A Level Key Infrastructure for Secure and Efficient Group Communication in Wireless Sensor Networks

Jyh-How Huang, Jason Buckingham, and Richard Han
Department of Computer Science
University of Colorado, Campus Box 0430
Boulder, CO 80309-0430
Email: Jyh.Huang@colorado.edu, buckingj@colorado.edu, rhan@cs.colorado.edu

***Abstract*— Group communication to and from sets of sensor nodes is an important paradigm in wireless sensor networks (WSNs). Securing this group communication is a difficult challenge given the energy-efficiency constraints posed by WSNs. In this paper, we introduce the protocol SLIMCAST, i.e. Secure Level key Infrastructure for MultiCAST and group communication, which uses level keys to provide an infrastructure that dramatically lowers the cost of nodes joining and leaving sensor groups. This level key infrastructure is shown to achieve energy-efficient key updates that are localized for group multicast 1→N communication, and can be further leveraged to achieve secure group aggregation N→1 communication. Simulation results comparing the performance of SLIMCAST to traditional secure group communication protocols are presented to demonstrate SLIMCAST's energy efficiency and flexibility.**

I. Introduction

Group communication in wireless sensor networks (WSNs) is emerging as an important communication paradigm. A WSN is typically organized as a hierarchical tree network, with leaf sensor nodes sending data to a root base station collection point via a multi-hop wireless routing network. Each micro sensor node is resource-constrained, with severe limitations on its energy lifetime, memory, CPU, and radio bandwidth. It is often important for the base station to communicate to groups of resource-constrained sensor nodes, e.g. all the temperature nodes in a given region. The base station may wish to dynamically reprogram or retask [19], [1] groups of sensor nodes, namely reset their trigger thresholds, recalibrate the sensors, etc. Similarly, groups of sensor nodes may need to be awakened to track targets moving through the sensor network [18].

Multicast 1→N communication and aggregation N→1 communication are two especially attractive forms of group communication for WSNs because they are both bandwidth-efficient and energy-efficient. A multicast routing tree routes a packet the minimum number of times needed on each link. For example, Figure 1 illustrates three multicast trees in the same WSN based on the sensor types temperature, humidity, and smoke. Each multicast tree is organized efficiently, so that packets are never sent along links that do not eventually lead to an interested sensor node. Aggregation reverses the direction of communication to be towards the base station up the tree. Aggregation efficiently uses bandwidth in that a node aggregates the sensor data reported by its N children before forwarding the compressed aggregate onto its parent for further aggregation[30], [15], [32].

Security is an important issue in WSN research. Applications of WSNs often include military deployments [2]. In such scenarios, in-situ WSNs face many security risks. First, wireless communication between sensor nodes is susceptible to eavesdropping, jamming, spoofing, and DoS attacks. Second, the resource constraints limit the type of security countermeasures that may be employed. For example, these resource constraints severely limit the applicability of compute-intensive public key approaches[20], such that sensor network security primarily focuses on symmetric key techniques. Third, in-situ sensor nodes and base stations are at risk of physical discovery. In this case, sensor nodes may be destroyed or worse, compromised.

This paper focuses on developing an infrastructure to support secure and energy-efficient group communication in WSNs for both multicast and data aggregation. Traditional secure multicast protocols such as LKH [11] and logical stars [12] incur heavy overhead for key update or rekeying events whenever a node wishes to join or leave the multicast tree, and as a result are largely unsuitable for WSNs. Members of a multicast tree typically have a key that is used to decrypt the data sent from a source. Normally, when a new node wishes to join a secure multicast tree, it is necessary to update this key with a new key to maintain *backward secrecy* [22]. The idea is to prevent a node with the new key from going backwards in time to decipher previous content encrypted with prior keys. Likewise, when a node leaves, it is necessary to update the key to maintain *forward secrecy* [22]. The idea is to prevent a node from using an old key to continue to decrypt new content. Traditional multicast protocols suffer heavy key update overhead since every node in the multicast group needs to be updated on every join and leave event.

To achieve more lightweight operation for key updates, SLIMCAST employs an approach based on subdividing the group routing tree into levels and branches. Each level in each branch of the group tree uses its own *level key* for decrypting or encrypting group data packets. When a node joins or leaves, only the local level key needs to be changed, rather than all or a large fraction of keys, resulting in dramatic energy savings compared to traditional key management methods.

SLIMCAST fits within the class of geographically-rooted *cluster based* key management frameworks. We explain in the

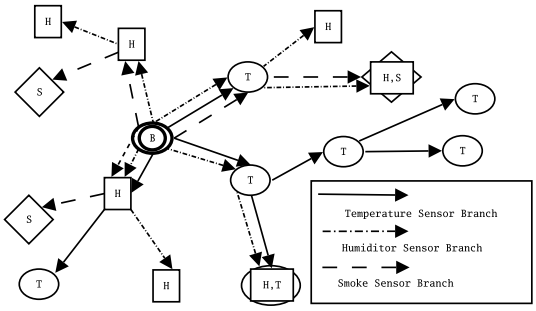

Fig. 1. Three multicast trees residing in one sensor network.

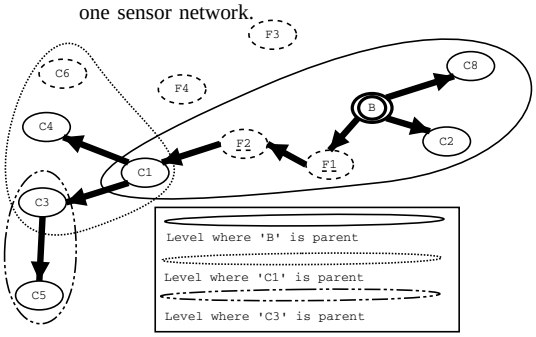

Fig. 2. Sensor network running SLIMCAST, 3 *levels* are formed

following how SLIMCAST improves upon two representative cluster-based key management frameworks, namely the Iolus framework developed for wired multicast networks[23], and the LEAP framework for WSNs[32]. When compared with a protocol family that builds logical key trees, e.g. LKH, LKH+ [4], OFT [3], [21], and ELK [25], SLIMCAST differs by constructing keys to closely adhere to the physical tree structure of a WSN, which ultimately enables reduced bandwidth for key update events.

In Iolus, nodes are grouped into clusters with a clusterhead (IOLUS uses the term GSA) controlling the cluster key for each cluster. The clusters are arranged hierarchically with a top-level cluster and parent-child relationships between clusters on down the hierarchy. A key update event is confined to a given cluster, thereby significantly reducing key update overhead. Iolus largely bypasses the critical issue of how to assign the clusterheads as well as the initial cluster keys. Instead, the backbone hierarchy of clusterheads is assumed to exist, and the focus is on key management thereafter. In WSNs that are deployed in-situ, it is crucial to provide a secure mechanism for building this hierarchy of clusterheads from scratch, so that the WSN is able to self-configure during/after deployment. SLIMCAST provides such a general framework for dynamically building this cluster key hierarchy from scratch in a secure and efficient manner.

In LEAP, cluster keys are built from randomly predistributed pairwise keys. Each node acts as the center of its own cluster, and chooses a cluster key, which is unicast to each neighbor with which it shares a pairwise key, encrypted by that pairwise key. LEAP provides these low-level primitives but does not address how to securely form multicast routing or aggregation trees from local cluster keys. SLIMCAST hierarchically organizes cluster keys via levels. Another difference is that LEAP's cluster key is formed only with immediate neighbors with whom a pairwise key is shared. Two interested parties in a multicast tree may be separated by multiple hops of uninterested nodes whose only duty is to forward multicast data between member nodes. SLIMCAST's level keys provide a more general framework for accommodating both member and non-member nodes. A third difference is that in LEAP compromise of a node's pool of pairwise keys allows an adversary to join anywhere in the network where at least one compromised key from the pool is shared with another node, say Y. This will also reveal the cluster key of Y to X. In contrast, SLIMCAST makes a more restrictive assumption to protect against node compromise, namely that each node starts out by sharing only a pairwise key with the base station. Level (cluster) keys are securely bootstrapped from this starting point. Compromise of a node does not allow an adversary to go elsewhere in the network and join because neither the compromised pairwise key nor the compromised cluster keys are immediately trusted by nodes elsewhere in the network, unlike pool keys. In this way, SLIMCAST trades off a longer setup process for more resiliency against node compromise.

In summary, SLIMCAST is distinguished by the following features:

- secure support for the different roles of member nodes and non-member forwarding nodes in group communication trees
- secure support for both 1→N multicast and N→1 aggregation
- secure support for both periodic joining and on-demand joining and leaving
- its lightweight overhead, accomplished through the use of level keys

The remainder of the paper begins with a high-level overview of levels and level keys in Section II. Section III addresses the core of the protocol including how nodes join a multicast group, how level keys are bootstrapped, and how SLIMCAST can flexibly support upstream aggregation/in-network processing in addition to downstream multicast. Section IV describes how SLIMCAST supports dynamic On-Demand Joins and Leaves. Section V explores other security issues with SLIMCAST. In Section VI, we analyze SLIMCAST's performance in simulation compared to LKH and Iolus in terms of overhead and energy consumption. Related works and conclusion can be found in section VII and VIII respectively.

II. Defining the Level Key Infrastructure

SLIMCAST's *level key* infrastructure enables secure and efficient group communication in WSNs. As shown in Figure 2, a multicast tree naturally divides into branches and levels. A level is roughly defined in terms of hop count along a particular branch. Suppose all nodes in a WSN participate in a multicast tree. In this case, a level is defined as a parent and all of its immediate children nodes on a specific branch. However, SLIMCAST makes the additional distinction between *member*

nodes and non-member *forwarding* nodes. These forwarding nodes are not interested in the contents of the packets sent along the multicast tree, but must be part of the routing of the tree in order to provide connectivity between member nodes. Given both member and forwarding nodes, a level is defined as a parent member node and all of its children member nodes, including possibly intervening non-member forwarding nodes. Forwarding nodes can route data, but cannot decrypt the data, since forwarding nodes will not have access to the level keys. Level keys are only established between member nodes.

Figure 2 illustrates three such levels. Level one consists of a parent node B, member nodes C1, C2, and C8, and forwarding nodes F1 and F2. Though F3 and F4 are also candidates for level one, they are not chosen for the multicast tree because SLIMCAST constructs a shortest path multicast tree. Level two consists of a parent node C1 and member nodes C4 and C3. Note that level-2 member nodes can be more than two hops away from the base station B, but are exactly two hops from B in terms of the number of "members-only" hops, i.e. C3 must go one hop to member node C1 and a second "members-only" hop from C1 to reach B.

To accomplish the goal of ensuring that key updates are localized in their impact, SLIMCAST assigns a *level key* to each level and branch. For example, in Figure 2, C1, C2, and C8 all share the same level-1 key along with their parent B. C3 and C4 share the same level-2 key, along with their parent C1. C5 shares the same level-3 key with its parent C3. Note that a level key is specific to a branch. Thus, the level-3 key on one branch is completely independent of the level-3 key on another disjoint branch.

Because a *level key* is the only key shared within a $level$, key update events, include joining and leaving, will only affect one $level$ under the SLIMCAST structure. This reduces the overhead and energy caused by key update events.

III. Secure Topology Discovery and Level Key Bootstrapping

This section describes the setup of the level key tree-structured hierarchy enabling secure group routing and data aggregation. We begin with a set of assumptions concerning the topology and shared security inherent in the WSN:

- The network contains one or more base stations, where a base station is significantly more powerful than the other nodes in the network.
- The routing structure is a shortest path tree rooted in each base station. Sensor nodes forward their data along this tree towards the base station, while the base station can send messages along this tree towards one or more sensor nodes.
- The network wishes to support dynamically created groups of sensor nodes organized into trees. These trees should support both aggregation and multicast.
- The dynamic formation of these groups must be both secure and efficient for WSNs.
- Base stations are the only nodes that can initiate a broadcast control message to construct group-based trees, called a Join Query.
- Base stations cannot be compromised, while sensor nodes can be compromised.
- Each base station B shares two unique secret keys with each node N in the network, KBN_a and KBN_b. Key a is always used for encryption while key b is always used for MACing. Initially sharing only a pairwise key with each base station improves resiliency against node compromise over the pairwise key pool approach.
- Each node must store at least two cryptographic keys and an initial one-way hash chain value per base station when it is deployed.

Given these assumptions, the basic structure of SLIMCAST consists of a *three-way handshake* process that allows new nodes to securely join the WSN tree for a given group and base station and set up their level keys:

1) JOIN QUERY: A given base station will broadcast a Join Query message to solicit interest in a given group. This sets up an initial tree structure.
2) JOIN REPLY: Interested nodes will reply to this solicitation, providing proof to authenticate their identity, and issuing a challenge. This pares the tree structure to interested and authenticated nodes.
3) JOIN CONFIRM: The network will reply to the authenticated node with the appropriate response to the challenge, so that the node can complete mutual authentication of the network and obtain its level key. This completes the secure tree hierarchy.

A summary of the complete process is illustrated in Figure 3 and Figure 4. A variety of higher-level considerations guided the development of this three-way handshake approach. The tree needs to be constructed in a manner that is secure, efficient, and dynamic. Mutual authentication is an important feature to support, as is resiliency against node compromise. The Join Query broadcast is required as the first step in dynamic topology discovery, i.e. setting up the initial routing tree for this group and providing a path for replies. A one-way hash chain number is included in this Join Query to limit the ability of an adversary to arbitrarily flood the network with a spoofed Query.

Interested nodes will reply along paths established by the Query. The Reply contains a MAC signature verifying the joining node to the base station. This Reply must propagate to the base station for verification because, initially, the intermediate nodes do not share any secrets with the joining node. SLIMCAST employs rate-limiting on Join Replies to prevent denial of service (DOS) Reply floods. Since SLIMCAST builds its secure group tree from the inside out, then the location of reply floods can also be detected by the base station as the point of entry into the already constructed tree. SLIMCAST uses this property to block Reply floods from the suspect entry point. To further complement this Reply filtering, we can assume the existence of pairwise key pools, though not for level/cluster key construction. In this approach, similar to SEF[30], pairwise keys are used for filtering not construction, i.e. bogus Replies are filtered if they don't have the appropriate MAC pairwise signatures. This is effective against malicious outsiders, and against compromised nodes seeking to launch

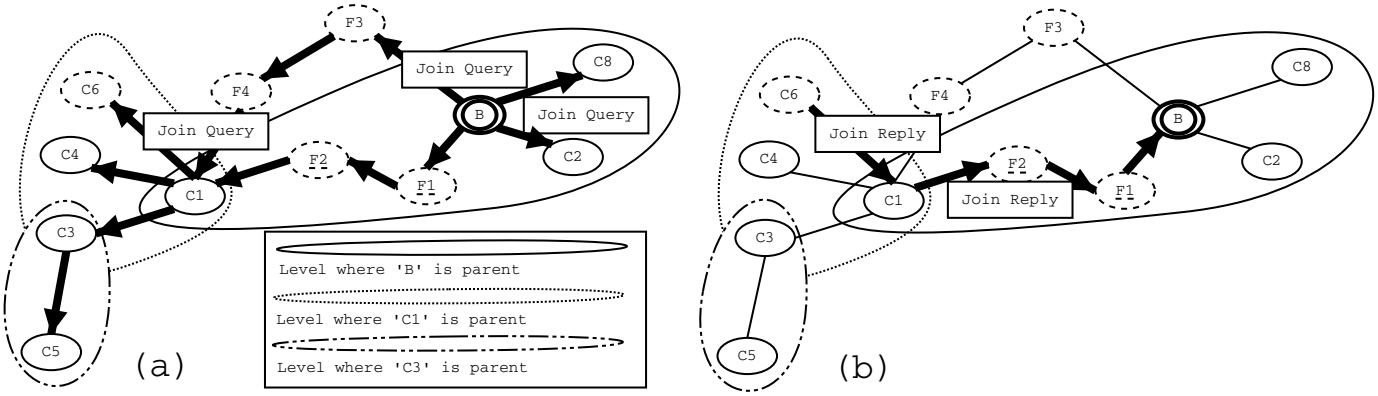

Fig. 3. (a) Base station floods out Join Query (b) Interested node $C6$ sends Join Reply with Challenge to base station.

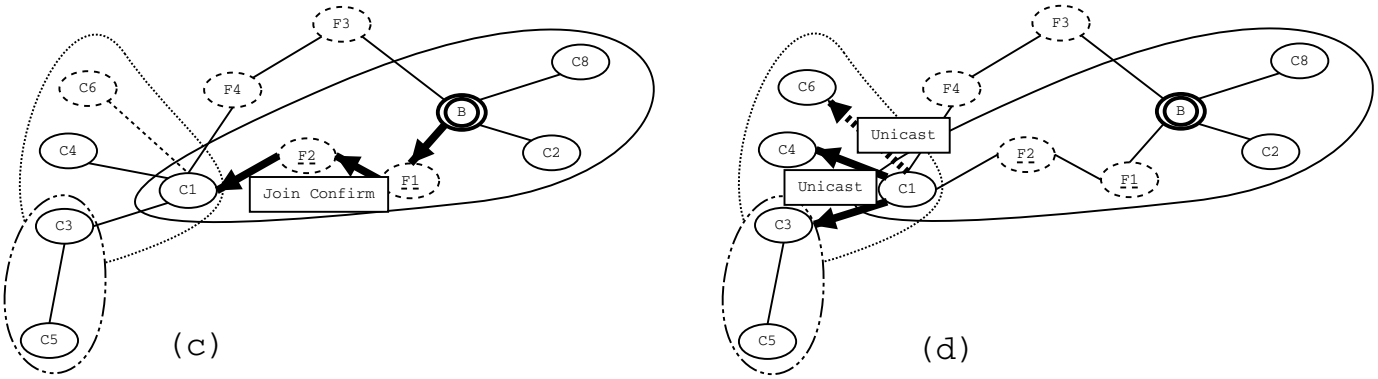

Fig. 4. (c) Base station sends a Confirm and Response of C6 to C1 (C6's parent) (d) C1 uses Response to setup new level key with C6 and unicast new level key to each existing member

Sybil attacks by inventing many identities. Replies from a compromised node that retains its identity will not be filtered by this technique alone.

The Reply also needs to contain a challenge to complete mutual authentication. The joining node generates a challenge that is encrypted by its pairwise key with the base station. This challenge is decrypted by the base station, and the confirmation response key is generated, which is some function of the challenge.

The third step is to securely add this new node to the existing group tree, now that it has been authenticated by the network. The Confirm message is unicast to the *member* node in the tree that is a *parent* of the joining node. This Confirm message informs the parent member node that it has a valid child that wants to join and also gives that parent member node the response so it can provide proof to the joining node of the network's validity.

SLIMCAST then allows the parent member to select the new local level key and propagate it both to the joining node and existing child members. In effect, the parent member node has been delegated authority to act as a clusterhead. The base station is thus involved only in verifying the new node on joins, not in selecting new level keys. This design choice is especially efficient when nodes are leaving, because all level key update activity is local and the base station is not involved at all. The new level key is unicast to the joining node from the parent encrypted by the response key, thereby authenticating the network to the joining node. The new level key is also unicast to each existing node, encrypted by the response keys used by the respective member nodes when they joined earlier. Unicasting the level key ensures backward secrecy.

A. Secure Setup: Join Query Broadcast

This section provides details on the first phase, namely the "JOIN QUERY" broadcast by a base station to the network so that all interested nodes can join the multicast group by replying to the Join Query. The packet format is

$$JoinQuery + GroupID + BaseID + OWS + LastHop$$

In our notation, the plus sign $+$ denotes concatenation. Also throughout this paper we will use $GroupID$ to identify the particular multicast group, and $BaseID$ to identify the base station setting up the multicast group. Using both the $GroupID$ and $BaseID$ identifiers enables support for networks with more than one base station and provides the opportunity for each base station to set up several multicast groups. The $LastHop$ field is overwritten at each hop with the current node's ID before the packet is forwarded. This field is then stored by receiving nodes to indicate who the receiving node's parent is and is also used to construct a reverse path for unicasting packets back to the base station.

The OWS field is a one way sequence number based on a one way hash chain, and is used for verifying the legitimacy of this JOIN QUERY. The idea of one way sequence numbers is described in [24] and [7]. By using OWSs, an arbitrary node

will not be able to flood a Join Query to our WSN because we check the authenticity of each packet before further processing occurs. Additionally, the OWS prevents a replay attack in much the same way standard sequence numbers do. All nodes in the WSN will be pre-programmed with the one way hash function F_{ows} that is used to compute and verify the OWS. Therefore, our sensor nodes will not waste energy replying to and forwarding adversary-forged Join Queries. The usage of OWS can be simplified and presented in this pseudocode

On receiving OWSNEW
if ($F_{ows}(OWSnew) == OWSold$) then
  OWSold = OWSnew
  forward JOIN QUERY packet
else drop JOIN QUERY packet

This scheme is still able to authenticate a packet even when some packets have been lost if the pseudocode is modified to loop when $F_{ows}(OWSnew)$ is not equal to $OWSold$, instead of dropping the packet immediately. The one way hash function can be applied N times where N is some number of packets that can reasonably be tolerated as lost. After each successive application of the hash function, we can again check it against the $OWSold$ value. If it is ever found to be equal, we have authenticated the packet and can forward it. If it is not equal within N steps, then we can drop the packet.

It is important that the value of N be chosen wisely. Choosing a value that is too low will cause packets to be dropped even when they contain a legitimate OWS, but choosing a value that is too high will enable an OWS sync attack. An OWS sync attack is an attack mounted to waste a node's limited energy. Therefore, we suggest N should be a reasonable number between 3 to 5, a value that a link with reasonable quality should be able to satisfy by not losing that many consecutive packets.

An attacker is still limited even though the Join Query packet is not protected with a MAC. The Join Query packets will be flooded out rapidly regardless of what an attacker does. If an attacker modifies a particular Join Query packet and floods it out, the attack will only affect the small portion of the network that is downstream from the attacker, at downstream nodes that the attacker's modified packet is able to reach before the legitimate packets do. Also, the attacker can only modify the $GroupID$ field, which will cause false unsolicited replies to the base station, which will drop them. The attacker cannot change the OWS of the Join Query because it will invalidate the packet and everyone will drop it. Likewise, he cannot modify the $BaseID$ value because the OWS will only verify based on the associated base station (since each OWS value is associated with a particular base station).

B. Secure Setup: Join Reply and Confirm

This section provides a detailed explanation of phases two and three. Now, suppose a node C6 shown as a dotted node in Figure 3 wishes to join the multicast group after hearing the Join Query. In SLIMCAST, the new node will send a request to join the multicast tree, called a JOIN REPLY message. The format of the JOIN REPLY message sent by C6 will look like this packet below

$$Join + GroupID + BaseID + MEMBER + NodeID + \{GenKey\}_{KB\_Node_a} + MAC(\text{packet})_{KB\_Node_b} + LastHop + NextHop$$

"Join" identifies the message as a JOIN REPLY message; "MEMBER" identifies the new node as interested in membership, as distinguished from a forwarding node which will be described later; and $NodeID$ denotes the joining node. At this point, the node generates a new random key value, $GenKey$, and encrypts it. This key value will later be used to communicate with the node's logical parent. Next a MAC of the packet is calculated and encrypted using the b key that is shared between the node and the base station. In our notation, $MAC(\text{packet})_{KB\_Node_b}$, the term "packet" indicates the portion of the packet up to where the associated MAC begins. Finally, as the node is sent back to the base station, the fields "LastHop" and "NextHop" are updated. LastHop identifies the node currently forwarding the packet, while NextHop identifies the next node that should receive the packet. Note that the NextHop value is already known and was stored locally in each node's routing table when the initial Join Query message was flooded throughout the network.

After the base station receives and verifies this Join Reply, it will generate a CONFIRM CHILD message. The Confirm Child packet looks like

$$ConfirmChild + GroupID + BaseID + MEMBER + LogicalParent + \{JoiningNode + PhysicalParent + GenKey\}_{KB\_LPNode_a} + MAC(\text{packet})_{KB\_LPNode_b}$$

The Confirm Child packet is going to be unicast to the logical parent of the joining node, rather than directly to the new node. The term *logical parent* denotes the first node that is upstream from a given node and is already a member of the multicast group. In this case, $LogicalParent$ represents the logical parent of the joining node. *Physical parent*, on the other hand, is the node that is exactly one hop upstream from a given node. The physical parent and the logical parent will often be the same, but that will not always be the case. In Figure 4, C1 is both the physical and logical parent of C4 since C1 is a member of the group. However, node F2 is the physical parent of C1, while B is the logical parent of C1 since neither F2 nor F1 are members of the group.

The base station decrypts the $GenKey$ field from the Join Reply packet that was just received and then appends it to this Confirm Child packet. The data $JoiningNode + PhysicalParent + Response$ is encrypted and MAC'd with the appropriate keys shared between the base station and the logical parent of the joining node.

C. Localized Joining

After the parent has received, verified, and decrypted the Confirm Child packet, the parent will compute new level keys. It will then send a PARENT CONFIRM message to the new child that looks like

$$ParentConfirm + GroupID + BaseID + JoiningNode + \{LogicalParent + KC1'_{LEVELa} + KC1'_{LEVELb}\}_{GenKey_a} + MAC(\text{packet})_{GenKey_b}$$

The unicast includes sending the two new level keys, $LEVEL_a$ and $LEVEL_b$, one for encryption and one for MACing. For the new child node, $JoiningNode$, the new

level key will be encrypted and MAC'ed by the $GenKey$ value received in the Confirm Child packet (here $GenKey_a$ is the same key as $GenKey_b$). A node can also generate two separate keys, $GenKey_a$ and $GenKey_b$, and send them both in the Join Reply. We include $LogicalParent$ in this transmission because the node will need to know its logical parent for data aggregation purposes which will be discussed later in Section III-E. Upon receiving this packet, the newly joining child knows that the packet is valid by verifying the attached MAC.

For all existing member children, the parent unicasts the new level key to each member child, encrypted and MAC'ed using the pairwise key(s) already established between the parent and that child. Unicasting preserves backward secrecy. The pairwise key(s) used could be the original response value(s), or the original response value(s) could have been used to bootstrap new pairwise key(s). This new level key is sent to all member children (except the newly added child) in a packet format similar to the Parent Confirm message.

D. Secure Data Multicasting

Now that the multicast trees have been securely set up, there are two different ways to multicast data to the group members in SLIMCAST, namely hop-by-hop encryption and SLIMCAST-Hybrid. In hop-by-hop encryption, the base station first multicasts by encrypting an entire packet with its level key. Each member node that is a logical child of the base station will then decrypt the message. If the member node receiving the packet is a parent of a subsequent level, the node will re-encrypt the message using its parent level key and send the message onto each of its children. The packet format looks like

$$Normal + GroupID + BaseID + \{Msg\}_{KB_{LEVEL_a}} + OWS + MAC(\text{packet})_{KB_{LEVEL_b}}$$

At each level the packet will be decrypted and re-encrypted with the next level's keys before being sent on. An alternative would be to have the base station generate a random key, K, and encrypt the message with this key. At each level, the parent decrypts and re-encrypts K, but not the entire message. This would speed propagation of the message by saving us from having to decrypt and reencrypt the entire message at every level during packet forwarding. According to [24], sensor nodes typically send very small messages around 30 bytes or so. If we added an encrypted 128 bit (16 byte) key to every message, this would significantly increase the length of each packet. For a packet of 50 bytes, a 16 byte key would comprise about 24% of the packet. Additionally, in WSNs, transmission is much more expensive than computation. We thus chose not to add a random key to each packet.

The second scheme is SLIMCAST-Hybrid. We use one global key for the multicast group and use the SLIMCAST structure to update the global key when there is a joining or leaving event. The global key update event should be initialized by the parent who has the level key update event and notifies the base station of that. After the level key update is done, the base station can then distribute the new global key via the level key structure, which will decrypt and re-encrypt the global key at each level and exclude the leaving member from this key updating event because it doesn't know the new level key. Hybrid eliminates the energy overhead for hop-by-hop re-encryption but affects all multicast nodes when there is a key update event. Still Hybrid uses less energy in key update events compared with Star, LKH, and Iolus. In a leaving event, LKH will send a large multicast packet (Length LogN) to the entire group, while Star will have to unicast to every single member. SLIMCAST-Hybrid will send a small multicast packet (new group key with length 1). The choice between hop-by-hop and Hybrid configurations will depend on how frequently key update events occur.

E. Support for In-Network Processing and Data Aggregation

In WSNs, in-network processing, most commonly in the form of secure data aggregation, is important for the network to efficiently gather data while eliminating redundancy and thus saving energy [30], [15]. By the nature of sensor networks, it is much more efficient if the network can aggregate similar data locally before sending it through the network to the base station. Additionally, protocols that perform data aggregation have to ensure security against an attacker who injects bad data to maliciously affect the aggregated report sent to the base station. In our protocol, we assume that groups of nodes that will gather data that should be aggregated will join the same multicast groups. As stated previously, the level key infrastructure will likely vary from multicast group to multicast group depending on the group members, and this feature potentially adds some benefits for data aggregation.

Nodes must send observed data towards the base station *in a manner that allows upstream nodes to view the data*. By doing this, the upstream nodes will be able to combine the data with other gathered data if the observed values are similar enough. Therefore, the network must support communication among nodes that are physically located near one another if we hope to perform aggregation. Protocols that encrypt data with a key shared only with the base station prohibit aggregation because no nodes along the path back to the base station will be able to decrypt the message and aggregate it.

With the level-key network already established by our protocol, several aggregation possibilities exist. When an event is observed, each node that observes the event should send a data packet to its logical parent encrypted with either the level key or the unicast key that it shares with the parent. Encrypting with the level key allows for passive participation which will be addressed below. At this point, several possible options for aggregation exist. First, if a node's parent, $P0$, receives at least N reports of the event, where N is some number of packets deemed large enough to securely aggregate data, the parent can aggregate the data itself and then send it directly to the base station using the key that it shares only with the base station. Note that the parent must have at least N children to receive N different reports. If the parent $P0$ does not receive at least N similar reports within a specified time limit, instead of aggregating the data it can re-encrypt each data packet that it received *with a key it shares with its parent* (again this can either be the level key or unicast key depending on the needs of the aggregation protocol), $GP1$, and

forward the data unchanged up one level. At this point, $GP1$ will presumably have children other than just $P0$. Additionally, since $P0$'s siblings are likely located physically near one another, the probability of these siblings and their children gathering similar data is high. We can therefore assume that $GP1$ will also receive several data reports from each of of its children, enabling it to aggregate all of the data, and then send it directly to the base station.

The process of re-encrypting the data packets with the key that a node shares with its parent and sending the data upstream can repeat indefinitely until enough data accumulates that it is aggregated or all of the data reaches the base station unaggregated. With each hop that the data moves towards the base station, the likelihood that the data will ever get aggregated is reduced because nodes that are physically near one another will most likely share a parent or a grandparent. Regardless of the network density, for any two nodes we can always identify a *least common parent*,i.e. the first node that two nodes have in common when we only travel in the upstream direction. We can then determine a value H for the network, such that if the least common parent of two nodes is more than H hops away from one of the nodes the likelihood of these two nodes sharing any children nodes that are located near one another is very small. Therefore, based on this premise, the aggregation protocol mentioned above could be modified to only re-encrypt and propagate the data up one level H times before giving up hope that the data will ever be aggregated and then unicasting each data packet directly to the base station. By aborting the propagation upward one level at a time after H hops, the network saves the energy of continuously decrypting and re-encrypting the data at each level as it progresses upward towards the base station. Depending on the density of the network, H will likely have a value somewhere between 2-4.

This aggregation scheme is only one possible way of aggregating data within the SLIMCAST infrastructure. Security could be added to this protocol in much the same way it has been added in Secure Information Aggregation [26]. It should be noted that our level key infrastructure as it stands fully supports certain existing aggregation schemes such as SIA. SIA assumes that each node shares a unique key with the base station and a separate key with the aggregator nodes. SIA could be run on top of the SLIMCAST network by defining certain nodes to be aggregator nodes, such as each level parent or only certain level parents, because each child node already shares both a unicast key and a level key with its parent. Each node also shares a unicast key with the base station, as required by SIA, so SLIMCAST can support data aggregation with SIA.

Another form of in-network processing is *passive participation* and is described in [32]. The idea with passive participation is that nodes overhear data transmissions and notice that other nodes are already reporting the same data. Upon overhearing such messages, other nodes do not report their same data. While data aggregation and passive participation are mutually exclusive, the SLIMCAST network is able to support either. Passive participation is possible because of the use of level keys in SLIMCAST, so nodes can encrypt their data using their parent's level key and many of their sibling nodes can overhear the message and not send their data packet if it is very similar.

F. Other Issues

Due to lack of space, some details of SLIMCAST will not be able to be described in this paper. SLIMCAST provides a procedure for secure integration of forwarding nodes into the group tree and for handling on-demand joining of forwarding nodes. In addition, unicast routing tables for each downstream node must be constructed by SLIMCAST during the setup phase.

IV. On-Demand Join and Leave

A protocol that allows for efficient joins and leaves is most beneficial when multicast group membership is likely to change frequently. SLIMCAST ensures that *on-demand* joins and leaves are fully and efficiently supported.

A. Joins for Newly Deployed Nodes

For new nodes that wish to join a previously existing multicast group, these nodes first need to establish a path back to the base station, obtain information about existing multicast groups, and then send the appropriate Join Reply messages. We assume that a new node share keys with the base station. When a newly deployed node first boots up, it will send a neighbor request message out. The packet format is trivial and only has to include the sending node's ID. All of the neighbors will reply to the node, and the node can pick any of these neighbors as its physical parent. It then sends a GROUP QUERY message to the base station that looks like

$$GroupQuery + BaseID + NodeID +$$
$$\{Count + PropertiesList\}_{KB\_NodeID_a} + MAC(\text{packet})_{KB\_NodeID_b}$$

The $Count$ field is the number of properties that the node is sending to the base station. The $PropertiesList$ is used by the base station to determine which multicast groups the new node should join. We encrypt both of these fields to minimize the amount of information an attacker can gather about particular nodes in the network. The values in $PropertiesList$ will come from some predetermined list of possible properties, and might include data that indicates the node has a temperature sensor and a light sensor. In this case the $Count$ field would be 2. Upon receiving a GroupQuery packet, the base station replies with a QUERY RESPONSE packet that looks like

$$QueryResponse + BaseID + NodeID + MostRecentOWS$$
$$\{Count + GroupIDList\}_{KB\_NodeID_a} + MAC(\text{packet})_{KB\_NodeID_b}$$

The most recent OWS value is sent with this packet so that the node will be able to update its OWSold value and verify future Join Query packets. The $Count$ field is used to indicate the number of groups in the group ID list. The $GroupIDList$ will contain all groups that the node can actually join. As in the GroupQuery packet we encrypt this data. Once the node has decided which groups it wants to join, it will send one Join Reply message for each group, and the level key bootstrapping process will proceed normally.

In a dynamically changing multicast group, nodes may change their membership status in the multicast group at any

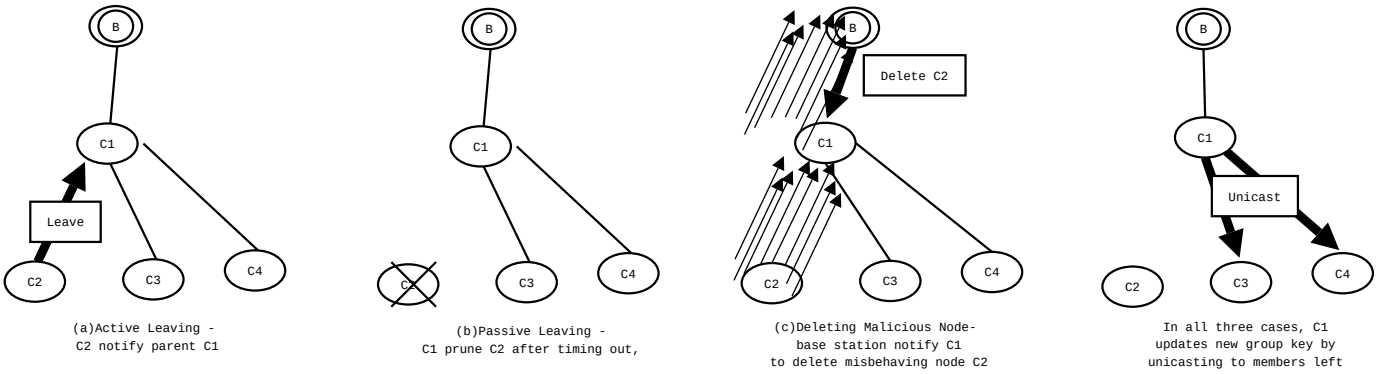

Fig. 5. Three different cases of Leaving and the corresponding update event

time. Nodes that are neither members nor forwarding nodes can easily join any group at a later time by simply sending a Join Reply as they normally would.

B. Node Leaving

Nodes can leave a group in a variety of ways. The most straightforward case is when a node wants to leave and it does not have any member children. Let's assume member node C2 in Figure 5 is now leaving the group. As the figure shows, there are three different ways a leave can occur.

Active Leaving: A node notifies its parent of leaving before it moves or runs out of battery power.

Passive Leaving: A node fails silently due to hardware failure or physical damage and does not notify its parent.

Deleting Malicious Node: A base station detects a misbehaving node and deletes it explicitly from the group. This is discussed in Section V.

In all three cases, the local parent node generates a new level key and unicasts it to its member children. After this update, the node that just left can no longer decrypt group messages and this guarantees forward secrecy for SLIMCAST. In the case of active leaving, if the leaving node has no children, it can simply leave the group without any other issues. However, when a member node that has children leaves the group, this leaves a gap in the level structure and physical tree path. To repair the tree structure, we want the leaving node's logical parent to become the parent of the leaving node's children, as shown in Figure 6. Once we bridge this gap, nothing should be effected above the leaving node's parent or below the leaving node's children.

Figure 6 illustrates the active leaving case. The leaving node first unicasts a message to the base station indicating that it is leaving. This informs the base station of the network's topology change. Next, the node unicasts a message to its logical parent notifying that it is leaving. This allows the parent to generate a new level key. Finally, if the node has children, it notifies them that it is leaving using the level key that they all share. This tells the children that they no longer have a parent and that they must rejoin the group. These children nodes then probe the neighbors and send a new Join Reply message through a new path to the base station as if they were joining the group for the first time. The tree structure thus is repaired.

In the case of passive leaving, the following approach is used to achieve fault tolerance. ACKnowledgements are used to confirm the existence of every link. A node with children will hear its children forward messages in a wireless broadcast medium, which will act as passive ACKs. If a node has no children, it will have to send an explicit ACK to its parent. If a parent doesn't hear from a faulty child node within a timeout period, k, which should be application dependent, it prunes that child and perform the same procedure as in the active leaving case. A blackmail attack could occur here, in which a compromised node reports to a base station that all its children are not responding and thus should be pruned. However, this is no worse than if the compromised node simply blocked all downstream nodes. If there are children nodes downstream of a faulty node, then SLIMCAST employs a simple solution that requires these children to wait until the next Join Query from the base station to rejoin the tree. A variety of other strategies were considered. For example, a child could timeout after absence of data from the parent and initiate a probe of neighbors plus a Join Reply. However, absence of data from a parent is not a sufficient indicator that a parent has failed, because downstream data could be communicated sporadically. A parent could transmit a heartbeat whose absence could be used as a definitive trigger, but this adds overhead. Other strategies also had limitations, so SLIMCAST chose a simple and lightweight solution of having children wait until the next Join Query. This Join Query flood is inherently fault-tolerant, and will build new routes around the faulty node.

V. Security Analysis

Our analysis of the security of the Join Query broadcast was completed in Section III-A, where we described how the OWS effectively prevented Join Query flooding and replay attack from adversaries. Hence, we focus on the security of the Join Reply packets in this section. A discussion of the security of Confirm packets is omitted due to lack of space, though we mention that spoofed Confirm packets are limited in the damage they can cause, primarily being confined to unicast downstream nodes.

A primary concern is to limit Join Reply packets from flooding the base station. This is accomplished by implementing rate control as the SLIMCAST tree is constructed.

Nodes keep a sliding window average of the number of Join Reply packets they have seen within some time window. If this number exceeds a threshold, then further Join Reply's are not forwarded until the sliding window average falls well below the threshold.

If over time the base station notices that an abnormally high flood of Join Replies is emanating from a particular branch in the tree, then the base station can choose to actively block Replies from that area. We borrow and modify the idea of nested MAC's [8] to enable the base station to identify where bogus Join Replies are emanating from. Recall that each Join Reply must have a MAC. As Join Replies propagate up the tree, each node will recompute the Join Reply's MAC using its own keyed MAC function computed over the current packet, including the current MAC. The new MAC is then appended to the Join Reply, along with the current node's ID. When the packet reaches the base station, the base station can identify the last node in the chain of nested MACs that computed correctly. No node can tamper with any part of the path without the base station's detection. Although nested MACs add some overhead to the Join Reply, the nested MAC serves two purposes. First it gives the base station full knowledge of the network's topology. Combining knowledge of existing multicast group members with the nested MACs, the base station can identify every node's physical parent and logical parent. Additionally, nested MACs provide us with the capability of pinpointing intrusion detection.

After exceeding a certain threshold of bad MACs, the base station will react by deleting a node from the topology. When there are N bad MACs immediately downstream of a malicious node, e.g. C5 in Figure 3, the base station should always assume the compromised node is C5. If we delete C5, it means that C3 will not forward any message whose last hop is C5, but C5 could still make up phantom sibling nodes and send the packets upstream through C3, subsequently causing the base station to think that C3 is now compromised. Instead of dealing with this problem in two phases, we suggest deleting C3 when we think that C5 is compromised, although in some cases C3 might not really be compromised and we might sacrifice it. In most cases, immediately deleting C3 will save enough energy to make this potential mistake worthwhile. So in this case, the base station will send a packet to C1, asking it to delete C3 and not forward any packet from C3 upstream. This will totally stop the possibility of C5 flooding the tunnel upstream to the base station. Also when we send out the deletion packet, we need to include all the phantom nodes and the deleted node in the deletion packet. This is because for every phantom node that C5 created, C1 will have an associated entry in its routing table. The deletion packet will be unicast from the base station to each node upstream of the deleted node and encrypted by the pairwise key shared between the sensor node and the base station. By reading the black list in the deletion packet, a node can clean up its routing table appropriately.

Another reason we need nested MACs is if we do not require every single node on the path to attach their MAC, any intermediate node can tamper with the path and redirect the path wherever it chooses. F2 can claim that the packet went through F3 and F4 and then forward it the F1, then when the base station receives it the WSN's topology saved in base station will be wrong.

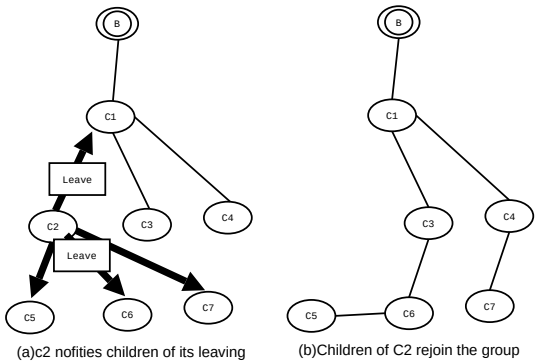

Fig. 6. An example of active leaving when there are children and repairing the tree.

In short, most nodes in WSNs running SLIMCAST will easily survive during DoS attack, and only the specific branch downstream of the malicious node will be sacrificed, but even these nodes can still preserve most of their energy in the hope that the malicious node will be detected and removed from the group so that they can rejoin the group.

VI. Simulation and Performance

We have simulated SLIMCAST in ns-2.26 on a AMD P2400+ box running SuSE Linux. For the joining overhead, we compare our protocol to the traditional star protocol and to LKH. While there several optimized versions of LKH, listed in Section VII, their key update event bandwidths are multiples of $logN$, and their differences are small in comparison to SLIMCAST's much lower key update event bandwidth. We thus choose to implement only LKH because its behavior is representive of the LKH family. The Star protocol is simply a logical topology where all nodes are logically viewed to be one hop from the base station and all the key update events are by unicast. During a join, LKH behaves exactly the same as the Star protocol, i.e. they unicast a key to the new member and update all existing members. Iolus has too many factors that make it hard to define the joining overhead such as the overhead of setting up secure channels between cluster heads, how they are synchronized, etc., so we only include comparisons to Iolus for leaving events. We use two topologies for our simulation - grid sparse and grid dense. In sparse mode each sensor node can hear four other neighbors, while in dense mode we double the range of sensor radio so each sensor should be able to hear on average 12 neighbors. In our simulations, nodes are randomly chosen to join the group. In each case we run the simulation ten times and our result is the average. Our simulation analyzes the overhead of SLIMCAST in two aspects: total *packets sent* for (1)Member Joining and (2)Member Leaving events.

A. Join Overhead

In the left Figure 7, the ratio of member nodes is fixed at 50%. The x-axis stands for total number of nodes and the y-

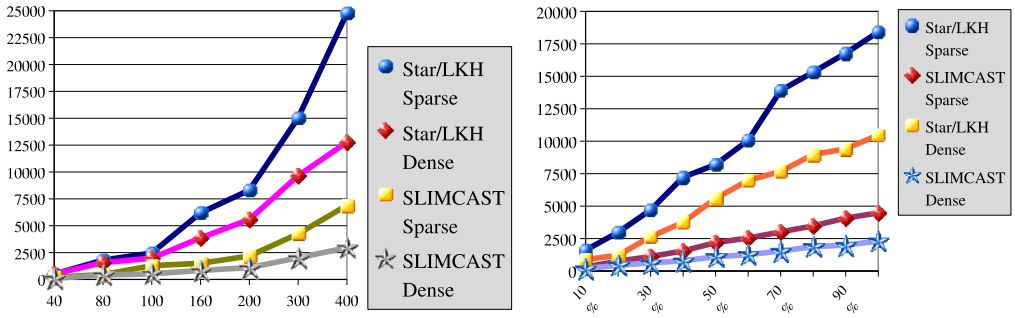

Fig. 7. Left: Total packets sent when 50% of nodes are assigned as member in network size from 40 to 400, Right: Total packets sent when different ratio of member nodes join 200 nodes group

axis stands for the number of total packets sent to set up the secure multicast group from the first member sending out a reply until the last joining member gets its level key and the local update is finished. We did not take the number of Join Query's sent into account here because this part, which floods the network to gather interest in group membership, is identical for all sender-initiated group communication protocols.

As the size of the network grows, the number of level members in SLIMCAST will not change. The number that will change is the average hop count from a newly joined node to the base station, and this only affects Star, Iolus and LKH dramatically. As mentioned earlier, the LKH behavior is identical to Star during joins, so only Star performance is plotted. In Star, in addition to increasing average hop count, we also have to count the increased number of multicast packets sent for group key update events to achieve backward secrecy. This is why the number of packets sent in Star grows much more rapidly than SLIMCAST. In the same figure we can see that both SLIMCAST and Star perform better in dense mode than in sparse mode, and this is because the hop count to the base station will be smaller in dense mode than in sparse mode. In dense mode, a node can be reach in about half the hop count of sparse mode.

In the Figure 7 on the right, we fix the number of total nodes to 200, and we show along the x-axis the percent of nodes that have joined the network (the joining nodes are randomly picked). Since the size of network is fixed and we pick member nodes randomly, the hop count will be approximately the same at each point on the x-axis. The only factor that changes is the number of members notified, so the packets sent grow linearly in SLIMCAST as a percentage of the multicast members. We also observe significant packet collision in the Star protocol when more than 50% of the nodes are members, which we attribute to frequent collisions with Join Replies going upstream. The packet count for Star doesn't grow perfectly as a linear or exponential function because of the random collisions and retransmissions.

B. Leave Overhead

In the simulation for leaving events, we pick one random member node and delete it from the group. As before, we ran the simulation ten times for each different member ratio and computed the average values. The topology is grid sparse and we use Iolus and LKH for comparison. Also notice that we use a logarithmic scale for the y-axis. In the left Figure 8 Star has to notify each member by unicast so packets sent grows linearly as member number increases. Iolus also uses unicast to update keys so it should perform identically to Star except that it is divided into five clusters, so the hop count and members to be notified will both be 1/5 of Star. LKH sends out one big message for leaving events, and this message has to reach all member nodes, so the number of packets sent is actually identical to one multicast event. While leaving update packets have the same size for Star, Iolus, and SLIMCAST, it is a lot bigger for LKH. LKH needs to send a O(log n) long packet for the leaving update event in a n member group. If group key is 128-bits long, for a 200 member node group, LKH will have to send 2048 bits. Compared with 128 bits for all three other protocols, LKH is sending a packet 16 times larger than the others. In the right Figure 8, we fix the ratio of member nodes to 50% and change the size of WSNs, so two variables will change in each of these simulations, the number of members and the hop counts. For SLIMCAST, the number of nodes to be updated only changes with the density of the WSNs, not with the number of members, so the increase will only grows linearly with hop count. For LKH, because the multicast characteristic, the increase in packets sent is close to linear growth as well. In Star and Iolus, hop count and member number multiply together to form an exponential growth in the number of packets sent.

C. Energy Consumption and Threshold

An overhead SLIMCAST has to pay is for hop-by-hop reencryption. Since we claim that SLIMCAST saves tremendous energy for key update events, we are interested in if it is worth of it for doing hop-by-hop reencryption rather than spending tremendous energy for key update events and have a global key, no reencryption like Star and LKH. Note that the overhead of Star and LKH occurs during the setup phase and is not taken into account here. In this section we compare the energy cost for one leaving key update event of Star and LKH with energy cost for reencrypting a multicast message in SLIMCAST. The sensor node bandwidth is assumed to be 19200bps, key size 128bits, encryption scheme is AES, hardware platform is Mica2 with 7.3MHz CPU.

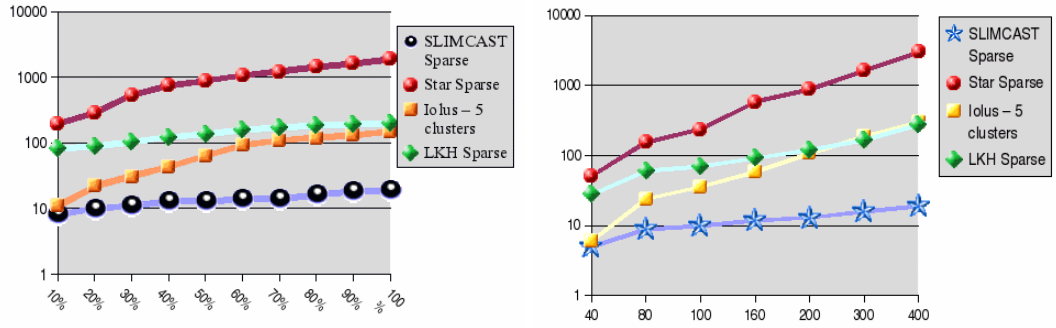

Fig. 8. Left: Total packets sent when one member leaves the group in different ratio of member in a size 200 sensor nodes group. Right: Total packets sent when one member leaves the group in different WSNs size when member ratio fixed at 50%

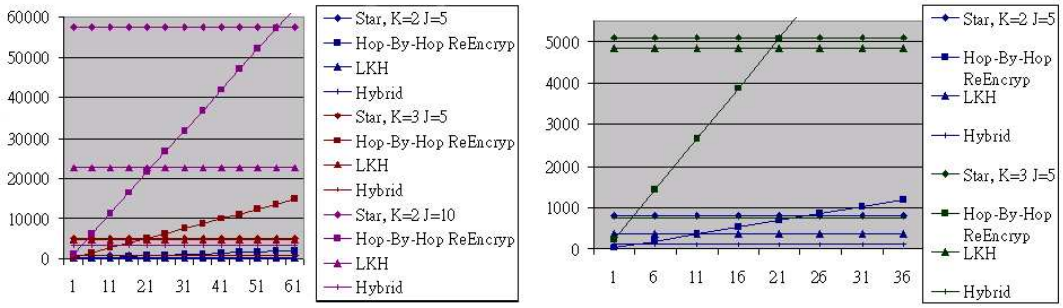

Fig. 9. Energy Comparison for SLIMCAST hop-by-hop Encryption and One Leaving Key Update Event

In the rest of this section we will try to find out the *number of multicast packets SLIMCAST can send until the hop-by-hop reencryption energy cost overhead hits the energy overhead LKH and Star have over SLIMCAST for one leaving event* in the same network topology. The total number of transmissions when unicasting to every single node in a network where each node has k children, and there is a total of j hops is $NumUnicast(k,j) = \sum_{i=1}^{j} ik^i$. This will be the transmission time needed by Star for one leaving key update event. The transmission times in one multicast event, i.e. for LKH key update even, is $NumUnicast(k,j) = \sum_{i=1}^{j-1} k^i = k^j - 1$. The number of encryption computations that SLIMCAST requires for hop-by-hop reencryption is $NumReEnc(k,j) = \sum_{i=1}^{j-1} k^i = k^j - 1$. In the Hybrid scheme, the number of reencryptions will be identical to number in level multicast, so the energy cost of Hybrid for one key update event will be $(k^j - 1)(Energyf orTransmissionnbytes + Energyf orReEncryptingnbytes)$.

The transmission power consumption for the Mica2 is 8.5mAh. Assuming a bandwidth of 19200 bps, transmitting 50 bytes will take 0.02083 seconds and will consume 0.000049 mA. For computation power, the CPU on the Mica2 is 7.3MHz ATmega128L with active mode current of 8.0mAh, so encrypting 50 bytes requires 51877 CPU cycles. Thus the encryption will require 0.0071 secs and 0.000016 mA[28]. That gives us the ratio of (Power for Tx n bytes)/(Power for Encrypt n bytes) = 3.12.

We then can then calculate the threshhold given that each node has an average of k children and hop counts of j for the SLIMCAST hop-by-hop reencryption scheme and Star. The equation below indicates the number of multicasts SLIMCAST can do before it reaches the threshold of energy spent for one key update event in Star

$$ThresholdMcastNumStar(k,j) = (\sum_{i=1}^{j} ik^i) * 3.12/k^j - 1$$

When k=2 and j=5 this number will be 26. For LKH, the packet multicasted for key update event for leaving will have $log_2(k^{j+1} - 1) * (KeyLength) * 2$ extra bytes than a key update packet for Star. So the equation to calculate SLIMCAST's threshold with LKH will be

$$ThresholdMcastNumLKH(k,j) = (((k^j - 1) * \log_2(k^{j+1} - 1) * (KeyLength) * 2/50) + 1) * 3.12/k^j - 1$$

When k=2 and j=5 this number will be 9. If, however, the Hybrid scheme is used, there will be no limitation on the multicast number, and the energy cost for one key update event is much lower than Star and LKH. The threshold for SLIMCAST and SLIMCAST-Hybrid is always 3.12, and SLIMCAST-Hybrid is always cheaper updating the global key than Star and LKH regardless of the network topology.

D. Future Work

As part of future work, we would like to study the performance under massive evictions for SLIMCAST and other protocols. Here, we will need to consider many variables, i.e. how is the batching executed and with what frequency. The LKH family may benefit more from batching key update events than SLIMCAST, but we can batch local key update events in SLIMCAST as well.

VII. Related Work

Canetti et al. proposed LKH+[4]. LKH+ halves the size of the key update message of LKH. One-way Function Tree

protocol(OFT)[3], [21] proposed by Balenson et al. derives parent key from two children keys, also halves the size of LKH. ELK[25] proposed by A Perrig et al. is a variant of OFT and uses hints to provide FEC. A. Eskicioglu. provides a complete survey of recent progress on secure multicast protocols[10]. For multicast routing over an Ad Hoc network, sender initiated ODMRP by S.J. Lee[16] inspired the QUERY-REPLY design in our protocol. J. Jetcheva and D. Johnson designed ADMR which is receiver initiated and provide a good method for on-demand join. VLM2 by A. Sheth et. al[27] is one of the earliest attempts for multicast over sensor networks.

R. Canetti and B. Pinkas describe requirements of secure multicast protocols for key updating events in membership changes[5], and more evaluation strategy can be found in [22] by S. Mishra. Performance comparisons of multicast protocols over Ad Hoc wireless network is provided by S.J Lee et. al[17]. A. Wood and J. Stankovic list various DoS attacks against wireless sensor networks in [29]. Works of group communication, namely content-based multicast, or content-based routing under wireless sensor networks can be found in CBM by H. Zhou and S. Singh[31] and Directed Diffusion by C. Intanagonwiwat, R. Govindan and D. Estrin[14].

There are also attempts to secure group communication in the form of content based routing for wireless sensor networks by B. Przydatek, D. Song and A. Perrig[26] and Secure Aggregation for Wireless Networks by L. Hu and D. Evans[13]. In [9], [6], authors proposed a scheme that does not need a base station to construct and manage level keys. The advantage is that the joining event will not have to travel all the way back to base station, but the disadvantage is that a reverse path and downstream routing table cannot be set up, and the trust between nodes is only level by level, which seems to be less secure than centralized authentication.

VIII. Conclusion

We have described SLIMCAST, the design and implementation of a level key infrastructure for secure and efficient group communication in wireless sensor networks. SLIMCAST employs level keys to delegate trust throughout each level and each branch of the WSN multicast tree. This delegation of trust localizes the overhead cost of key updates due to joins and leaves resulting in highly energy-efficient key updates. SLIMCAST provides a complete security solution from the setup phase to key update events. SLIMCAST protects data confidentiality via hop-by-hop encryption based on level keys and also tolerates the compromise of parent nodes. SLIMCAST further implements intrusion detection and deletion to limit the damage from DoS-based flooding attacks. Finally, the SLIMCAST infrastructure supports data aggregation to efficiently send gathered data back to the base station. Our simulations of SLIMCAST in NS2 demonstrate that SLIMCAST achieves dramatically lower overhead than traditional secure multicast protocols.

References

[1] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, and R. Han. Mantis: System support for multimodal networks of in-situ sensors. *WSNA*, 2003.

[2] U. A. F. ARGUS Advanced Remote Ground Unattended Sensor Systems, Department of Defense. Argus.

[3] D. Balenson, D. McGrew, and A. Sherman. Key management for large dynamic groups: One-way function trees and amortized initialization. *Internet Draft, Internet Engineering Task Force*, 1999.

[4] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and efficient constructions. *Proc. of INFOCOM'99*, 1999.

[5] R. Canetti and B. Pinkas. A taxonomy of multicast security issues. *Internet Draft, draft-irtf-smug-taxonomy-01.txt*, 2000.

[6] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. *IEEE Symposium on Research in Security and Privacy*, 2003.

[7] J. Deng, R. Han, and S. Mishra. A performance evaluation of intrusion-tolerant routing in wireless sensor networks. *Proc. of IPSN03*, 2003.

[8] J. Deng, R. Han, and S. Mishra. Insens: Intrusion-tolerant routing for wireless sensor networks. *Elsevier Journal on Computer Communications, Special Issue on Dependable Wireless Sensor Networks*, 2005, to appear.

[9] L. Eschenauer and V. Gligor. A key-management scheme for distributed sensor networks. *CCS*, 2002.

[10] A. Eskicioglu. Multimedia security in group communications: Recent progress in wired and wireless networks. *ACM Multimedia Systems Journal*, 2003.

[11] H. Harney and E. Harder. Logical key hierarchy protocol. *Internet Draft, draft-harney-sparta-lkhp-sec-00.txt*, 1999.

[12] H. Harney and C. Muckenhirn. Group key management protocol(gkmp) architecture. *RFC 2094*, 1997.

[13] L. Hu and D. Evans. Secure aggregation for wireless networks. *SAINT'03*, 2003.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion. *Proc. MobiCom'00*, 2000.

[15] B. Krishnamachari, D. Estrin, and S. Wicker. Impact of data aggregation in wireless sensor networks. *DEBS'02*, 2002.

[16] S.-J. Lee, W. Su, and M. Gerla. On-demand multicast routing protocol in multihop wireless mobile networks. *ACM/Kluwer Mobile Networks and Applications*, 2000.

[17] S.-J. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia. A performance comparison study of ad hoc wireless multicast protocols. *Proc. of IEEE INFOCOM*, 2000.

[18] J. Liu, J. Liu, J. Reich, P. Cheung, and F. Zhao. Distributed group management for track initiation and maintenance in target localization applications. *IEEE IPSN*, 2003.

[19] A. Mainwaring, J. Polastre, R. S. D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. *WSNA*, 2002.

[20] D. Malan, M. Welsh, and M. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. *IEEE SECON*, 2004.

[21] D. A. McGrew and A. T. Sherman. Key management for large dynamic groups using one-way function trees. *http://www.cs.umbc.edu/ sherman/Papers/itse.ps*, 1998.

[22] S. Mishra. Key management in large group multicast. *Technical Report CU-CS-940-02, Univ. of Colorado*, 2002.

[23] S. Mittra. Iolus: A framework for scalable secure multicasting. *Proc. of the ACM SIGCOMM*, Sept. 1997.

[24] A. Perrig. Spins: Security protocols for sensor networks. *Proc. of MOBICOM*, 2001.

[25] A. Perrig, D. Song, and J. Tygar. Elk, a new protocol for efficient large-group key distribution. *IEEE Symposium on Security and Privacy*, 2001.

[26] B. Przydatek, D. Song, and A. Perrig. Sia: Secure information aggregation in sensor networks. *ACM SenSys'03*, 2003.

[27] A. Sheth, B. Shucker, and R. Han. Vlm2: A very lightweight mobile multicast system for wireless sensor networks. *IEEE WCNC 2003*, 2003.

[28] V. Shnayder, M. Hempstead, B. rong Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. *SenSys '04*, 2004.

[29] A. Wood and J. Stankovic. Denial of service in sensor networks. *IEEE Computer , Vol. 35, No. 10, pp. 54-62.*, 2002.

[30] F. Ye, H. Luo, S. Lu, and L. Zhang. Statistical en-route filtering of injected false data in sensor networks. *IEEE INFOCOM'04*, 2004.

[31] H. Zhou and S. Singh. Content based multicast (cbm) in ad hoc networks. *Proc. of ACM MobiHoc'00*, 2000.

[32] S. Zhu, S. Setia, and S. Jajodia. Leap: Efficient security mechanisms for large-scale distributed sensor networks. *Proc. of ACM CCS '03*, 2003.