A*, so that they are all added in L_A. Compared to the rest of *B*'s former siblings, *A* or *X* are likely to be the most robust or the "closest" ones to *B*. In any case, those two members have been selected in the tree prior to the other siblings, and they are considered better candidates. So, in the general case it is quite likely that some of these siblings: *a*) already belong to L_A or L_X, and there is not need for the routing to generate paths for them leading to either, and *b*) do not belong to L_A or L_X, but the paths to be generated will be 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. Thus, by substituting *B* with *A* or *X*, we reconfigure the tree in a simple way, with little latency and we expect that the new schedule does not result in considerable extra overhead, compared to the original one. The decision about which among the two members, *A* or *X*, 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, provided that its resulting robustness is no worse than some threshold, designates which member will substitute *B*. The reasoning behind this is that we want the member that replaces *B* to accommodate all affected members with low overhead, ensuring as high robustness as possible. It is more likely that



Scheme2: Illustration of the schedule maintenance after the failure of member *B*, when parent *A* is selected to "replace" *B*.

the best selection for the least robust (and probably most remote) 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, latency and required member coordination, make this solution impractical and preventive for an environment with all the limitations discussed.

Eviction of Member B: - Parameters:

```
Parent(B)=A, OFirst(B)=X, OLast(B)=L, Offspring(B)=[X, L];
// The last chosen sibling of B compares the two different paths and decides if A or X substitutes B
GetRtPath ((OLast(B), OFirst(B));
If (Parent(B) ≠ ∅ )) GetRtPath ((OLast(B), Parent(B));
If (Parent($B$) $\neq \emptyset$ )) {
If (($r_r$ ($OLast(B)$,$Parent(B)$) < $r_r$ ($OLast(B)$, $OFirst(B)$) && ($|R_r((OLast(B), Parent(B)) - R_r(OLast(B), OFirst(B))| < R$ ))
then $P$ = Parent($OLast(B)$) = Parent($B$);
else $P$ = Parent($OLast(B)$) = OFirst($B$); }
If (Parent($B$)=$\emptyset$ )) $P$ = Parent($OLast(B)$) = OFirst($B$);

$\forall y \in L_B$: if ($Parent(y)= P) \notin L_y$ GetRtPath ($P$); Update($y$);
```

B. Addition:

The routing finds the shortest paths from the new member T to members in the proximity, and the list L_T is generated. Member 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. As a first step, T looks for a member B so that both B, $Parent(B) \in rL_T$. This criterion is needed to render the tree robust to members' failures, as discussed. If $R_r(T,B)>R$ and $R_r(T,Parent(B))>R$, we consider the solution optimal, otherwise if only $R_r(T,B)>R$, we consider it 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, i.e. $((R_r(T, RL_T[1])-R_r(T, J))$ <Thr_X). Then, J is "expanded" one level and becomes the parent and first sibling of T.

Add Member T to Subgroup – Version I

```
$T$: Obtain $L_T$ from the routing protocol.
Apply "Preparation Stage" for Subgroup Member $T$

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 ) | | ($y \neq \emptyset$ )) {

// Optimal Solution – Rule1
if (($Parent(y) \in rL_T$ )&&($R_r(T, y)>R$)&&($R_r(T, Parent(y))>R$))
{ $Opt=y$; FoundOpt=1; Exit=1; }

// SubOpt. Solution – Rule1
if (($Parent(y) \in rL_T$)&&($R_r(T, y)>R$)&&($R_r(T, Parent(y))<R$))
{ FoundSOpt=1; if ($R_r(T, y) > R_r(T, SOpt)$ ) $SOpt= y$; }

//SubOpt.(Rule2)-Select y w/ min. path &threshold robustness
else if (($R_r(T, RL_T[1]) - R_r(T, y)$) < Thr_$X$ ) {
 if ($r_r(T, y)< r_r(T, Rev1)$) $Rev1= y$; } $y$ = GetNext ($rL_T$); }

// Process results
if (FoundOpt) { Set: $Parent[T]=Opt$, FSibl[$T$]=$Opt$; }; else
if (FoundSOpt) {Set: $Parent[T]=SOpt$, FSibl[$T$]= $SOpt$;}
else { Set: $Parent[T]= Rev1$, FSibl[$T$]= $Rev1$; }
```

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

A.Eviction/Failure:

Let the last chosen sibling of B be member L. L needs only do two comparisons to determine which candidate will replace the evicted member. The processing cost for L is: $(2\times C_{COMP}\times K_S)$ (bits). Let A be replacing B. Those among the 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, $h_B<D$, since a subset of members in L_B have been prior reserved by other tree members as discussed. Furthermore, according to the eviction rules, members $j \notin L_A$ are likely to have been enlisted by B towards the end of its path. Let N_E denote the avg. member number that will compute such paths. Here, we make the following observations: as we move from level i to i+1, the number of members at level i that attempt to expand at level i+1 doubles, affecting directly or indirectly the probability of j finding an offspring k, s.t. $k \in L_A \cap L_B$. For a relatively balanced tree, there exists a mirror sub-tree of A, whose descendants belong to L_A as well (from construction or selection with probability 1 or lower respectively). The previous sibling in the path of B, reduces the availability in L_A as well. We further assume that all the rest of members at level i reduce the availability in L_A, though less directly. Considering all the above, we can roughly assume that in avg. $N_E \approx h_B/2$.

B.Addition

About DM_T shortest routing paths are discovered. Member T does: $L_T\times(4\times C_{COMP}\times K_S)$(bits) (*rule1 optimal* + *suboptimal*) + $L_T\times$ $(2\times C_{COMP}\times K_S)$ (bits) (*rule2*) + $(2\times C_{COMP}\times K_S)$ (bits) (*Process Results*) = $(3\times L_T+1)\times(2\times C_{COMP}\times K_S)$ (bits) which is the worst case scenario. Hence, the total computation cost is approximately: $(3\times L_j+1)\times$ $(2\times C_{COMP}\times K_S)$ (bits).

	TGDH Sched. Adj.Costs
Deletion Comm/tion	$N_E\times K_S$ (approx.> h_B /2)
Deletion Computation	$(2\times C_{COMP}\times K_S)$
Add Comm/tion	$DM_T\times K_S$ (<D paths, 1 member)
Add Computation	$(3\times D+1)\times$ $(2\times C_{COMP}\times K_S)$
Storage	$7\times D\times K_S$

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

In TGDH, h rounds between member-sponsor are required for each member to compute the group key, during which each member communicates its contribution (K bits) to the sponsor. The sponsor waits to collect all contributions and then multicasts a combined message including all contributions ($n\times K$ bits) to all members. At the end of the h rounds, each member will have communicated to the sponsor $h\times K$ bits, the sponsor will have broadcast to the members $h\times n\times K$ bits, and each member will have received a total of $h\times n\times K$ bits, from which the useful information is contained only in $h\times K$ bits. The latter 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.

In DS-TGDH, the number of pairs of members that interact in the initial case is reduced to half after each round. This operation is exactly reflected in the way we have constructed the schedule tree. The total number of the resulting tree nodes is $2\times n$, even if the tree is unbalanced. Hence, we have $2\times n$ communication exchanges of key parts over the designated routing paths. The group key is constructed upwards the tree (leaves to root). Each member needs to know the BKs of the members of its co-path. The member that participates actively at any step to the key generation knows all the appropriate BKs up to this point. This is true if we consider how the schedule tree is built. A member, e.g. A, that is active at step i, has been active during all previous steps from the start of its path, and has collected and sent all the required BKs (one during each step). Let h_{A_up} denote number of times A appears in the tree or the path of A for the upward phase – different from the path of A to the root (h_A). During the upward phase, A uses h_{A_up} 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 required to compute the group key, i.e. those of the members in its co-path.

After the group key is generated at the root, the distribution of the missing key parts (BKs) to the appropriate members follows an up-down approach. Starting from the root, each parent sends to its offspring only the required (missing) BKs of the siblings of all the ancestor members it has obtained. As soon as an internal member obtains all the required BKs and computes the group key, it sends to its own offspring the designated BKs. The number of BKs to be distributed in the top-down approach from a parent A to offspring B is equal to the number of hops B is away from the root. Therefore, A receives one message $(h_A - h_{A\_up})\times K$ long. It sends h_{A_up} BKs to its descendants: for offspring j, at level i, it sends $(h_A - i)$ BKs, or a message of length $(h_A - i)\times K$ bits, where $i \leq h_{A\_up}$.

Member A receives:
Upward Phase: $(h_{A\_up}-1)$ BKs, $(h_{A\_up}-1)$ routes.
Downward Phase: $(h_A - (h_{A\_up}-1))$ BKs, singe route.
Total for Member *A*: h_A BKs, h_{A_up} routes.

Total for all Members: $\sum_{i=1}^{n} h_i$ BKs, $\sum_{i=1}^{n} h_{i\_up}$ routes.

Assuming $h_{A\_up} = \frac{1}{2}\times h_A$, in avg., the expected number of routes is: $E[R_R] = E[\sum_{i=1}^{n} h_{i\_up}] = \sum_{i=1}^{n} E(h_{i\_up}) = \frac{1}{2}\times \sum_{i=1}^{n} h_i$ (5).

Member A sends:
Upward Phase:$(h_{A_up}-1)$BKs, $(h_{A\_up}-1)$ routes (1 BK /route).
Downward Phase: $(h_A - i)$ BKs to offspring at level i, $(h_{A\_up}-1)$ routes (avg. $h_A - \frac{1}{2}(h_{A\_up}-1)$) BKs per route).

Total for Member *A*: $(h_{A\_up}-1+\sum_{i=1}^{h_{A-up}-1}(h_A - i)) = h_{A\_up}\times h_A - \frac{1}{2}\times h_{A\_up}\times(h_{A\_up}-1) = h_{A\_up}\times(h_A - \frac{1}{2}\times(h_{A\_up}-1))$ BKs, $(h_{A\_up}-1)$ routes, each traversed twice.
Total for all Members: The expected number of routes is:

$E[R_S] = E[\sum_{i=1}^{n} h_{i\_up}] = \sum_{i=1}^{n} E(h_{i\_up}) = \frac{1}{2}\times \sum_{i=1}^{n} h_i$, used twice.

The BKs sent are: $E[BK_S] = E(\sum_{i=1}^{n} h_{i\_u}\times(h_I - \frac{1}{2}(h_{i\_u}-1)))=$

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

The schedule tree is not necessarily balanced. In both balanced and unbalanced trees (binary, n leaves), there exist n paths leading to the root. Any arbitrary schedule tree can be derived from a balanced tree with the same number of internal and external nodes, since the following observation is true: for each path (of the balanced tree) that is extended from height h to $h+1$ (always two nodes are added in the path because of the way that the schedule tree is constructed), another path must be abbreviated from h to h-1 (always two nodes are going to be eliminated as well). By induction, we can also 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 in total. Thus, even though in unbalanced trees the participation of individual members to both phases is not necessarily the same, the avg. participation remains the same. So, assuming that $\sum_{i=1}^{n} h_i = \sum_{i=1}^{n} h = n\times h$, the previous results can be revisited.

So, $E[R_S] = \frac{1}{4}(n\times h)$. With the use of probability theory:

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

where σ_h^2 is the variance of the tree paths, expected to be low.

Hence, $E[BK_S] = \frac{3}{8}(n\times h^2) + \frac{3}{8}(n\times \sigma_h^2) + \frac{1}{4}(n\times h)$.

In TGDH, the total number of BKs communicated, given the same assumptions about the network, the routing, and the use of unicast vs. multicast etc., is provided below:

$E[BK_S]_{\mathrm{TGDH}} = (n\times h)$ [members] + $(n^2\times h)$ [sponsor] .

It can be seen that our protocol is substantially more efficient w.r.t. the overall key messaging communication, and as we are going to show next via simulations, it achieves a much better routing performance (measured w.r.t. number of hops).

We have considered a very generic framework and specifications 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. Also, the use of multicast brings about several issues: the network and subgroup members configuration may be such that is not benefited from multicast (arbitrary relays assumed), underlying multicast should be optimized as well in order to improve the performance of the key generation scheme, and such a cross-layer consideration is additionally complex, and also, multicast may not be supported by all network nodes that are assumed heterogeneous in general.

Scheduling + KM Comm/tion OH	**Original TGDH (no schedule)**	**Distr., Schedule (DS) - TGDH**
BKs (per member /sponsor)	h (member), $n^2\times h$ (sponsor)	$(h-\frac{1}{2}(\frac{1}{2}h-1))\times$ $\frac{1}{2}h$, in avg.
BKeys Total	$(n\times h) + (n^2\times h)$	$\frac{3}{8}n\times(h^2+\sigma_h^2)+$ $\frac{1}{4}(n\times h)$
Routes Discover/ Use, per member	**Member** discover any, use 1 (nK bit) **Sponsor** use n (nK bits each)	$(h_{A\_up}-1)$ (1 BK) + $(h_{A\_up}-1)$ (avg.$h_A - \frac{1}{2}(h_{Aup}-1)$ BKs, $2D$ (sched.)
Routes Discover/ Use, Total	Disc.>n, use n, $((n\times h)+(n^2\times h))K$ bits totally	$(h_{A\_up}-1)$ routes$\times 2$, $2nD$ (schd. Route) Disc.,K_S bit mssg)
Redundant Info	$h(n-1)K$ / member, $hn(n-1)K$ in total	**No**

Table 3: Brief Summary of analytical results w.r.t. KM application of Original TGDH vs. DS-TGDH.

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 for selecting the subgroup leader in this simulation: either randomly, or the subgroup 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) of the routing is gradually expanded until a member is found.

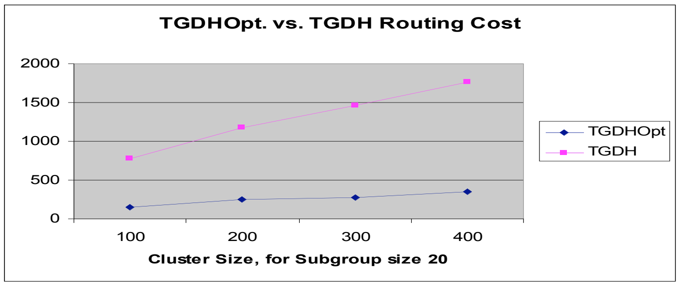
We further assume that while the schedule tree is being generated, the relative placement of members does not change significantly, and consequently, the proximity lists of members do not change much. Such a change could result in a different "optimal" solution, and the one currently being generated would be outdated, and probably suboptimal. On the other hand, our algorithm generates a schedule fast given that the appropriate path information is provided. 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 the nodes, the worse the performance of our algorithm is. Even though DS-TGDH is much more sensitive to mobility than TGDH, it is lowering significantly the overhead associated with the subgroup key generation.

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* subgroup members in a different (random) manner in every repetition. For each random graph with input <*n*, *m*>, and for each subgroup configuration, we have evaluated the total routing cost of DS-TGDH our algorithm *vs.* original 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:

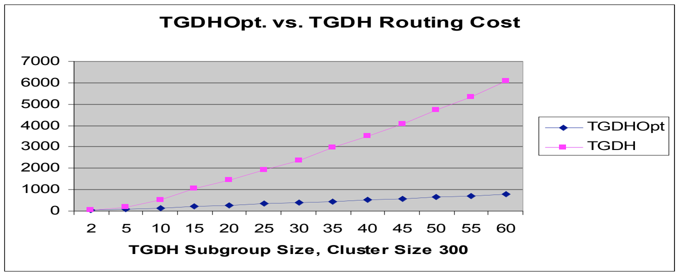
Cluster Size: [100,…500], **Subgroup Size**: [2,…60].

ii) Simulation Results:

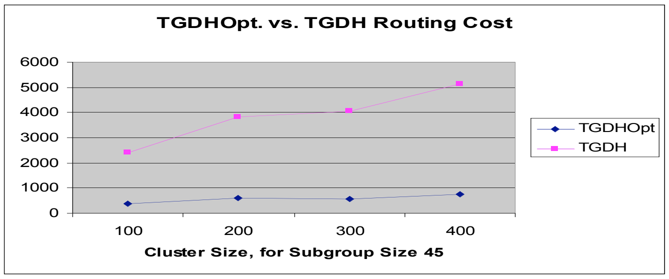
Below we illustrate some indicative results on the routing overhead produced by both protocols DS-TGDH and TGDH, measured in number of hops (relays). The selection of the cutoff number *D* in the proximity lists is determined by the subgroup size. The following graphs reflect the routing overhead produced from the key generation in both schemes. Since the KM messaging is very heavy for the network nodes, our aim is to reduce the overall number of bits (or packets) required and relieve as many nodes as possible from "relaying" a large number of keying data. Furthermore, since it is not always possible to find members capable of undertaking the heavy tasks of the subgroup leader, we would prefer 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 such that: $\frac{DS-TGDH(RC)}{TGDH(RC)}<0.3$. Considering also the amount of redundant key generation data, as calculated in our analysis, members and the resulting relay nodes are burdened with during every round in TGDH, the total savings from the use of DS-TGDH instead, are even more significant. 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 generated and 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 considerably enhance the performance and behavior of the original ancestor scheme. We present a topology aware schedule on top of which the DS-TGDH is executed, and we analytically evaluate the performance and 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. Laboratory 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, J.Stanton, G.Tsudik. "Exploring Robustness in Group Key Agreement". Procs of the 21th IEEE Int'l Conference on Distributed Computing Systems, Phoenix, Arizona, 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 and Privacy, California, USA, May 2003.

[19] M.Striki, J.Baras. "Key Distribution Protocols for Secure Multicast Communication Survivable in MANETs". Proceedings of the IEEE MILCOM 2003, Boston MA, October 2003.

[20] M.Striki, J.Baras "Fault-Tolerant Extension of Hypercube for Efficient, Robust Group Communications in MANETs with Octopus Schemes", CSHCN TR 2005.

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

[22] S.Yi, R.Kravets, "Key Management for Heterogeneous Ad hoc Wireless Networks," University of Illinois at Urbana-Champaign, Department of Commputer Science Technical Report #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 Communications and Networking, Pasadena, CA, 2002.

[24] A.M.Fisiran, R.B.Lee, "Workload Characterization of Elliptic curve Cryptography and other Network 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.