SECURE DATA AGGREGATION SCHEME

FOR SENSOR NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Kavit Shah

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Electronics Engineering

December 2014

Purdue University

Indianapolis, Indiana

This is the dedication.

ACKNOWLEDGMENTS

This is the acknowledgments.

PREFACE

This is the preface.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

SYMBOLS

m	mass
v	velocity

ABBREVIATIONS

abbr	abbreviation
bcf	billion cubic feet
BMOC	big man on campus

NOMENCLATURE

Alanine	2-Aminopropanoic acid
Valine	2-Amino-3-methylbutanoic acid

GLOSSARY

chick	female, usually young
dude	male, usually young

ABSTRACT

Shah, Kavit Master, Purdue University, December 2014. Secure data aggregation scheme for sensor networks. Major Professor: Dr. Brian King.

This is the abstract.

1. INTRODUCTION

TIPS: USE ACTIVE VOICE

USE VERBS

DON'T TURN VERBS INTO NOUNS

COMMON MISTAKE: DATA ARE ; DATA IS PLURAL THAT/WHICH

Advancements in compute, storage, networks and sensors technologies have led to many new promising applications.

1.1 Sensor Networks

The sensor networks of the near future are envisioned to consist of hundreds to thousands of inexpensive wireless sensor nodes, each with some computational power and sensing capability, operating in an unattended mode. They are intended for a broad range of environmental sensing applications from vehicle tracking to habitat monitoring. Give an example and talk about energy, security constraints.

1.2 Internet Of Things

In the world of mass connectivity people need to get information all the time on an array of devices. Everything from your refrigerator to your thermostat is connected to wireless networks and joining the "internet of things". Write about bandwidth constraints.

1.3 Big Data

All the large internet companies process massive amounts of data also know as "Big Data" in real time applications. These include batch-oriented jobs such as

data mining, building search indices, log collection, log analysis, real time stream processing, web search and advertisement selection on big data. To achieve high scalability, these applications distributes large input data set over many servers. Each server process its share of the data, and generates local intermediate. The set of intermediate results contained on all the servers is then aggregated to generate the final result. Often the intermediate data is large so it is divided across multiple servers which perform aggregation on a subset of the data to generate the final result. If there are N servers in the cluster, then using all N servers to perform the aggregation provides the highest parallelism. Talk about compute constraints. [**?**]

Airplanes are also a great example of "big data". In a new Boeing Co.747, almost every part of the plane is connected to the Internet, recording and sometimes sending continuous streams of data about its status. According to General Electric Co. in a single flight one of its jet engines generates half a tera bytes of data. This shows that we have too much of data and we are just getting started.

1.4 Data Aggregation

Data aggregation is an important technique used in many system architectures. The key idea is to combine the data coming from different sources eliminating the data redundancy, minimizing the number of packet transmissions thus saving energy, bandwidth and memory usage. This technique allows us to focus more on data centric approaches for networking rather than address centric approaches. [**?**]

1.5 Cloud Computing

1.6 Fog Computing

2. PROBLEM STATMENT AND ASSUMPTIONS

2.1 Assumptions

1. Trusted Querier 2. Leaves can report only one value at a time. 3. pre-knowledge of network topology

3. SECURE DATA AGGREGATION SCHEME

The goal of this thesis is to examine secure data aggregation schemes for various distributed systems.

Many modern world system designs are distributed in nature. The system design includes small, individual components doing their tasks precisely and lots of these components synchronize with all other components to complete the bigger task.

Many applications of sensor network are inharenlty distributed in nature. For example, scientific data collection, building health monitoring , building safety monitoring systems are distributed systems. Write an example how data aggregation happes in one particular application. [**?**]

The application design architecture for the internet of things is distributed as well. Write an example how data aggregation happens in one particular application. [**?**]

3.1 Network topology

Write about how all theses distributed systems can be classifieds into general tree structure.

Subsubsection heading

This is a sentence. This is a sentence.

4. COMMITMENT TREE GENERATION

Theorem 4.0.1 *Binary commitment tree is optimal for terms of verification as it requires minimum number of off-path values.*

Proof Let us say

$\log_3(n) = y$

$3^y = n$

$\log_2(3^y) = \log_2(n)$

$y * \log_2(3) = \log_2(n)$

$\log_3(n) * \log_2(3) = \log_2(n)$

$\log_3(n) = \frac{\log_2(n)}{\log_2(3)}$

$2 * \log_3(n) = [2/\log_2(3)] * log_2(n) = (1.2618) * log_2(n)$

$2 * log_3(n) > log_2(n)$

■

5. VERIFICATION OF CONFIRMATIONS

5.1 Network Model

We assume a multihop network with a set $S = \{s1, ..., s_n\}$ of n sensor nodes. The network is organized in a tree topology, with the base station as the root of the tree. The trusted querier resides outside of the network & has more computation, storage capacity then the sensor nodes in the network. The querier knows total number of sensor nodes n and that all n nodes are alive and reachable. The querier also knows the network topology. All the wireless comuunication is peer-to-peer and we do not consider local wireless broadcast. We also assume that the querier has the capacity to do peer to peer communication with every sensor node in the network. We also assume that all the sensor nodes mapped to the vertices in the commitment tree who reported the authentication codes with NACK messages during the collection of confirmations step are honest.

5.1.1 Sensor Nodes

We assume that each sensor node has a unique identifier s and shares a unique secret symmetric key K_s with the querier. We assume all the sensor nodes are capable of doing symmetric- key encryption and symmetric key decryption. They are also capable of computing collision-resistant cryptographic hash function H. We also assume that if the sensor node has to send multiple messages to its parent it can combine those messages into single packet in a way that parent node can distinguish both the messages.

5.1.2 Commitment trees

The aggregation tree is a physical network over which all the communication happens while the commitment tree is a logical network on top of aggregation tree. In the similar manner, vertices in a commitment tree is a logical element in a graph while a sensor node is a physical device.

The commitment trees have the following properties. At most there will be $\lceil lgn \rceil$ commitment trees in the forest for an aggregation tree with n sensor nodes. The binary representation of the number of sensor nodes n in the network indicates the number of commitment trees in the forest. It also indicates the height of all the commitment trees in the forest. For instance, if there are $11_{10} = 1011_2$ sensor nodes in the aggregation tree then there will be three commitment trees in the forest. From those three commitment trees, one will be of height three, one will be of height one and one will be of height zero. At most there will be $n + (n - 1)$ vertices in the forest of commitment trees for an aggregation tree with n sensor nodes.

It is also possible that only one sensor node s in the aggregation tree, will be the root vertex of all the commitment trees in the forest.

It is also poosible that all the internal vetices in the commitment tree are of different sensor nodes in the aggregation tree.

5.1.3 Collections of confirmations

After each sensor node s has successfully performed the verification step for its leaf vertex u_s, it sends an authentication code to its parent in the aggregation tree. The authentication code for sensor node s is $\mathrm{MAC}_{K_s}(N||ACK)$ where ACK is an acknowledgemnet message, N is the query nounce and K_s is the secret key that sensor node s shares with the trusted querier. Once an internal sensor node s in the aggregation tree has received the authentication codes from all of its descendants, it computes the XOR of its own authentication code with all the received authentication codes, and forwards the result to its parent. Finally, the querier will receive a

single authentication code from the base station that consists of the XOR of all the authentication codes received in the network.

5.1.4 Verification of confirmations

Since the querier knows the secret key K_s for each sensor node s in the aggregation tree and it also knows the topology of the commitment trees in the forest, it can simulate the commitment trees in the forest. The querier simulates the commitment trees in the forest by computing the following authentication codes

$$\text{MAC}_{K_1}(N||ACK) \oplus \text{MAC}_{K_2}(N||ACK) \oplus \ ... \ \oplus \text{MAC}_{K_n}(N||ACK) \ ;$$
$$\text{MAC}_{K_1}(N||NACK) \oplus \text{MAC}_{K_2}(N||NACK) \oplus \ ... \ \oplus \text{MAC}_{K_n}(N||NACK)$$

and creates two simulated commitment trees, one with the authentication codes of ACK messages and one with the authentication codes of NACK messages; where ACK is an acknowledgemnet message, NACK is a negative acknowledgemnet message & N is the query nounce. Then the querier merges all the commitment trees in the forest simulated with the authentication codes of ACK messages, by taking XOR of the root of all the commitment trees in the forest and calculates a single root authentication code for the forest simulated with the authentication codes of ACK messages. The querier does the same procedure for the commitment trees' forest simulated with the authentication codes of NACK messages. The querier stores all the simulated commitment trees and root authentication codes in the memory.

The querier receives a signle root authentication code from the base station during the collection of confirmation messages step. The querier compares the received root authentication code with the relevant simulated authentication code of ACK messages. If those two codes match it means every vertices in the subtree rooted at that vertex sent the authentication codes with ACK message during collection of

confirmation step. If those two codes do not match it means one or more vertices in the subtree rooted at that vertex sent the authentication codes with NACK message during collection of confirmation step. In the case where root authentication codes do not match, the querier proceeds futher to find out which vertex or vertices in the network sent the authentication codes with NACK message during collection of confirmation messages step.

To find out which vertex or vertices in the commitment tree reported the authentication codes with NACK message, the querier asks the base station to send the authentication codes of all the commitment trees' root in the forest. After receiving the authentication codes from the base station, the querier compares those authentication codes with the relevant simulated authentication codes of ACK messages. After this comparision, whose authentication codes do not match, the querier classifies those commitment trees as BAD TREES in the forest.

Then the querier compares the authentication codes of those BAD TREES with the relevant simulated authentication codes of NACK messages. If any of those authentication codes match it means all the vertices rooted in that subtree sent the authentication codes with NACK message during the collection of confirmation stpe. And the querier stops doing the quiers to the subtree rooted at that vertex. If those authentication codes do not match then the querier proceeds furhter to find the senosr node or nodes who reported the authentication codes with NACK message in BAD TREES.

5.1.5 Algorithm for detecting vertex or vertices who reported authenctication codes with NACK messages

For each BAD TREE in the forest the querier does the following :

1. The querier asks the relevant nodes in the aggregation tree to send the authentication codes of their children in the commitment trees.

2. The querier compares the received authentication codes in step 1 with the authentication codes of the relavant vertices of the commitment trees simulated with the authentication codes of ACK messages. The querier classifies the subtree rooted at those vertices as BAD SUBTREES, whose authentication codes do not match.

3. The querier compares the received authentication codes in step 1 with the authentication codes of the relavant vertices of the commitment trees simulated with the authentication codes of NACK messages. If those codes match it means all the vertices rooted in that subtree reported NACK messages during collection of confirmation steps. And the querier stops doing the queries to that subtree. If those codes do not match then the querier goes to step 2.

   The querier follows this algorithm until it finds all the vertices who reported the authentication codes with NACK messages in the commitment trees during collection of confirmation step.

5.1.6 Analysis

To simulate the commitment trees in the forest the querier has to do the following calculations. Suppose there are n sensor nodes in the network, means there will be at most $\lceil lgn \rceil$ commitment trees in the forest. If there are $n^{'}$ leaves in each commitment tree in the forest then the height($h^{'}$) of each commitment tree will be $lg(n^{'})$ and the number of intermediate vertices ($i^{'}$) in the network will be $(2^{h^{'}} - 1)$. So, there will be total of $(n^{'} + i^{'})$ * $\lceil lgn \rceil$ vertices in the forest. As the querier needs to simulate two commitment trees one for ACK messages and one for NACK messages it has to do ($2 * (n^{'} + i^{'}) * \lceil lgn \rceil$) calculations.

As we know from the properties of commitment trees, at most there will be $n + (n-1)$ vertices in the forest of commitment trees for an aggregation tree with n sensor nodes. So, the querier has to do $2 * (n + (n-1))$ calculations and it also need that much memory space to store those values.

With Dr.King with a single NACK message in a tree:

Maximum number of leaves in a single commitment tree is equal to n. So, we can upper bound it with $O(n)$. As there is a signle NACK message in the tree, during verification phase we know in which part that vertex is.

So, we can identify the node in at most $\log n$ steps. We need $2 * \log n$ messages to identify that. The querir will have to ask $\log n$ times and some node from the aggregation time has to reply $\log n$ times. This is total number of messages is required but number of communication depends on the aggregation tree.

Worst case: In the worst case, the querier has to ask values of all the vertices in the commitment tree. At max there will be $(2 * n - 1)$ messages. Again, remember these are the number of messages, number of communications might be different.

Best case in terms of number of communications: It needs minimum number of communications when one single node is everywhere in the aggregation tree. In that case if the querier has to know the values of all the leaves vertices it has to ask $\log(n)$ questions and node has to reply to those messages. So, number of communication is equal to $2 * log(n)$.

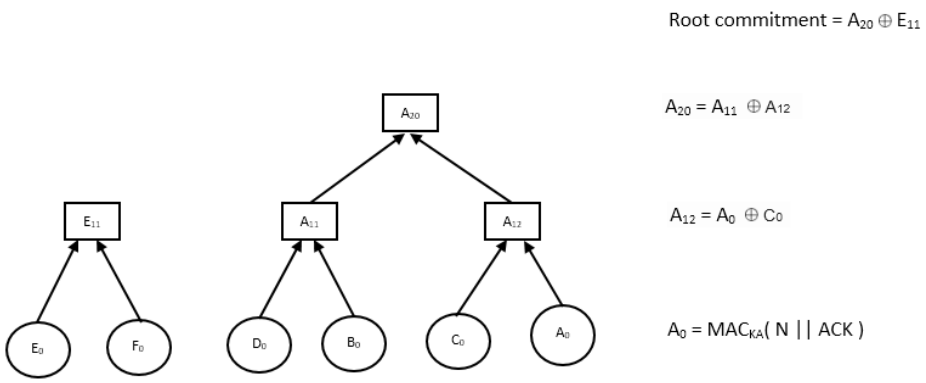

Fig. 5.1.: Simulated commitment tree with ACK messages

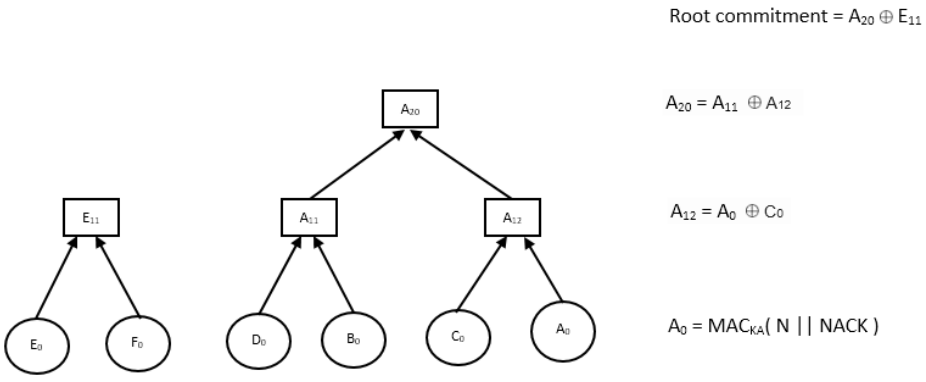

Fig. 5.2.: Simulated commitment tree with NACK messages

6. CHEATING

6.1 Definition

If an aggregator changes the sensor readings reported by its children to skew the final aggregated result is consider as cheating.

6.2 Aim

Aim of this section is to detect the cheater with given definition.

6.3 Assumptions

We make an assumption that the cheater can not say NACK during verification phase. If a cheater is allowed to send NACK message then it can send NACK messages all the time and create a lot of traffic in the network which might create Denial of service attack.

6.4 What is not cheating ?

In figure 7.1, A is an aggregator if A is a cheater it can skew the final aggregation result irrespective of B's sensor reading. We do not consider this case as a cheating because A is adjusting its sensor reading, it's not changing the B's sensor reading.

For example, if maximum allowed value = 10

case I: $B_0(2) = 5$, $A_0(2) = 13$, $A_1(2) = 18$. In verification, A will be caught due to out of range off path value.

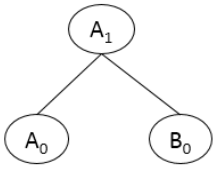

Fig. 6.1.: Possible commitment tree

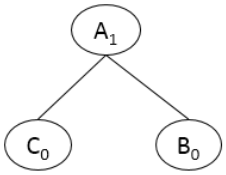

Fig. 6.2.: Possible commitment tree

case II: $B_0(2) = 5$, $A_0(2) = 10$, $A_1(2) = 15$. $B_0'(2) = 6$, $A_0'(2) = 9$. that's not cheating.

Similar arguments can be done for figure 7.2 if A, C both are cheaters. In that case A is adjusting C's sensor reading to skew the final aggregation result and C will not complain as it is a cheater. We do not consider that as cheating either.

6.5 Probabilistic bound on a cheater

To derive Probabilistic bound on a cheater using following example.

In figure 7.3, all vertices in a commitment tree are unique. And, remember cheater can not say NACK during verification phase.

- A_0 says NACK during verification phase it implies that atleast one of the following is {I}, {B, I}, {B, M} is a cheater.
- A_0, B_0 says NACK during verification phase it implies that atleast one of the following is {I}, {M}, {C, D, O} is a cheater.

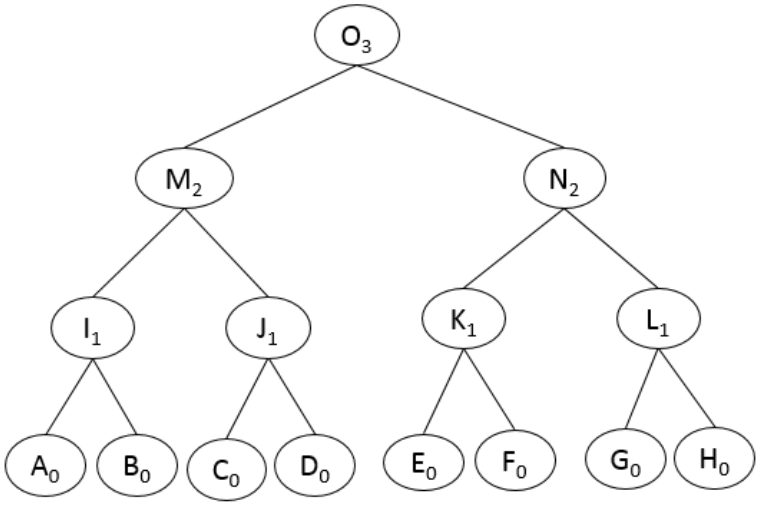

Fig. 6.3.: Possible commitment tree

- A_0, B_0, C_0 says NACK during verification phase it implies that atleast one of the following is {J ,I}, {J, M}, {D, O} is a cheater.
- A_0, B_0, C_0, D_0 says NACK during verification phase it implies that atleast one of the following is {O}, {M}, {I, J}, {E, F, G, H, O} is a cheater.
- A_0, B_0, C_0, D_0, E_0 says NACK during verification phase it implies that atleast one of the following is {O, K}, {M, K}, {I, J, K}, {F, G, H, O} is a cheater.
- $A_0, B_0, C_0, D_0, E_0, F_0$ says NACK during verification phase it implies that atleast one of the following is {I, J, K}, {M, N}, {O, K}, {O, N} is a cheater.
- A_0, C_0 says NACK during verification phase it implies that atleast one of the following is {I}, {J} is a cheater.

Similar, kind of analysis can be done for figure 7.4 in which all the vertices in the commitment tree are different.

From all above examples we can derive the following pattern as well,

If d = depth of a tree,

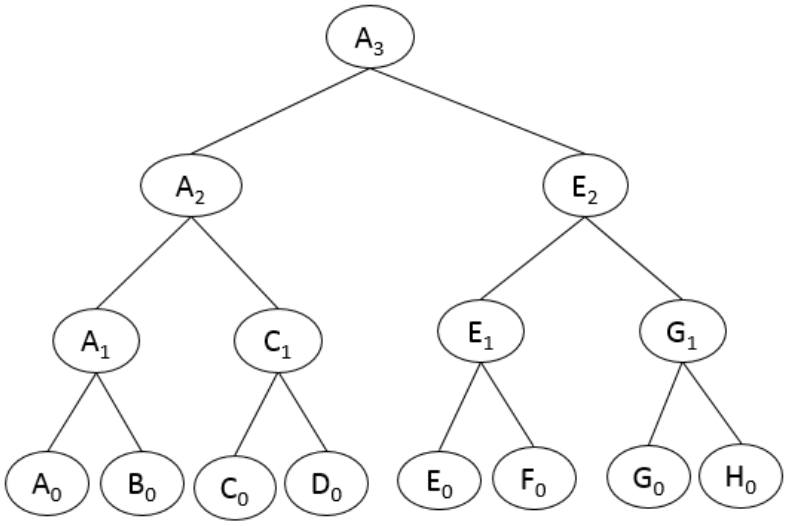

Fig. 6.4.: Possible commitment tree

Depth of a cheater	Minimum number of NACK messages
d - 1	1
d - 2	2
d - 3	4
d - 4	8

6.6 Why do we need digital signatures ?

Digital signatures allow us to achieve authenticity of the message. The labels and signatures have the following format:

id = id

label = $< count, value, commitment >$

signature = $E_{Private_{key}}(H(N||label))$

Where *count* is the number of leaf vertices in the subtree rooted at this vertex; *value* is the SUM aggregate computed over all the leaves in the subtree; *id* is the sum of all the leaves id in the subtree; *signature* is a cryptographic scheme for demonstrating the authenticity of a message; N is the query nonce.

There is one leaf vertex u_s for each sensor node s, which we call the leaf vertex of s. The label of u_s consists of count = 1, value = a_s where a_s is the data value of s, and signature is the node's unique ID.

Internal vertices represent aggregattion operations, and have labels that are defined based on their children. Write up examples after talking to Dr.King : Do you have to aggregate ID's as well ?

6.7 Why digital signatures are not sufficient to detect a cheater ? or Why do we need public key infrastructure to detect a cheater ?

Digital signatures allow us to achieve authenticity of the message but do not provide any mechanism to achieve integrity of the message. To achieve integrity we need public key infrastructure.

For example, in figure 7.3 one set of possible lables could be the following:

$id_A = 1; A_0 =< 1, 5, H(N||1||5) >; SigA_0 = E_{K_A}(H(N||A_0));$

$id_B = 2; B_0 =< 1, 6, H(N||1||5) >; SigB_0 = E_{K_B}(H(N||B_0));$

$id_I = 3; I_1 =< 2, 11, H(N||2||11||A_0||B_0) >; SigI_1 = E_{K_I}(H(N||I_1));$

$id_J = 4; J_1 =< 2, 15, H(N||2||15||C_0||D_0) >; SigJ_1 = E_{K_J}(H(N||J_1));$

$id_M = 5; M_2 =< 4, 26, H(N||4||26||I_1||J_1) >; SigM_2 = E_{K_M}(H(N||M_2));$

Above labels and signatures are the case where no one is cheating in the network. If A, B say NACK message during the verification phase it means either M or I is a cheater. To preceisely find who is cheater we have following problems:

- M can say it received $(I_1^{'}, SigI_1)$ eventhough it received $(I_1, SigI_1)$ from I.
- M can not verify that it received $(I_1^{'}, SigI_1)$ instead of $(I_1, SigI_1)$ from I.

Because of this we can not not detect cheater between I, M. The fundamental problem is that signatures can be verified only by the base station and not by any of the intermediate nodes. We want the ability in which an intermediate node can verify the signatures from its children. And that is why we need public key infrastructure.

7. AUGUST

Things discussed in meeting:

Analyzed congestion and why is it sub linear ?

In SHIA leaves verify their values with final results not with intermediate results. But in surveillance application data is compared with some base value in such network intermediate values are important.

Analyze the protocol with Digital signatures. How many signatures do we need ?

Analyze properties of commitment tree.

Definitions

A **direct data injection attack** occurs when an attacker modifies the data readings reported by the nodes under its direct control, under the constraint that only legal readings in [0, r] are reported.

An aggregation algorithm is **optimally secure** if, by tampering with the aggregation process, an adversary is unable to induce the querier to accept any aggregation result which is not already achievable by direct data injection.

For example, if A is an aggregator and it receives one reading from B. So, A needs to aggregate two values one of its own and the other is B's value. Suppose, maximum allowed value is 40. A0 = 10, B0 = 20. A1 = 30. A1 ¡= 80. If A reports any value out of that range it will get caught and any cheating within the range falls under direct data injection attack.

Congestion

As a metric for communication overhead, we consider node congestion, which is the worst case communication load on any single sensor node during the algorithm. Congestion is a commonly used metric in ad-hoc networks since it measures how quickly the heaviest-loaded nodes will exhaust their batteries [6, 12]. Since the heaviest-loaded nodes are typically the nodes which are most essential to the connec-

tivity of the network (e.g., the nodes closest to the base station), their failure may cause the network to partition even though other sensor nodes in the network may still have high battery levels. A lower communication load on the heaviest-loaded nodes is thus desirable even if the trade-off is a larger amount of communication in the network as a whole.

For a lower bound on congestion, consider an unsecured aggregation protocol where each node sends just a single message to its parent in the aggregation tree. This is the minimum number of messages that ensures that each sensor node contributes to the aggregation result. There is $\Omega(1)$ congestion on each edge on the aggregation tree, thus resulting in $\Omega(d)$ congestion on the node(s) with highest degree d in the aggregation tree. The parameter d is dependent on the shape of the given aggregation tree and can be as large as $\Theta(n)$ for a single-aggregator topology or as small as $\Theta(1)$ for a balanced aggregation tree. Since we are taking the aggregation tree topology as an input, we have no control over d. Hence, it is often more informative to consider per-edge congestion, which can be independent of the structure of the aggregation tree.

Consider the simplest solution where we omit aggregation altogether and simply send all data values (encrypted and authenticated) directly to the base station, which then forwards it to the querier. This provides perfect data integrity, but induces O(n) congestion at the nodes and edges nearest the base station. For an algorithm to be practical, it must cause only sublinear edge congestion.

Our goal is to design an optimally secure aggregation algorithm with only sublinear edge congestion.

1. remove complement 2. variable range

8. NOVEMBER

Misc. topics to write about:

Why do you want to communicate an entire aggregation tree to the querier ?

If the querier knows the entire aggregation tree and also if it knows the protocol which all the sensor nodes will be running then the querier can simulate the commitment trees on its own. Because of that we do not have to communicate the commitment tree every time we run the protocol which saves a lot of communications in the network. Also, note the fact that aggregation tree does not change often so the communication required to send the aggregation tree is negligible over time.

How to communicate an entire aggregation tree to the querier ?

The base station in the aggregation tree needs to know the entire network topology. It will relay that information to the querier.

How does the base station know the entire aggregation tree topology ?

If every sensor nodes has a small table containing the path to reach to the certain destination then the base station can ask for this information to the individual sensor nodes. While it is receiving this information it can relay the same information to the querier. Note: the base station is also a simple sensor node like all other nodes it can not store all the forwarding tables so it will relay those table information directly to the querier and querier can make big table containing the information related to the aggregation tree.

Mobility

You can talk about the aggregation tree topology is mobile. It's increasingly mobile topology not leap mobility.

Caching of certificates

Certificates are sent only once for the first time. They are cached for subsequent

communications. Every node in the tree needs to know the certificates of all the root nodes in its forest.

Why does the internal vertex in the commitment tree need to send what it received and what it sent to its parent ?

To detect a cheater, if an internal vertex send (to the querier) only the values which it sent to its parent in the commitment tree then it is no value to the querier. Because the querier can not verify that value and the signature. For the querier to verify the aggregated data and its signature it needs both the values over which aggregation has happend.

Why don't you need backward signatures ?

Because according to the protocol, every parent checks its children's message and its signature. If those two do not match then it will not accept the message.

Do you need signature on forest ?

No, we do not need the signature on forest.

Analyses of being root in as many tree as possible:

- *Bandwidth perspective*

  Off path values

  It takes same bandwidth (same hop counts) to distribute off path values in include itself or exclude itself stratergy. You can have inductive argument for it to prove it.

  Certificates Parent node needs to deal with less nodes in the aggregation tree means it needs less certificates, means less memory storage. For example, in pseudo palm tree case if we use include it self streategy then it is possible that one node has to propagate its value from the bottom to the top of the tree. It means all the intermediate nodes need to know its certificate. This can be avoided by using exclude itself(being root in as many possible tree as possible) stratergy.

- *Security perspective*

Exclude itself stratergy is more secure in the sense that aggregator needs to partner with two nodes to achieve cheating. If it includes itself then it has to partner with only one node which is relatively easy.

Why do we need authenticated broadcast from the querier ?

Significance of Nonce

Why do we need public key infrastructure ?

Why don't we use aggregation tree as commitment tree ?

Why is commitment tree binary and not n-ary ? (proof)

How to detect following cheating ?

The querier knows an aggregation tree and a protocol. So the querier can simulate commitment tree. All the nodes in the network are supposed to run the same protocol. Suppose if they don't then the commitment tree will look different. How will you detect such cheating ?

9. PROTOCOL

The commitment tree is a tree where each vertex has an associated label representing the data that is passed on to its parent. The messages have the following format:
MESSAGE

ID	COUNT	VALUE	COMMITMENT
20 bits	21 bits	20 bits	256 bits

SIGNATURE (MESSAGE)

$\text{Encryption}_{secret-key_{node}}$(HASH (MESSAGE))
500 bits

CERTIFICATES

Public key	Signature	ID
1000 bits	500 bits	20 bits

9.1 Aggregation-Commit Phase

In this phase, the sensor nodes construct a commitment forest. First, the sensor nodes at the highest depth in the aggregation tree (leaf nodes) send their messages to their parents in the aggregation tree. Each internal sensor node in the aggregation tree performs an aggregation operation whenever it receives messages from all of its children. Whenever a sensor node performs an aggregation operation, it creates a commitment to the set of inputs used to compute the aggregate by computing a hash over all the inputs (including the commitments that were computed by its children). Both the aggregation result and the commitment creates a message. Then the message, with the signature of the message signed by the sensor node are passed on to the parent of the sensor node. Once the final messages and the signatures of those messages are sent to the querier, if the adversary tries to claim a different aggregation structure it gets caught. Our algorithm generates perfectly balanced binary trees to create commitment forest which saves the bandwidth in the verification phase.

Definition 9.1.1 *A **commitment tree** is a logical tree build on top of an **aggregation tree** in which each vertex has an associated message to it, representing data being passed on to its parent. The messages have the following format:*

$$< id, count, value, commmitment >$$

Where id is the unique id of each vertex; count is the number of leaf vertices in the subtree rooted at this vertex; value is the aggregate computed over all the leaves rooted in the subtree; and commitment is the cryptographic commitment.

There is one leaf vertex v_s for each sensor node s with the message $m_s = < s.id, 1, s.value, Hash(N \mid\mid s.id \mid\mid 1 \mid\mid s.value) >$, where N is the query nounce.

Internal vertices represent aggregation operations, and have messages that are defined based on their children. Suppose an internal vertex has child vertices $v_1, v_2, \ldots, v_q$ with the following messages: $m_1, m_2, \ldots, m_q$, where $m_i = < i.id,\ i.count,\ i.value,$

$i.commitment >$. Then the vertex has message $< id, count, value, commitment >$ where $id = s.id$, $count = \sum i.count$, $value = \sum i.value$ and $commitment = H[\ N\ ||\ id\ ||\ count\ ||\ value\ ||\ m_1\ ||\ m_2\ ||\cdots||\ m_q]$.

Since we use the hash function which is collision resistant its impossible for an adversary to change the any of the commitments.

Algorithm 1 CommitmentTreeGeneration

depth = $AggregationTree$.MaxDepth
while depth ≥ 0 **do**
  for all $\mathcal{N} \in AggregationTree$.depth **do**
    $\mathcal{N}.forest$= NULL
    Create ($\mathcal{N}.msg$, $SIGN_{\mathcal{N}}(\mathcal{N}.msg)$)
    Attach ($\mathcal{N}.msg$, $SIGN_{\mathcal{N}}(\mathcal{N}.msg)$) to $\mathcal{N}.forest$
    if $\mathcal{N}.children \neq 0$ **then**
      for all $\mathcal{C} \in \mathcal{N}.children$ **do**
        for all tree root $\mathcal{R} \in \mathcal{C}.forest$ **do**
          if $\mathcal{N}$ has $\mathcal{R}.cert$(else get $\mathcal{R}.cert$) **then**
            if $\mathcal{N}$verifies $\mathcal{R}$.msg (else raise an alarm) **then**
              Add $\mathcal{R}$ to $\mathcal{N}.forest$
        $\mathcal{N}.forest$ = CommitmentTreeCoding ($\mathcal{N}.forest$)
  depth = depth - 1

Algorithm 2 CommitmentTreeCoding

1: $temp$ = SortLinkedList($\mathcal{N}.forest$)
2: **while** $temp.nextTree \neq 0$ **do**
3: **if** $temp.height \neq temp.nextTree.height$ **then**
4: $temp = temp.nextTree$
5: **else**
6: Create an aggregation node A_N
7: $A_N.height = temp.height + 1$
8: $A_N.leftChild = temp$
9: $A_N.rightChild = temp.nextTree$
10: Insert A_N to $\mathcal{N}.forest$
11: Remove $temp$
12: Remove $temp.nextTree$
13: $temp$ = SortLinkedList($\mathcal{N}.forest$)
14: return $temp$

Algorithm 3 Pseudo algorithm to detect a cheater

1: $\mathcal{Q}$ finds out all the $\mathcal{C}_{\mathcal{N}} \in AggregationTree$ using a complainer detecting algorithm
2: **for all** $\mathcal{C}_{\mathcal{N}}$ **do**
3: $\quad\mathcal{Q}$ gets $\mathcal{N}_0$, $SIGN_{\mathcal{N}}$ ($\mathcal{N}_0$)
4: $\mathcal{Q}$ finds possible $CHEATER$ based on $\mathcal{C}_{\mathcal{N}}$
5: **for all** $CHEATER$ **do**
6: $\quad\mathcal{Q}$ gets $\mathcal{N}_{\mathcal{I}}$, $SIGN_{\mathcal{N}}$ ($\mathcal{N}_I$) $CHEATER$ receives and sends.
7: $\quad$If needed $\mathcal{Q}$ gets $\mathcal{N}_{\mathcal{I}}$, $SIGN_{\mathcal{N}}$ ($\mathcal{N}_I$) of the $\mathcal{P}$ $CHEATER$
8: $\mathcal{Q}$ determines the $CHEATER$ based on recived information

Properties of commitment tree and aggregation tree

If you have $O(n)$ children then you need atleast $\Omega(n)$ & at max $O(nlog(n))$ certificates.

If you have $O(n)$ descendents then you need $\Omega(log(n))$ & at max $O(nlog(n))$ certificates.

10. ANALYSIS

10.1 Background

Aggregation tree Commitment tree Analogy with binary representation

10.2 Star Tree

To do : Material on star tree, star tree analysis gives you 1 cert savings.

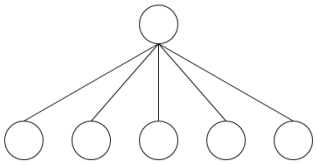

Fig. 10.1.: Star aggregation tree

10.3 Maximum savings

Analysis is true for any n bit forest size. Give names to the following tolopogies. **Maximum savings**, with n(=4) bit forest, fanout(=2), savings of n(=4) certificates:

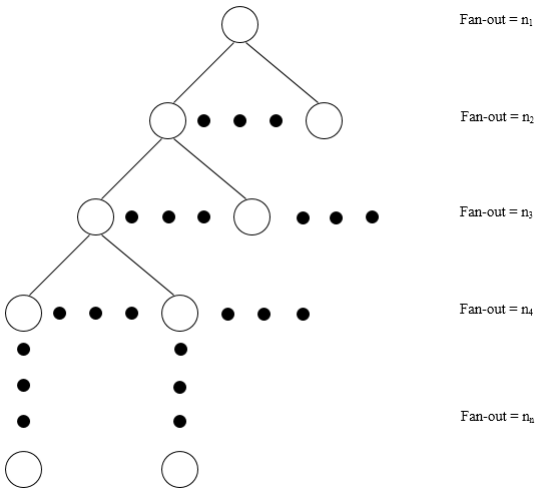

Fig. 10.2.: Symmetric Tree

1	1	1	1	0
0	1	1	1	1
0	1	1	1	1
0	0	0	0	1
A	C	C	C	C

—

1	1	1	1	0
0	1	1	1	1
0	1	1	1	1
0	0	0	0	1
A	A	A	A	A

No savings, with n(=4) bit forest with alternate bit positions, fanout(=2, 3, 5) :

1	0	1	0	0
0	1	0	1	0
0	1	0	1	0
0	0	0	0	1
A	0	A	0	A

—

1	0	1	0	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	0	0	0	1
A	C	A	C	A

—

0	1	0	0	0	0
0	1	0	1	0	0
1	1	1	1	0	0
0	0	1	0	1	0
0	0	1	0	1	0
0	0	1	0	1	0
0	0	1	0	1	0
0	0	1	0	1	0
0	0	0	0	0	1
A	A	0	0	C	A

Savings of n - 1 certificates, with n(=4) bit forest, fanout(=3) :

0	1	1	1	1	0		0	1	1	1	1	0
1	1	1	1	1	0		1	1	1	1	1	0
0	0	1	1	1	1		0	0	1	1	1	1
0	0	1	1	1	1	—	0	0	1	1	1	1
0	0	1	1	1	1		0	0	1	1	1	1
0	0	0	0	0	1		0	0	0	0	0	1
A	0	C	C	C	0		A	0	A	A	A	0

Savings of n - 1 certificates, with n(=4) bit forest, fanout(=4) :

0	1	1	1	0	0		0	1	1	1	0	0
0	1	1	1	1	0		0	1	1	1	1	0
1	1	1	1	1	0		1	1	1	1	1	0
0	0	1	1	1	1		0	0	1	1	1	1
0	0	1	1	1	1	—	0	0	1	1	1	1
0	0	1	1	1	1		0	0	1	1	1	1
0	0	1	1	1	1		0	0	1	1	1	1
0	0	0	0	0	1		0	0	0	0	0	1
A	A	C	C	0	C		A	A	A	A	0	A

10.4 Pseudo Palm Tree

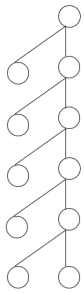

Fig. 10.3.: Pseudo palm tree

Theorem 10.4.1 *At any given level, if an aggregator priorities aggregating its childrens' messages over its own message, it can save bandwidth by not sending its childrens' certificates to its parent.*

Proof We can see from Figure 10.3 that every aggregator has odd number of messages to aggregate, including itself. It means an aggregator at each level has odd number of $1's$ in therir least significant bits. If an aggregator aggregates messages of its children and creates a carry then it needs to send its own certificate to its parent or else it has to send one of its children's certificate as well. Following example illustrates the idea, where C means an aggregator has to send its child's certificate to its parent, A means an aggregator has to send its own certificate to its parent and X is don't care.

0	0	0	1	0		0	0	0	1	0
0	0	0	0	1		0	0	0	0	1
0	0	0	0	1	—	0	0	0	0	1
X	X	X	X	1		X	X	X	X	1
X	X	X	X	C		X	X	X	X	A

Hence, this approach saves bandwidth by sending one less certificate at each level.

■

10.5 Binary tree

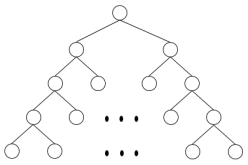

Fig. 10.4.: Binary tree

Theorem 10.5.1 *At any given level, if an aggregator priorities aggregating its childrens' messages over its own message, it can save bandwidth by not sending* $\lceil lg(n/2) \rceil$ *certificates,* n *is number of descendants for given node, to its parent.*

11. THEOREMS

Tree properties & notations

Every node is identical, following the same procedure

Add picture of the topology

d_i is the depth at i

n_i is the fanout at d_{i-1}

c_i is the number of certificates forwarded by each node at d_i to its parent

$= \lceil log((n_{i+1} * c_{i+1}) + 1) \rceil$

N_i is the number of nodes at d_i

$= \Pi_{k=1}^{i} n_k$

T_i is the totality at d_i where totality is the number of certificates received/needed

$= N_i * (n_{i+1} * c_{i+1})$

Total number of nodes in a tree

$= N + 1$

$= N_0 + N_1 + N_2 + +N_3 + ... + N_{n-2} + N_{n-1} + N_n$

$= n_0! + n_1! + n_2! + n_3! + ... + n_{n-2}! + n_{n-1}! + n_n!$

$= 1 + n_1! + n_2! + n_3! + ... + n_{n-2}! + n_{n-1}! + n_n!$

Theorem 11.0.2 *Given an aggregation tree with N + 1 nodes, having $N_i = n_i!$ nodes at depth d_i, equally distributed among thier $n_{i-1}!$ parents then in totality network needs $O(N * log(C))$ certificates where C is a constant.*

Proof Case I: For d_n ; $N_n = n_n!$; $T_n = 0$; $c_n = 1$

Case II: For d_{n-1}

$N_{n-1} = n_{n-1}!$

$T_{n-1} = n_{n-1}! * (n_n * c_n) = n_{n-1}! * (n_n)$

$c_{n-1} = 2$

Case III: For d_{n-2}

$N_{n-2} = n_{n-2}!$

$T_{n-2} = n_{n-2}! * (n_{n-1} * c_{n-1}) = n_{n-2}! * (n_{n-1} * 2)$

$c_{n-2} = \lceil log((n_{n-1} * c_{n-1}) + 1) \rceil = \lceil log((n_{n-1} * 2) + 1) \rceil$

Case IV: For d_{n-3}

$N_{n-3} = n_{n-3}!$

$T_{n-3} = n_{n-3}! * (n_{n-2} * c_{n-2}) = n_{n-3}! * (n_{n-2} * \lceil log((n_{n-1} * 2) + 1) \rceil)$

$c_{n-3} = \lceil log((n_{n-2} * c_{n-2}) + 1) \rceil = \lceil log(n_{n-2} * \lceil log((n_{n-1} * 2) + 1) \rceil + 1) \rceil$

■

Theorem 11.0.3 *Given an aggregation tree with N + 1 nodes, having $N_i = n_i!$ nodes at depth d_i, equally distributed among thier $n_{i-1}!$ parents then in totality network needs $\Omega(N)$ certificates.*

Proof Case I: For d_n ; $N_n = n_n!$; $T_n = 0$; $c_n = 1$

Case II: For d_{n-1}

$N_{n-1} = n_{n-1}!$

$T_{n-1} = n_{n-1}! * (n_n * c_n) = n_{n-1}! * n_n$

$c_{n-1} = 1$

Case III: For d_{n-2}

$N_{n-2} = n_{n-2}!$

$T_{n-2} = n_{n-2}! * (n_{n-1} * c_{n-1}) = n_{n-2}! * n_{n-1}$

$c_{n-2} = 1$

Case IV: For d_{n-3}

$N_{n-3} = n_{n-3}!$

$T_{n-3} = n_{n-3}! * (n_{n-2} * c_{n-2}) = n_{n-3}! * n_{n-2}$

$c_{n-3} = 1$

Case n-2: For d_2

$N_2 = n_2!$

$T_2 = n_2! * (n_3 * c_3) = n_2! * n_3$

$c_2 = 1$

Case n-1: For d_1

$N_1 = n_1!$

$T_1 = n_1! * (n_2 * c_2) = n_1! * n_2$

$c_{n-1} = 1$

■

Things to include in above analysis:

Every node does the same thing

Individual throughput for each node

Theorem 11.0.4 *Given an aggregation tree with N nodes, in totality network needs* $\Omega(N)$ *certificates.*

Proof Let T represent a node in an aggregation tree whose number of children are $T.CHILDREN$, $C$is one of its children and P is its parent.

We can say that T needs certificates of all of its children because while creating a commitment tree T receives at least one $C.msg$ and $SIGN_C$ $(C.msg)$ from all $T.CHILDREN$

If your aggregation tree is such that $\forall T$ needs to send only one message to P then every P receives only $P.CHILDREN$ number of messages. Hence, P needs $P.CHILDREN$ number of certificates.

So, if every P needs certificates only of its children and we have N nodes in the network then since every node has a unique parent as aggregation tree is a rooted tree, in totality we need only N certificates in the network.

■

Theorem 11.0.5 *Given an aggregation tree with N + 1 nodes, having* $N_i = n_i!$ *nodes at depth* d_i*, equally distributed among thier* $n_{i-1}!$ *parents then in totality network needs* $\Omega(N)$ *certificates.*

Proof Let say we have N + 1 nodes in an aggregation tree, also d_i represents depth at level i. Tree is constructed such that root has n_1 children, all n_1 nodes at d_1 have n_2 children, all $(n_1 * n_2)$ nodes at d_2 have n_3 children and all $(n_1 * n_2 * n_3 ... n_{n-1})$ nodes at d_{n-1} have n_n children.

$N + 1 = (1 + (n_1) + (n_2 * (n_1)) + (n_3 * (n_2 * n_1)) + ... + (n_n * (n_{n-1} * n_{n-2} * n_{n-3} ... n_1)))$

All $(n_{n-1} * n_{n-2} * n_{n-3} ... n_1)$ nodes at d_{n-1} need to know certificates of all of their n_n children. If there is only one carry after aggregation then all nodes at d_{n-1} need to send only one certificate to their parent.

All $(n_{n-2} * n_{n-3} * n_{n-4} ... n_1)$ nodes at d_{n-2} need to know certificates of all of their $n - 1$ children.

All n_1 nodes at d_1 need to know certificates of all of their $n - 2$ children.

The root need to know the certificates of all of its n_1 children.

Also, the querier needs to know the certificate of the root.

So, in totality we need $(1 + (n_1) + (n_2 * (n_1)) + (n_3 * (n_2 * n_1)) + ... + (n_n * (n_{n-1} * n_{n-2} * n_{n-3} ... n_1)))$ which is $\Omega(N)$. Hence, prooved.

■

12. NETWORK-FLOW

MESSAGE

ID	COUNT	VALUE	COMMITMENT
20 bits	21 bits	20 bits	256 bits

SIGNATURE (MESSAGE)

$\text{Encryption}_{secret-key_{node}}$(HASH (MESSAGE))
500 bits

CERTIFICATES

Public key	Signature	ID
1000 bits	500 bits	20 bits

NETWORK FLOW

For the given aggregation tree, when a child communicates for the first time with its parent, it sends three things: Message, Signature of message & Certificate. For any subsequent communications a child sends Message & Signature of messsage.

12.1 Star aggregation tree

In star aggregation topology root has to create (n - 1) intermediate vertices, where n is the number of children root has. *Note*: number of certificates = signatures = messages = public keys

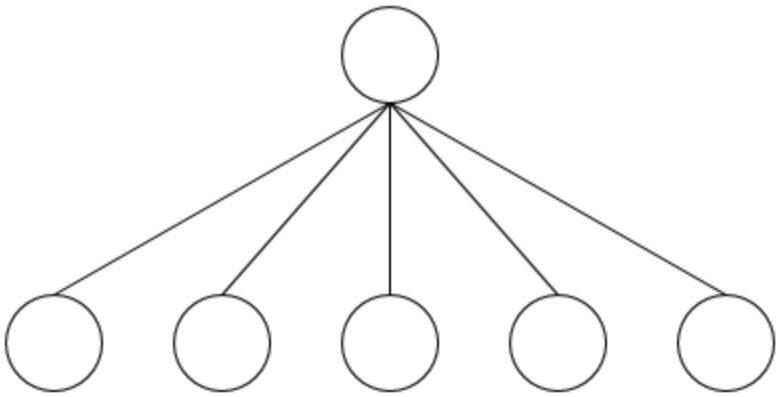

Fig. 12.1.: Star aggregation tree

	#Messages	#Certificates
To root	O (n)	O (n)
From root	1	1

The following property might be true for all possible topologies:

If you have N children, each of your children has n descendent then following equality holds true:

$N < numberofcertificatesneeded < Nlog(n)$

13. SUMMARY

This is the summary chapter.

14. RECOMMENDATIONS

Buy low. Sell high.