Fault-tolerance and efficiency considerations for key distribution protocols in MANETs.(*)

Maria Striki and John S. Baras

Electrical and Computer Engineering Department
and the Institute for Systems Research
University of Maryland College Park
College Park, MD 20742
mstriki@isr.umd.edu, baras@isr.umd.edu

Abstract: **In this paper we address the Fault-Tolerance and Efficiency of key distribution protocols for group communications in Mobile Ad Hoc Networks. Most key distribution protocols that exist today are primarily designed for wireline networks. These protocols either fail to work as intended or cannot work at all when they are applied to the demanding environment of MANETs. The main reasons for this are: frequent node failures, network partitions and merges, inefficient computational and communication capabilities of certain wireless nodes, network delay, bad quality of signal etc. We determine the framework under which protocols can efficiently work in MANETs, design new protocols or modify existing ones, so that they can be robust, scalable and applicable in this environment. We classify these protocols in two families: contributory and non-contributory. We evaluate them from the point of view of MANETs and compare their performance.**

1. Introduction

A MANET is a collection of wireless mobile nodes, possibly heterogeneous, communicating among themselves over possibly multi-hop paths, without the help of any fixed infrastructure. Furthermore, in wireless mobile networks high mobility may result in nodes frequently going out of range or running out of battery power, leading in temporary links. Collisions, low link quality, distance between nodes and various other factors result in unreliable links or excessive delay in the network. Due to the increasing demand for secure and scalable multicast services in MANETs, key distribution protocols designed for such mobile wireless environments are needed. *Key Distribution* and *Entity Authentication* are the major parts of *Key Management* that ensures secure communications. Here we assume that participating members have already been authenticated and we focus on key distribution only. Most of the current key distribution protocols are designed for wireline networks that are free from most of the constraints of MANETs. Furthermore, the computational power of nodes is considered an issue for some wireless mobile nodes due to resources or capacity limitations. Thus, key distribution protocols that are robust enough to survive or tolerate frequent node failures, network partitions and merges, delays in critical messages, ambiguity to determine the state of group members under certain circumstances, extensive computations etc., are needed. In MANETs, we cannot always guarantee the existence of a node with direct connections to all other participants that can broadcast to the whole group. Also, a change in the topology of a group might occur while the group key is being calculated. In some protocols this event may cause enormous overhead, as the operation of calculating the group key must start all over. These constraints render most group key distribution protocols inefficient in an environment that requires fast operations with the lowest possible overhead.

We classify existing protocols in two families: *contributory* protocols where all participants take equally part in the key generation and guarantee for their part that the resulting key is fresh, and *non-contributory*, where group key generation does not require equal participation from all members.

Our objectives are to study the properties of these two families, discuss and analyze their limitations from the perspective of MANETs. Most of all, we want to design a key distribution protocol that is fault-tolerant to failures that occur frequently in MANETs but not at the expense of efficiency. We claim that the contributory 2^d-Octopus protocol that is based on the Hypercube key exchange scheme is a very appropriate protocol for MANETs: it can tolerate various kinds of failures or resume from failures with minimal overhead. To this end, we designed two new hybrid protocols based on the 2^d-Octopus protocol (GDH.2-based (MO) and (TGDH)-based (MOT)) that are more efficient than the

(*) Research partially supported by the U.S. Army Research Laboratory under Cooperative Agreement DAAD19-01-2-0011.

existing 2^d-Octopus in terms of Computational and Communication Costs. We also compare these protocols to the One-Way Function Tree protocol (OFT) (a non-contributory protocol). The original OFT assumes a fixed group leader with considerable processing capabilities and therefore may not be fault-tolerant or scalable in MANETs. On the other hand, it is considered highly efficient. By comparing the fault-tolerant Octopus-based protocols to OFT, we gain insight about the overhead required to render key distribution protocols robust, scalable and applicable in MANETs.

Some of the most important aspects of *Fault Tolerance* for key distribution protocols that we consider are: the issue of a single, non-flexible, "omnipotent" group leader that may constitute a single point of failure, the issue of whether protocols can recover from members' failure during the group key establishment without starting this very costly procedure all over again, and the issue of whether protocols tolerate frequent node failures, group partitions and merges at any time during a session.

Most **non-contributory** protocols are based on a fixed trusted central controller to distribute the key. Finding members within the group able to replace the faulty leader is not enough. The new leader should securely and quickly obtain all the information gathered by the previous leader up to that point. It would be preferred that the "leader" is selected among group members (*as in contributory protocols*) and have a rather coordinating role, storing the least information possible that can be easily retrieved by any member becoming leader in the future (*as in TGDH*). Furthermore, in order to reduce group partitions and frequent leader elections, we must take into account the mobility of nodes in the network, the robustness, the computational and processing capabilities of individual nodes. One solution is to dynamically select a node as group leader according to a certain policy that makes sense in a MANET (e.g. select the node that stays connected with the largest number of nodes within its group for the largest amount of time), and to make every such leader operate in a rather restricted area of the network. Therefore, we also require that the procedure of leader election be dynamic and flexible. In most non-contributory protocols (tree-based), in the event of a node failure, a new group key is computed by updating only a restricted number of keys. The contributions of members for the key establishment are independent and need not follow a strict ordering. In the event of a node failure or delay to respond, the rest of the nodes proceed normally to the key establishment process.

In a **contributory** protocol like GDH.2, each member is expected to contribute its portion of the key in a defined slot according to strict ordering. If a node does not respond during the given slot, the whole procedure comes to a standstill as all further actions of members depend on the contribution of the "disappeared" member and we cannot always determine on time if the response of the node is simply delayed or lost, or if the node itself is down or out of reach. Inevitably the key establishment process starts all over again. However, contributory protocols still acquire some very important properties: they are most appropriate when no previously agreed common secrets among nodes exist, they reflect the totally distributed nature of a group, and their nature is such that no node constitutes a single point of failure. It would be desirable to derive a hybrid protocol that is fault tolerant in MANETs, efficient, and combines the main advantages of the two families of protocols.

We claim that MO and particularly MOT satisfy these requirements. We prove the fault tolerance of Octopus-based protocols by analyzing in detail scenarios of failures most likely to occur in MANETs. We discuss the modifications we made to the original 2^d-Octopus. We then show how these modifications that lead to the new protocols MO and MOT, improve the fault-tolerance, the scalability and efficiency of the original 2^d-Octopus for MANETs.

2. Previous Work

Becker and Wille [1] derived lower bounds for contributory key distribution systems from the results of the gossip problem and applied them to DH-based protocols. They used the basic DH distribution extended to groups from the work of Steiner *et al* [2]. TGDH by Kim *et al* [10], is a new hybrid, efficient protocol that blends binary key trees with DH key exchange. Becker *et al* [1], introduced the Hypercube protocol as requiring minimum number of rounds. In [5], Asokan added to the Hypercube protocol ways to recover from node failures. Becker introduced the Octopus protocol that required minimum number of messages and then derived the 2^d-Octopus, that combined Octopus with Hypercube to a very efficient protocol that worked for an arbitrary number of nodes. Most protocols from the non-contributory family are based on a simple key distribution center. The simplest is Group Key Management Protocol (GKMP) [9]. The Logical Tree Hierarchy method (LKH) [8], creates a hierarchy of keys. It is more complicated but more efficient. Evolution of the latter is OFT [7], that minimizes the number of bits broadcast to members after a membership change. It was selected to represent the family of contributory protocols.

3. Secure Group Key Agreement and Extensions

3.1 Octopus Protocol

It uses DH key computed in one round as a random input for the subsequent round. Four parties *A, B, C, D* generate a group key using only four exchanges. First, *A* and *B*, then *C* and *D* perform a DH key exchange generating keys α^{ab} and α^{cd} respectively. Then, *A* and *C* as well as *B* and *D* do a DH key exchange using as secret values the keys generated in the first step. *A(B)* sends $\alpha^{\phi(a^{ab})}$ to *C(D),* while *C(D)* sends $\alpha^{\phi(a^{cd})}$ to *A(B)* so that *A* and *C* (*B*, *D*) can generate the joint key $\alpha^{\phi(a^{cd})\phi(a^{ab})}$. Parties $P_1, P_2, \ldots, P_{n-4}$, *A, B, C, D* generate a common group key by first dividing themselves into five groups. *A, B, C, D* take charge of the central control. The remaining parties are distributed into 4 groups: $\{P_i \mid i \in \mathrm{I_A}\}$, $\{P_i \mid i \in \mathrm{I_B}\}$, $\{P_i \mid i \in \mathrm{I_C}\}$, $\{P_i \mid i \in \mathrm{I_D}\}$, where $\mathrm{I_A}$, $\mathrm{I_B}$, $\mathrm{I_C}$, $\mathrm{I_D}$ are pair-wise disjoint, and $\mathrm{I_A} \cup \mathrm{I_B} \cup \mathrm{I_C} \cup \mathrm{I_D} = \{1,...,n\text{-}4\}$. $P_1,\ldots,P_n$ generate a group key as follows:

1. $\forall\ X \in \{A, B, C, D\}$, $\forall\ i \in \mathrm{I_X}$, *X* generates a joint key k_i with P_i via the DH key exchange.

2. *A, B, C, D* do the 4-party key exchange using values: $a=K(I_A),\ldots,d = K(I_D)$, $K(J) = \prod_{i \in J} \phi(k_i)$ for $J \subseteq \{1, .., n\text{-}4\}$ and hold the joint key $K = a^{\phi(a^{K(I_A \cup I_B)})\phi(a^{K(I_C \cup I_D)})}$.

3. The step is described only for *A*. Parties *B, C, D* act accordingly. $\forall\ j \in \mathrm{I_A}$, *A* sends 2 values to P_j: $a^{K(I_B \cup I_A \setminus \{j\})}, a^{\phi(a^{K(I_C \cup I_D)})}$. P_j derives $(a^{K(I_B \cup I_A \setminus \{j\})})^{\phi(k_j)} = a^{K(I_A \cup I_B)}$ first, then $K = a^{\phi(a^{K(I_C \cup I_D)})\phi(a^{K(I_A \cup I_B)})}$.

3.2 Hypercube Protocol

It minimizes the number of simple rounds. 2^d parties agree upon a key within *d* simple rounds by performing DH key exchanges on the edges of a *d*-dimensional cube. We identify the 2^d participants on the *d*-dimensional space $\mathrm{GF}(2)^d$ and choose a basis $b_1,\ldots, b_d$ of $\mathrm{GF}(2)^d$. In round 1, every participant $v \in \mathrm{GF}(2)^d$ generates a random number r_v and does a DH key exchange with participant $v+b_1$ using the values r_v and r_{v+b1}. In round *i,* every participant *v* does a DH key exchange with $v+b_i$, where both parties use as secret value the one generated in round *i*-1. In every round, parties communicate on a maximum number of parallel edges of the *d* cube. All parties share a common key at the end of this protocol.

3.3 2^d-Octopus Protocol

For an arbitrary number of participants that require low number of rounds, the idea of Octopus is generalized. In 2^d–Octopus participants act as in the simple Octopus. However, 2^d instead of four parties are distinguished to take charge of the central control. The remaining n-2^d parties divide into 2^d groups.

3.4 GDH.2 and One-Way Function Tree (OFT)

We omit them for lack of space. All protocols are described in our references and Technical Report [6].

3.5 Tree Group Diffie-Hellman (TGDH) protocol

The DH protocol resembles OFT. The basic differences are the following: any member of the tree can act as a leader (or group security controller-GSC) depending on its position in the tree, a member knows all blinded keys of the tree at any given time, and in TGDH the merging function is the two-party DH key exchange. The secret key *x* of an internal node *s* is the result of the DH key exchange between its offspring *left*(*s*) and *right*(*s*) with associated secret keys *y* and *z*. Then, $x = \alpha^{yz}$ and the blinded key of node *s* is α^x. Any member at any time can become group leader and broadcast a message to all members of the group. During the initial construction of the tree every member becomes a sponsor: computes all nodes from the leaf up to the root and broadcasts them to the group. For every successive level of nodes in the tree, the number of sponsors is reduced to half. Each member knows all keys in its path from the leaf to the root and all blinded keys of the tree.

4. Fault Tolerance Issue for 2^d-Octopus protocol

In [5] the authors claim that the scheme is fault tolerant but they don't analyze all the group disruption cases that may occur in a MANET. Here, we attempt to look at the most frequent scenarios and determine whether the protocol is or can be made fault tolerant. We will use an example to make the study of all cases easier. Each node is assigned to a vertex in the hypercube and has a unique *d*-bit address. The protocol takes *d* rounds. Assume that in round *j*, a node with address *i* performs a two-party DH with the node whose address is $i \oplus 2^{j-1}$. In round *j* neighbors along the j^{th} dimension of the hypercube participate in a two-party DH protocol run. After *d* such rounds, all parties obtain the same key.

1st Round: {000-001, 010-011, 110-111, 100-101}
2nd Round: {000-010, 001-011, 100-110, 111-101}
3d Round: {000-100, 011-111, 001-101, 110-010}

1st round DH keys: a^{AB}, a^{CD}, a^{EF}, a^{GH}.

2nd round DH keys: $a^{a^{AB}a^{CD}}$, $a^{a^{EF}a^{GH}}$

3d round DH keys: $a^{a^{a^{AB}a^{CD}}a^{a^{EF}a^{GH}}}$ (group key).

5. Possible Scenarios:

1. *Node A {*000*} becomes faulty immediately before*

the first round and does not re-appear even after the group key has been derived. Node *B* does not exchange DH keys with any node in the 1st round, it uses a secret of its own, say *BB*. In the 2nd round, all nodes that need to communicate with *A* in the hypercube, communicate with its pair-mate of the 1st round, i.e. *B*. This idea applies for all three rounds.

2. *Two nodes from the same pair become faulty before the first round begins.* Another pair of the hypercube "logically splits" to fill the gap of the disappeared pair: one substitutes the pair that became faulty so that this scenario resembles the previous one. Now however, two nodes instead of one undertake the duties of two pairs of nodes.

3. *A participating node A goes down during the 1st round and does not re-appear during the group key establishment.* As in scenario 1, its pair-mate *B* takes over on behalf of *A* as well; *B* has already computed the common DH key with *A:* a^{AB}. *B* creates a new secret share *BB,* computes a^{BB}. *B* takes over from hereon exactly as in scenario 1 but it communicates both a^{AB} and a^{BB} blinded values to its future pair-mates. The result is that two group keys are computed, based either on the secret share *A* known to *A*, or on the secret share *BB* unknown to *A*. If *A* returns to the group before any data transmissions and authenticates itself to the group, it can reconstruct the first group key itself, by getting the appropriate blinded values of keys that are communicated freely in the network. If it comes back after the 2nd round, it can be sent the partial key α^{CD} by any node that knows this value. *A* requires this to reconstruct the partial key $a^{a^{AB}a^{CD}}$. Even if the node reappears in another part of the network and its direct neighbors are different, the scheme still works. As long as multi-hop communication is supported by the routing protocol, two nodes can be virtual neighbors as well. Assume that node *A* reappears at the end of the 3d round. Then the group key has been already established. Similarly to the previous case, *A* will receive partial information: $a^{a^{CD}}$ and $a^{a^{a^{EF}a^{GH}}}$. On receiving this information *A* adds its own portion, and with the appropriate computations derives the group key. This scenario demands that each node stores the partial keys it computes during all *d*-1 rounds. Considering however that *d* itself is not a very large number, we see that the nodes need not store too many values. The overhead would be significant if we started over the procedure of key establishment.

If *A* does not reappear on time, the second group key that excludes *A* is used. Then, if *A* comes back after the session has started, authenticates itself and still wishes to participate in the group, the group key should be changed again if we want to maintain backward secrecy. However, since the "new" node is a reappearing node we might want to overlook the backward secrecy rule. In this case, *B* can communicate either the group key or the share *BB* to *A*, encrypted with a^{AB}, the common two-party DH key *A* and *B* had initially calculated. This notion can be *generalized* to prevent the group key establishment from starting all over because of the abrupt disappearance of any node during this process. So, during the first round any of the 2^d nodes, computes two values: one generated by itself only, and one based on the contribution of its pair node as we have already seen. The remaining process changes only in that more blinded values are communicated now from one pair to another, and more group-keys are finally derived.

4. *Network merge and partition*. If the group gets partitioned due to bad network connectivity then again there is no need to start the computation of the sub-group keys from scratch. The network gets partitioned in two or more groups, each of which can create new subgroup keys, based on previously stored information, limiting the communication and computation overhead. Given the structure of the subgroups, the communication and computation savings can be more (when pairs of one direction in a subgroup were initially pairs in the original group) or less (when all members of one subgroup acquired the same key in the *d*-1 round of the original group. They have to retrieve partial keys stored from the previous round, take scenario 2 into account and perform a hypercube key exchange the same manner as before, using an alternate direction pattern this time).

The case of subgroups merging into the one original group after communication has been restored is less complicated and less costly. It suffices that one member of each subgroup blinds its own subgroup key, sends it to the rest of the subgroups according to a predefined directional pattern, exactly as we do for the hypercube key establishment via DH exchanges. Then, with some additional DH key exchanges-one per member, all the remaining members of all the subgroups can compute the new group key.

6. Discussion

The fact that the hypercube protocol requires 2^d participants imposes restrictions to the 2^d-Octopus protocol in terms of addition/eviction cases and group merging/partitions. However, the 2^d-Octopus acquires a lot of beneficial properties. The 2^d hypercube scheme is proven to be robust and fault-tolerant for

most cases. In 2^d-Octopus, each member of the hypercube structure is the leader (GSC) of a subgroup of nodes of arbitrary number. Members of a subgroup establish a two-party DH key with the subgroup leader. The GSC uses the partial keys of its members to construct its initial secret share for the hypercube. After the group key is derived, the 2^d GSCs distribute parts of the group key to their sub-group members in a way that a member that does not belong to the sub-group cannot derive the group key. The key distribution protocols the sub-groups support can be selected with broad freedom. The original group is divided into 2^d subgroups. Each of these subgroups can be deployed in a relatively restricted area of the network and it is easier to handle these subgroups in a localized manner. Given the topology of the network we have the freedom to assign to each subgroup from 0 to as many members the protocol allows. This results in less communication overhead within the sub-group, in less bandwidth consumption, in less traffic for the routing protocol. Moreover, if a GSC becomes "faulty", it can be replaced by another node from its own subgroup. It is clear now how these properties render the protocol fault tolerant in the cases of addition/deletion, merging/partition.

7. Brief description of MO and MOT protocols.

We modified the original Octopus (O) protocol by replacing its first step with GDH.2 or TGDH. The protocols derived are denoted as **MO** and **MOT** respectively. The members of the group are divided into 2^d subgroups of equivalent size. The sponsor of TGDH for MOT, and the M_n member of GDH.2 for MO, becomes the sub-group leader (GSC). At step1, GDH.2 and TGDH produce subgroup keys: $a^{ab\cdots z} = a^N$ and $a^{xy} = a^N$ respectively. These keys are equivalent to the subgroup key produced by (O) when its subgroup contains one member only. During the 1st step, each subgroup establishes its own sub-group key, or handles member additions/evictions exactly as indicated by GDH.2 and TGDH protocols.

All these sub-groups are independent from each other and can support different key distribution protocols. The three steps of the protocol are independent modules: at the 2nd step only one secret share (sub-group key) is received from each sub-group, regardless how it is derived. The task of the 2nd step is only to distribute the group key to all sub-group leaders. Then, it is the task of the 3d step to make sure that each sub-group leader distributes the sub-group key to its members. Thus, overall fault-tolerance of the protocol demands fault-tolerance for each step individually. Now, the key distribution protocol of each subgroup comes into play.

The sub-group key distribution protocol in (O), assumes the existence of a single fixed sub-group leader that stores alone all the information of the sub-group. This is not a fault tolerant protocol even when the size of the sub-group is relatively small. GDH.2 is used for sub-groups in MO. Like most contributory protocols, it requires strict ordering during key establishment and cannot tolerate delays and node failures. Even if the overhead for starting over key establishment in the event of failures is much smaller, GDH.2 cannot be considered fault-tolerant. TGDH is used for sub-groups in MOT. This protocol assumes that any node should be ready to become group leader and that the same amount of information is stored in all members with no considerable overhead. Since the size of the sub-group is relatively small, and it is also deployed on a restricted area of the network, TGDH can be considered fault-tolerant and applicable for MANETs. Thus, we can claim that MOT is scalable and fault-tolerant overall.

The 2nd step is identical for all three protocols. For the initial derivation of the group key the 2^d GSCs execute the 2^d–Cube protocol via DH exchanges with initial values the keys obtained from step1. It takes d rounds; in each we have an exchange of 2^d messages for the group key to compute. In the case of member addition/eviction we can re-compute the subgroup key (step1) only for the subgroup where the change of membership has occurred. We observe that for step2 not all calculations have to be done anew. The two-party DH exchanges between GSCs whose sub-group keys were not modified at step1 or during the previous rounds of step2 need not be executed again. Thus, for round i, 2^i DH key exchanges are done. In total $2(2^d-1)$ messages are communicated.

The initial case for step3 continuing with the same example where d=3 is as follows: the 2^d GSCs broadcast d values to their group. For instance, GSC A sends the following d parts of the group key to its member j: $a^{a^{a^{EF}a^{GH}}}$, $a^{a^{CD}}$ and a^{AB/K_j} for (O), or a^B for MO and MOT. The latter value differs for every member in (O), but is the same for MO or MOT since all members of a sub-group share a common sub-group key. The rest (d-1) values are the same for all members in the subgroup for all three protocols. The GSC broadcast to its members such d values and members must do d expns to compute the final group key. As for the addition/deletion case, (d-1) values need not be broadcast anew. They remain unchanged since they are already stored in every member from the previous time. All GSCs send at most one value to every member of their sub-group. Assume in our example that a change of membership occurs in the sub-group of A and its new sub-group key is x. At the

end of the 2nd step A stores: $x, a^{B}, a^{a^{CD}}, a^{a^{a^{EF}a^{GH}}}, a^{a^{a^{AB}a^{CD}}a^{a^{EF}a^{GH}}}$. Intermediate values of A remain unchanged from the previous time and will be used by the members of its subgroup to construct the group key. The number of messages communicated at step3 for MO and MOT is (2^d-1) for the addition/eviction case.

8. Results

Cost	2^d-Octopus (O)	Mod. 2^d- Oct (MO)	Mod.2^d-Oct (MOT)
Initial GSC Comp	$C_E\left(3\left\lceil \frac{n-2^d}{2^d}\right\rceil+2d\right)+\left(\left\lceil \frac{n-2^d}{2^d}\right\rceil^{4/3}+1.25\left\lceil \frac{n-2^d}{2^d}\right\rceil\right)K^2+\left\lceil \frac{n}{2^d}\right\rceil C_{rr}$	$C_E\left(\left\lceil \frac{n}{2^d}\right\rceil+2d\right)+\left\lceil \frac{n}{2^d}\right\rceil C_{rr}$	$C_E\left(2\log\left\lceil \frac{n}{2^d}\right\rceil+2d\right)+\left\lceil \frac{n}{2^d}\right\rceil C_{rr}$ max.
Delete GSC Comp	$C_E\left(3\left\lceil \frac{n-2^d}{2^d}\right\rceil+2+2\left\lceil \frac{d+1}{2}\right\rceil\right)+2\left(\left\lceil \frac{n-2^d}{2^d}\right\rceil-3\right)K^2+C_{rr}$, one $2\left(\left\lceil \frac{n-2^d}{2^d}\right\rceil+\left\lceil \frac{d+1}{2}\right\rceil\right)C_E$ rest	$C_E\left(\left\lceil \frac{n}{2^d}\right\rceil+2\left\lceil \frac{d+1}{2}\right\rceil\right)+C_{rr}$, one $C_E\left(2\left\lceil \frac{d+1}{2}\right\rceil\right)$, rest	$C_E\left(2\log\left\lceil \frac{n+1}{2^d}\right\rceil+2\left\lceil \frac{d+1}{2}\right\rceil\right)+C_{rr}$ one $2C_E\left\lceil \frac{d+1}{2}\right\rceil$ rest
Delete Comm	$\left(2\left\lceil \frac{n-2^d}{2^d}\right\rceil+3\,2^d-2\right)K$	$\left(\left\lceil \frac{n-1}{2^d}\right\rceil+3\cdot 2^d-2\right)K$	$\left(\log\left\lceil \frac{n-1}{2^d}\right\rceil+3\,(2^d-1)\right)K$

Table 1: Comput. and Commun. costs of key agreement protocols

Table1 shows some of the comparison results. The performance evaluation for OFT can be found in [6], [7]. MOT demonstrates the best behavior for Initial GSC Comput.. For GSC Add/Evict Comput., (O) is the worst, MO outperforms OFT for certain cases (small d, large n), MOT has the best performance. MO behaves poorly in terms of Initial Commun.. MOT and (O) however perform much better: they slightly outperform OFT. For the Addition/Eviction Communication (critical in MANETs), (O) is outperformed by both MO and MOT, and MOT gets closer to OFT than any other contributory protocol.

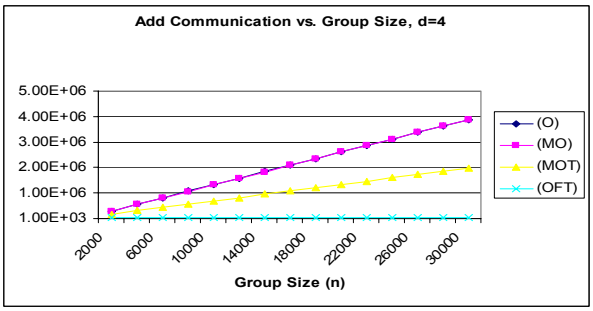

Fig. 1: Add Communication *vs* Group Size, d=4. OFT achieves the lowest overhead. MOT gets quite close to OFT. The overheads of MO and (O) are similar but still much worse.

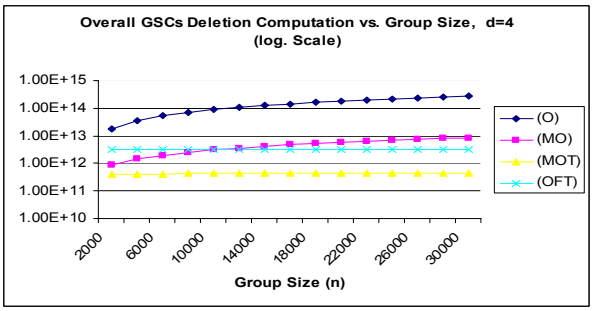

Figure2: Delete Computation *vs* Group Size, d=4. MOT achieves the lowest overhead. OFT and MO have similar performance but (O) behaves poorly.

9. Conclusions

The paper addresses the issues of Fault-Tolerance and efficiency of key distribution protocols intended for MANETs. We show that Octopus-based protocols in general are fault-tolerant in MANETs. We present two novel hybrid protocols MO and MOT-based on the original Octopus, developed as an attempt to render contributory protocols scalable and efficient. The cost functions for all three protocols are derived in terms of communication and computation. From our performance evaluation, we demonstrate that MOT is the most efficient of all protocols in terms of the overall computation cost and achieves the lowest communication overhead among the Octopus-based protocols, getting quite close to OFT.

10. References

[1] K. Becker and U. Wille, "Communication Complexity of Group Key Distribution", *Proc. 5th ACM Conf. on Comp. & Comm. Security*, pp. 1-6, Nov. 1998, ACM Press.

[2] M. Steiner, G. Tsudik and M. Waidner, "Diffie-Hellman Key Distribution Extended to Groups", *Proc. 3rd ACM Conf. on Comp. and Comm. Security*, pp. 31-37, 1996, ACM Press.

[3] M. Hietalahti, *Efficient Key Agreement for Ad-Hoc Networks*, M.S. Thesis, Helsinki University of Technology, Department of Comp. Science and Eng., Finland, May 2001.

[4] A. Perrig, "Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication", *International Workshop on Cryptographic Techniques and E-Commerce CrypTEC '99.*

[5] N.Asokan and P. Ginzboorg, "Key-Agreement in Ad-Hoc Networks", in *Computer Communications*, Vol. 23, N. 17, pp. 1627-1637, 2000.

[6] M. Striki and J.S. Baras, "Efficient Scalable Key Agreement Protocols for Secure Multicast Communications in MANETs", *CSHCN Technical Report* 2002.

[7] D. McGrew and A.T. Sherman, "Key-Establishment in Large Dynamic Groups Using One-Way Function Trees", *Technical Report No. 0755, TIS Labs at Network Associates*, Inc., Glenwood, MD, May 1998.

[8] H. Harney and E.Harder, "Logical Key Hierarchy Protocol", *Internet Draft*, Internet Eng. Task Force, March 1999.

[9] H. Harney and C.Muckenhirn, "Group Key Management Protocol (GKMP) Specification", *RFC 2093*, Internet Engineering Task Force, July 1997.

[10] Y. Kim, A. Perrig and G. Tsudik, "Simple and Fault Tolerant Key Agreement for Dynamic Collaborative Groups", *Proc. 7th ACM Conf. on Comp. and Comm. Security*, pp. 235-244, Nov. 2000, ACM Press.