DOI: 10.3745/KIPSTC.2008.15-C.5.329

무선 센서 네트워크에서 에너지 효율성과 보안성을 제공하기 위한 클러스터 기반의 Tributaries-Deltas

김 은 경[†] · 서 재 원[†] · 채 기 준[††] · 최 두 호[†††] · 오 경 희[††††]

요 약

무선 센서 네트워크는 여러 가지 제약점을 가지고 있기 때문에 에너지 효율성과 보안성 제공은 중요한 이슈이다. 기존에 라우팅에 있어서 tree-based와 multipath-based라는 두 가지 접근법을 효과적으로 통합시킨 Tributaries and Deltas(TD)가 제안된 바가 있으며, 본 논문에서는 TD 구조에 계층성이 더해진 클러스터 기반의 TD를 제안하여 기존 TD보다 향상된 성능을 증명하였다. 클러스터 기반의 TD 구조는 두 가지 상황에서 기존의 TD보다 더 좋은 성능을 가짐을 보여 주었다. 하나는 베이스스테이션(BS)이 잘못된 정보를 받았다 판단하고 재전송을 요구할 때와, 또 다른 하나는 BS가 이동성을 가지고 있을 때이다. 또한 제안된 구조에 적합한 키 설립 메커니즘을 제안하여 에너지 효율성뿐만 아니라 보안성도 고려한 새로운 센서 네트워크 구조를 제안하였고 TinyOS 2.0을 기반으로 TmoteSKY 센서 보드에 구현하여 실제 네트워크에서의 응용 가능성을 입증하였다.

키워드 : 센서네트워크, 네트워크 구조, 에너지 효율성, 키 설립 메커니즘

Clustered Tributaries-Deltas Architecture for Energy Efficient and Secure Wireless Sensor Network

Eunkyung Kim[†] · Jaewon Seo[†] · Kijoon Chae[††] · Dooho Choi[†††] · Kyunghee Oh[††††]

ABSTRACT

The Sensor Networks have limitations in utilizing energies, developing energy-efficient routing protocol and secure routing protocol are important issues in Sensor Network. In the field of data management, Tributaries and Deltas(TD) which incorporates tree topology and multi-path topology effectively have been suggested to provide efficiency and robustness in data aggregation. And our research rendered hierarchical property to TD and proposed Clustering-based Tributaries-Deltas. Through this new structure, we integrated efficiency and robustness of TD structure and advantages of hierarchical Sensor Network. Clustering-based Tributaries-Deltas was proven to perform better than TD in two situations through our research. The first is when a Base Station (BS) notices received information as wrong and requests the network's sensing data retransmission and aggregation. And the second is when the BS is mobile agent with mobility. In addition, we proposed key establishment mechanism proper for the newly proposed structure which resulted in new Sensor Network structure with improved security and energy efficiency as well. We demonstrated that the new mechanism is more energy-efficient than previous one by analyzing consumed amount of energy, and realized the mechanism on TmoteSKY sensor board using TinyOS 2.0. Through this we proved that the new mechanism could be actually utilized in network design.

Keywords : Sensor Network, Network Architecture, Energy-efficiency, Key establishment mechanism

1. 서 론

무선 센서 네트워크를 구성하여 센서를 통해 다양한 정보들을 수집하는 것은 유비쿼터스 환경으로 진입하는 이 시점에서 많은 분야에서 응용될 수 있음을 보여주고 있다. 그러나 센서 네트워크가 가지는 전력 공급의 제약점으로 인해 에너지 효율성을 높이는 것은 센서 네트워크의 큰 이슈 중 하나로 활발히 연구되고 있으며 기존 네트워크에서 보다 많은 내부 공격 및 외부 공격이 존재하여 보안에 있어서 취약성을 보이기 때문에 보안성 제공 또한 센서 네트워크에서의 중요한 이슈이다. 새롭게 제안되는 센서 네트워크 관련 메커니즘은 주로 수학적 분석이나 시뮬레이션을 통해 그 타당

※ 본 연구는 지식경제부 및 정보통신연구진흥원의 IT핵심기술개발사업 [2005-S-088-04, 안전한 RFID/USN을 위한 정보보호 기술] 사업의 일환으로 수행하였음
† 정 회 원 : 이화여자대학교 컴퓨터정보통신학과 석사
†† 종신회원 : 이화여자대학교 컴퓨터정보통신학과 교수(교신저자)
††† 정 회 원 : 한국전자통신연구원 RFID/USN보안안연구팀 팀장
†††† 정 회 원 : 한국전자통신연구원 RFID/USN보안안연구팀 선임연구원
논문접수 : 2008년 1월 4일
수 정 일 : 2008년 8월 18일
심사완료 : 2008년 8월 26일

성을 증명하고 있으며, 실제 센서노드에 구현하여 증명하는 경우도 있으나 시뮬레이션을 통해 증명하는 메커니즘에 비하여 그 수가 적다.

센서 네트워크의 특성과 요구사항에 맞추어 본 논문에서는 이에 적합한 에너지 효율적인 네트워크 구조를 제시하고 그 구조를 구성하는데 유리할 뿐만 아니라 보안성도 제공하는 키 관리 메커니즘을 도입하였다. 또한 실제 네트워크 설계에 응용될 수 있음을 증명하고자 제안된 메커니즘을 TinyOS 2.0을 기반으로 TmoteSKY 센서 보드에 구현하여 에너지 효율성과 함께 메커니즘의 실효성을 증명하였다. 이러한 연구 결과를 이용하여 에너지 효율성뿐만 아니라 보안성도 고려하는 센서 네트워크를 설계 하는데 도움이 될 것으로 예상된다.

본 논문은 다음과 같은 순서로 구성되어 있다. 1장의 서론에 이어서 2장에서는 센서 네트워크의 기본 개념과 함께 기존에 제안되어있는 네트워크 구조와 키 설립 메커니즘에 대해서 살펴볼 것이다. 3장에서는 본 논문에서 초점을 두고 있는 문제점을 제시하고 이에 알맞은 새로운 구조를 제안할 것이며 4장을 통해 구현 결과를 바탕으로 기존의 센서네트워크와 비교한 에너지 효율성을 증명할 것이다. 5장에서는 제안된 구조와 보안 구조를 마무리하면서 향후 연구에 대하여 논할 것이다.

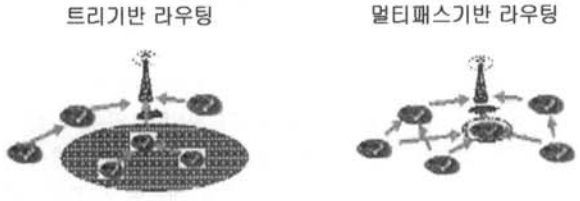
2. 관련연구

2.1 센서 네트워크 구조

2.1.1 Tributaries-Deltas 구조의 센서 네트워크

센서 네트워크는 응용 목적에 따라 다양한 구조로 구성하는 것이 가능하지만, 데이터 통합의 관점에서 평면 구조의 센서 네트워크를 트리 기반의 센서 네트워크와 멀티패스 기반의 센서 네트워크로 나누어 볼 수 있다.

센서 네트워크에서 데이터를 통합하는 방법인 트리 기반의 토폴로지와 멀티 패스 토폴로지는 각각 장점과 단점을 가지고 있다. 정확한 자료 수집에 효율적인 트리 토폴로지는 정확한 자료 수집에 적합하지만 네트워크 내의 노드 실패에 큰 영향을 받는 단점을 가지고 있다. 반면에 멀티 패스 토폴로지의 경우, 중복되는 값이 있을 수 있지만 자료를 잃어버릴 확률이 적기 때문에 네트워크 내의 노드의 실패에 민감하지 않은 특성을 가지고 있다.

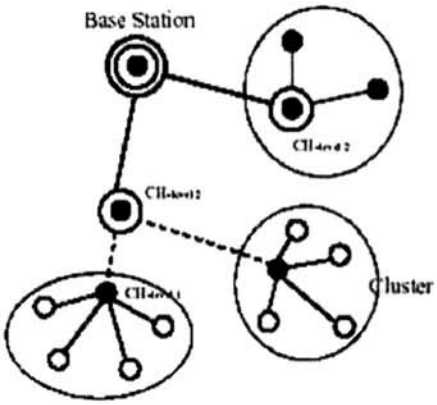


(그림 1) 트리 토폴로지와 멀티 패스 토폴로지의 예

A.Manjhi 는 [2]을 통해 트리 토폴로지와 멀티 패스 토폴로지를 통합하는 구조를 제안하였다. 정확한 자료 수집에 효율적인 트리 토폴로지와 중복되는 값이 있을 수 있지만 자료를 잃어버릴 확률이 적은 멀티 패스 토폴로지를 각각 다른 지역으로 배치하여 동시 수행함으로써 두 가지를 통합하는 방식을 제안하였다. Tributaries and Deltas 구조는 BS로부터 멀리 떨어져 있는 센서들을 트리 구조(tributaries)로 조직하고 BS로부터 가까운 센서들은 멀티 패스(deltas)로 구성하도록 한다. 데이터 수집 및 통합을 수행하면서 네트워크의 상태에 따라 멀티 패스 영역을 늘리거나 줄임으로써 데이터를 정확히 전달하도록 한다. 예를 들어, 데이터 손실률이 높아지면 근사치의 값이 오더라도 확실한 전달이 보장되는 멀티 패스 영역을 넓히도록 네트워크를 재구성함으로써 네트워크의 견고성을 보장한다.

2.1.2 계층 구조의 센서네트워크

계층 구조의 센서 네트워크는 멀티 홉 무선 네트워크의 확장으로 계층화시킨 것을 말한다. 클러스터링의 기본적인 알고리즘을 보면, 네트워크를 노드들의 부분집합으로 나누는데 대부분 지역적으로 가까운 센서끼리 모이게 되며, 이를 클러스터라고 부른다. 각각의 클러스터는 클러스터 헤드(Cluster Head : CH)라고 불리는 하나의 센서를 가지고 있으며 주로 데이터 통합과 베이스스테이션(Base Station : BS)으로 데이터를 보내는 역할을 한다. 현재 다양한 접근방법을 가지고 클러스터링 프로토콜이 제안되어 있으며 이러한 구조는 네트워크 지연성 측면에서도 일반 멀티 홉 구조에 비해 좋은 성능을 가진다. Low Energy Adaptive Clustering Hierarchy(LEACH), chain-based 3 level PEGASIS, Maximum Energy Cluster Head(MECH) 등은 대표적인 클러스터링 프로토콜이다. 그 중에서도 LEACH는 한 계층의 클러스터를 구성하는 방법에 대하여 제안하고 있다. 클러스터를 구성하는데 앞서, 클러스터 헤드의 수는 이미 정해져 있고, 어떤 노드가 클러스터 헤드가 될 것인지는 노드들 중에 일정 계산법에 의해 무작위로 선정하도록 한다. 이는 일반 센서끼리 전송할 때보다 베이스스테이션과 통신을 하게 되는 클러스터 헤드가 더 많은 에너지를 소모하게 되는데 이를 모든 센서 노드에게 분산시키기 위함이다. 결정된 클러스터 헤드는 자신의 멤버들에게 자신의 위치를 알리면서 클러스터를 구성한다. 클러스터 헤드는 자신의 클러스터에 속한 멤버 노드에게 TDMA 스케줄을 제공하여 전송 순서를 정한다. 마지막 데이터를 받으면 다음 라운드의 클러스터 헤드를 뽑는다. 클러스터링을 함으로써 가지게 되는 장점 중 본 논문에서 초점을 두고 있는 것은 에너지 효율적이라는 점이다. 클러스터링을 함으로써 일반 센서들로 구성된 평면 구조의 센서 네트워크에서 비해 에너지 절감을 효과를 볼 수 있다. 센서가 소모하는 전력량 중 통신을 위해 사용하는 부분이 다른 계산하는 부분에 비해 크기 때문에 클러스터로 네트워크를 나누어 설계하는 것은 에너지 효율적이라는 장점을 가지게 한다. 일반 센서에서 모



(그림 2) 계층 구조의 센서네트워크

은 데이터를 클러스터 헤드의 계층적 구조를 통해 베이스스테이션으로 보내는데, 클러스터 안에서는 가까운 거리에 있는 센서들끼리 통신하기 때문에 모든 센서들이 계층적이지 않게 서로 통신하여 모든 센서가 BS와 통신하는 것에 비하여 적은 전력을 소모하게 된다.

2.2 센서 네트워크에서의 키 설정 메커니즘

이 절에서는 센서 네트워크에서의 노드 및 게이트웨이에 대한 보안성을 제공하는 보안 연구를 키 관리 및 분배 기술을 중심으로 살펴보도록 한다. 현재까지 순수하게 키 생성 및 관리 메커니즘만을 제안한 것뿐만 아니라 클러스터를 기반으로 하는 센서 네트워크 구조, 혹은 중간에 Aggregator를 두어 계층적 성격을 갖는 센서 네트워크의 구조, 육각의 셀 형태로 네트워크를 나누는 등 다양한 구조에 기반하여 이 구조에 적합한 키 설정 및 분배 등과 연관된 메커니즘들을 제안한 몇몇 연구들이 보고되고 있다. 본 논문에서는 선분배 키풀 방식의 키 관리 메커니즘에 대해서 자세히 알아보도록 한다.

2.2.1 A Key Management Scheme for DSN

L. Eschenauer, V. Gligor가 [6]을 통해 센서 네트워크에서 노드간 pairwise 키 설정을 위해 제안한 프로토콜은 베이스 스테이션이 먼저 다량의 랜덤 키를 생성하여 이를 키 풀(pool)에 저장하고 키 풀에서 무작위로 임의의 키 집합을 선택하여 키 링을 생성하여 이를 각 센서 노드에게 분배한다. 센서 노드들은 자신이 갖고 있는 키 링의 키 정보를 이웃 노드들에게 브로드캐스트함으로써 무선 통신 반경 내에서 자신의 이웃하는 노드들과 공유키를 찾는다. 두 링크 또는 그 이상 떨어져 있으면서 서로 공유하는 키가 없는 임의의 두 노드가 공유키를 갖기 위해서는 path key를 생성하여 공유한다. Path key를 설정하고자 하는 두 노드는 두 노드간의 direct link path를 통해 키를 교환하여 공유키를 설정한다. 이 스킴은 센서 노드의 개수가 매우 많더라도, 수백개 정도의 키로 기존의 pairwise key와 동일한 안전성을 제공한다는 장점을 갖는다. 키 분배 및 설립을 위한 상세 프로토콜은 다음과 같다.

- 1단계 : 키 풀 P와 키의 id를 생성하고 무작위로 선택된 k개의 키를 묶어서 키링을 생성한다.
- 2단계 : 각 센서 노드는 키링을 선택해서 메모리에 장착하고 키링의 id와 자신의 노드 id를 trusted controller node에 저장한다. 또한 자신에 관한 정보를 가지고 있는 지 controller node의 id를 메모리에 저장한다.
- 3단계 : 센서 노드는 자신이 가지고 있는 키링에 속한 키의 id를 평문으로 한 홉 내의 이웃 노드에게 브로드캐스트 한다.
- 4단계 : 이웃 노드가 소유한 키의 id와 자신이 소유한 키의 id를 비교하여 공통키를 찾아내어 자신의 로컬 키 그래프에 저장한다.
- 5단계 : 한 홉 통신 범위 내의 A 노드와 B 노드 사이에 공유키가 없을 경우 이 노드들과 키를 동시에 공유하는 C 노드를 경유해서 path-key를 설정한다. C 노드는 자신의 키 링에서 쓰이지 않은 키를 한 개 선택해서 A 노드와의 공유키로 암호화해서 A 노드로 보내고 B 노드와의 공유키로 암호화해서 B 노드에게 보내준다.

2.2.2 A Pairwise Key Pre-distribution Scheme for WSN

Wenliang Du, Jing Deng, Yunghsiang S. Han, Pramod K. Varshney는 [11]을 통해 Blom의 키 사전 분배 스킴을 바탕으로 이 스킴과 랜덤 키 사전 분배 방식을 조합한 키 설정 메커니즘을 제안하였다. 제안 메커니즘은 기존의 키 분배 방식에 비해 네트워크 연결 가능성 및 저항성을 높였으며 Blom의 방식을 채택함으로써 λ - security를 제공할 수 있다는 장점을 갖는다. 이 메커니즘은 노드가 오염되는 경우의 네트워크 전체에 미치는 영향을 기존의 메커니즘에 비해 낮출 수 있으며 키 스페이스가 노출되는 경우 이를 공유하는 노드수를 줄임으로써 상대적으로 적은 노드가 위험에 노출되도록 하여 보안성을 높일 수 있다는 장점을 갖는다. 키 분배 및 설립을 위한 상세 프로토콜은 다음과 같다.

- 1단계 : G 행렬을 생성한다. 이 때 G 행렬은 모든 선형 독립적이어야 하며 각 노드가 G로부터 한 행을 사전에 분배받을 때 저장 장소의 오버헤드를 줄이기 위해 다음과 같이 하나의 seed 값만으로 나머지 위치의 값을 계산할 수 있는 형태로 구축된다.
- 2단계 : D 행렬을 생성한다. 이 때 D 행렬은 Blom의 스킴에서와 같이 대칭행렬로 만들어지며 이를 이용하여 비밀행렬 A를 계산할 수 있다.
- 3단계 : 다중의 키 space를 선택한다. 랜덤하게 서로 다른 키 스페이스를 선택하여 만일 두 노드가 동일한 키 스페이스로부터의 키 정보를 갖고 있다면 둘 사이에 키가 설정되도록 한다.
- 4단계 : 각 센서 노드가 배치된 후 모든 노드는 자신과 자신의 이웃노드간에 공유하는 키 스페이스를 찾는다. 이를 위해 각 노드는 자신의 노드 id, 키 스페이스의 인덱스, G 행렬의 seed 값을 브로드캐스트하고 서로 간에 키 설정이 가능한지 파악한다.

3. 클러스터 기반의 Tributaries-Deltas 구조

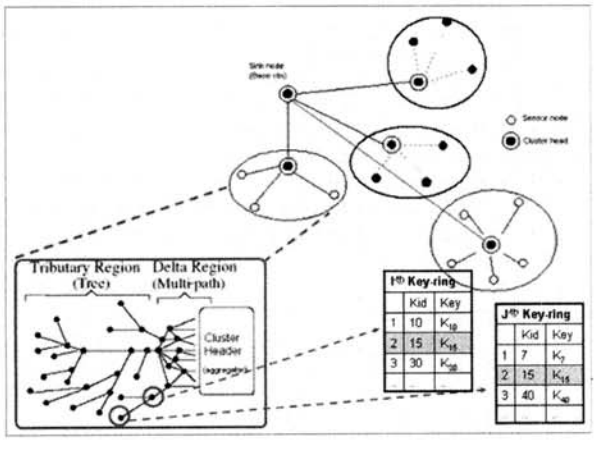
3.1 제안하는 네트워크 구조

앞서 살펴본 Tributaries-Deltas 구조는 평면 구조 네트워크에서 에너지 효율적인 면은 물론 데이터 통합에서도 장점을 가지고 있다. 하지만 이 구조에 하나의 계층을 더 얹음으로써 그 에너지 효율성은 극대화될 수 있다. (그림 3)은 본 논문에서 제안하는 클러스터 기반의 Tributaries-Deltas 네트워크 구조이다.

[2]에서 제안된 Tributaries-Deltas(TD) 구조에 발생 가능한 상황을 토대로 TD 구조가 가지는 세 가지 문제점을 제시하고자 한다.

첫째로 통합 되어서 베이스스테이션에서 받은 데이터가 적합하지 않다고 결정이 되었을 때 Tributaries-Deltas 구조에서 트리 노드들은 정확한 데이터를 전송할 수 있지만, 중간 노드가 공격을 당할 경우 그 데이터 자체를 잃어버릴 수 있다. 또한 무사히 멀티 패스 노드까지 전달되었다 하더라도 모든 노드가 서로에게 전송함으로써 이를 구별하기 위해 전송되는 데이터가 길어지고, 중복되는 데이터를 통합하여 새로운 시놉시스를 만들어내는 과정에서 정확한 데이터를 만들지 못할 수 있다. 이런 경우, 베이스 스테이션은 다시 데이터를 통합할 것을 결정하고 질의를 네트워크에 재전송하는데 네트워크에 브로드캐스트함으로써 모든 센서들은 똑같은 결과를 가지고 두 배의 에너지를 소모해야 한다.

두번째로 베이스스테이션이 이동성을 가진 모바일 에이전트인 경우, 모바일 에이전트가 질의를 보내고 자리를 이동하는 경우를 고려해보자. 이미 분포되어 있는 노드들은 질의가 보내진 모바일 에이전트의 위치 정보를 분석하고 이에 맞추어 Tributaries-Deltas 구조로 네트워크를 구성한다. 하지만, 네트워크를 구성하고 데이터 통합을 하는 동안 모바일 에이전트가 움직인다면 BS를 중심으로 이미 구성된 멀티 패스 노드들과 베이스 스테이션 사이의 거리에 변동이 생긴다. 원래 위치에서 이동한 위치까지의 거리 차가 많지 않다면 에너지 소비량에 큰 변화가 없겠지만, 베이스 스테이션의 거리 변화가 크다면 최종적으로 베이스 스테이션과 통신해야 하는 한 홉 내의 모든 멀티 패스 노드들은 더 많은 에너지를 가지고 전송을 해야 한다. 이러한 상황에서 [2]에 제안된 TD 구조를 도입한다면 에너지 소모량이 극대화되어서 사용할 수 없는 노드의 수가 급격히 증가할 것이다.



(그림 3) 클러스터 기반의 Tributaries-Deltas 구조

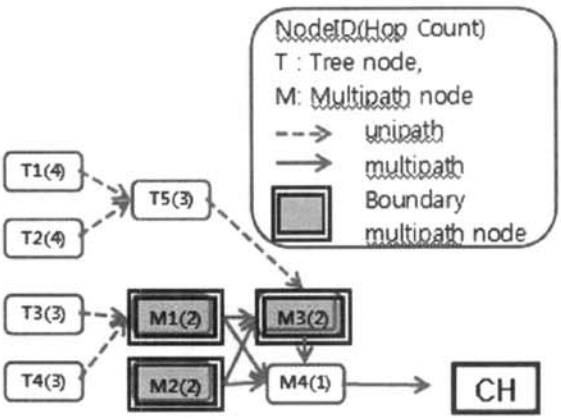
세 번째로 센서 네트워크는 노드의 제한점 때문에 많은 공격에 노출되어있으며 이를 방지하기 위해 공격에 대항할 수 있는 보안 메커니즘이 반드시 필요하다. 특히 보안성 제공의 기반이 되는 키 분배 및 관리 메커니즘이 필요하지만 기존의 Tributaries-Deltas 방식은 보안성을 고려하지 않았기 때문에 공격에 취약성을 보인다.

위에서 언급한 세 가지 문제점을 해결하기 위해 기존의 TD의 취약점을 보완한 클러스터 기반의 Tributaries-Deltas 구조를 제안한다. 클러스터 헤드의 수는 정해져 있고, 클러스터 헤드의 위치와 각각의 클러스터 헤드가 포함할 수 있는 지역은 이미 선언되어 있다. 키풀에서 미리 만들어진 키 링을 미리 탑재한 센서 노드가 랜덤하게 분포되면 클러스터 헤드들은 자신의 지역 안에 들어와 있는 센서들에게 신호를 브로드캐스트하여 클러스터 헤드의 위치와 센서 자신이 누구와 같은 클러스터를 구성해야 하는지를 통보한다. 질의가 보내지면 같은 클러스터의 멤버들을 찾아내어 Tributaries and Deltas 구조로 설계된다. 클러스터 기반의 Tributaries-Deltas 구조를 통해 평면 센서 네트워크에서 가장 효율적인 Tributaries-Deltas 구조를 활용하여 그 장점과 계층 네트워크 구조의 장점을 통합하였다.

3.2 키 설정 및 동작 메커니즘

본 논문에서 제안하는 클러스터 기반의 Tributaries-Deltas 구조는 미리 정해진 수만큼의 클러스터 헤드와 수 많은 센서 노드로 이루어지며, 센서 노드가 랜덤하게 분포되면 클러스터 헤드들은 자신의 지역 안에 들어와 있는 센서들에게 신호를 브로드캐스트하여 클러스터를 구성하는데, 클러스터 헤드를 향하는 Tributaries-Deltas 구조로 클러스터 내부를 구성하게 된다. 이 때 우선적으로 이루어지는 것은 키 설립으로, 각 센서 노드는 미리 탑재한 키의 아이디를 브로드 캐스트 함으로써 공통키를 설립하여 로컬 키 그래프를 완성하게 된다. 직접 키를 설정할 수 없는 경우 패스 키를 설정하여 로컬 키 그래프를 작성하게 되며 클러스터 헤더와 베이스 스테이션은 같은 방법 혹은 CH-BS간의 그룹 키로 통신을 하게 된다.

(그림 4)는 클러스터 기반의 Tributaries-Deltas의 내부 구조를 보여주고 있다. 각 노드는 현재 네트워크의 상황에 따라 자신이 트리 노드인지 혹은 멀티 패스 노드인지를 명시하는 로컬 변수를 가지고 있으며 키 설립을 통해 완성된 키 그래프를 바탕으로 홉 카운트를 유지하게 된다. 멀티 패스 영역의 확장 혹은 축소가 필요한 시점이 되면 명령에 따라 멀티 패스 영역의 경계 노드를 기준으로 주어진 홉 카운트 만 큼 확장이나 축소를 수행하게 된다. 세부 동작 메커니즘은 다음에 명시되어 있다.



(그림 4) 클러스터 기반의 Tributaries-Deltas 내부 구조

- Setup Phase

1단계 : Key pre-distribution

① 키 풀 P에서 랜덤하게 생성된 키 링을 각 센서 노드마다 하나씩 로드한다

② 필드에 배치된 후 BS에서 setup_req를 보내면 클러스터 헤더는 주변의 노드에 join-req를 보낸다.

③ 각 센서노드는 cluster join-req의 신호에 따라 CH를 중심으로 로컬 키 그래프 구성을 시작한다.

2단계 : Shared key discovery

④ 각 센서 노드는 자신이 가지고 있는 KeyID를 브로드캐스트하고 자신의 부모노드를 중심으로 공유키를 발견하여 local key graph를 구성한다.

3단계 : Path key establishment

⑤ 부모 노드와 직접 키를 공유하고 있지 않은 경우에는 path key 설정을 통해서 local key graph를 구성한다.

4단계 : Secure communication

⑥ 각 센서 노드의 성격에 따라 수신할 노드를 선택하여 암호화된 데이터를 전송한다.

- Tree node : 자신의 부모 노드 중에서 direct parent node에게 공유키로 데이터를 암호화해서 보낸다.
- Multipath node : 자신의 모든 부모 노드에게 각각의 공유키로 암호화해서 데이터를 전송한다.

- Shrinking and Expanding Phase
  - Shrinking Phase

① Data loss rate가 threshold 값보다 낮아지면 네트워크가 안정 상태에 있는 것이므로 BS 혹은 CH가 shrink_req(홉 수) 메세지를 보낸다.

② Delta 지역의 boundary multipath node를 기준으로 홉 수만큼 CH를 향해 shrinking을 진행한다.

- Multipath node -> tree node로 전환, 하나의 부모 노드에게만 data를 전송하게 됨. 전환되는 노드를 기점으로 ID(홉 수) 갱신한다.

  - Expanding Phase

① Data loss rate가 높아지면 BS은 모든 CH, 혹은 해당 CH에게 expand_req(홉 수) 메시지를 보낸다.

② Delta 지역의 boundary multipath node를 기준으로 홉 수만큼 Tributary 영역을 향해 expanding을 진행한다.

- Tree node -> multipath node로 전환, 모든 부모 노드에게 data를 전송하게 되고, 전환되는 노드를 기점으로 ID(홉 수)를 갱신한다.

- Re-Setup Phase

Timer가 만료되어서 네트워크 토폴로지를 재구성해야 하거나 mobile agent가 다시 setup-req를 보내는 경우 위의 Setup Phase ②부터 반복하여 수행한다.

3.3 제안한 메커니즘의 특성 및 강점

클러스터 기반의 Tributaries-Deltas 센서 네트워크는 앞서 언급한 세 가지 문제점을 해결할 수 있다. 첫 번째 문제에 대한 해결책은 중간에 데이터가 손실되거나 왜곡되어 잘못된 데이터가 들어왔을 때, 베이스 스테이션은 자신에게 데이터를 전송한 클러스터 헤드가 관리하는 지역 중에서 잘못된 데이터를 보냈을 확률이 큰 지역을 결정하고, 이에 해당되는 클러스터 헤드에게 데이터 통합을 요청하는 질의를 재전송한다. 그렇다면, 그 클러스터 헤드가 해당하는 클러스터 내의 센서들만 다시 데이터 통합을 수행하게 됨으로써 클러스터 헤드의 수의 비율로 에너지 소모량을 절감할 수 있다.

베이스스테이션이 이동성을 가진 경우, 네트워크의 변동사항에 민감한 TD 구조가 클러스터로 인해 한 단계 아래 감춰지기 때문에 네트워크의 변화가 전체 네트워크에 크게 영향을 미치지 않는다. 베이스 스테이션의 이동으로 통합된 데이터를 최종적으로 보내야 하는 거리가 커졌다고 한다면 전체 네트워크를 Tributaries-Deltas 구조로 구성했을 때의 멀티 패스 노드보다 수적으로 적은 클러스터 헤드들과의 통신 에너지 소모량만 늘어나는 것이고, 멀티패스 노드의 극심한 에너지 소모로 인한 노드 사용 불가 상태로 되는 것을 늦출 수 있기 때문에 일반적인 Tributaries-Deltas 구조에 비해 에너지 효율성과 함께 비용 절감 측면에서도 더 효과적인 네트워크 구조라고 볼 수 있다.

또한 키 설정 메커니즘인 EG 메커니즘을 이용하여 클러스터 헤드와 센서간의 인증과 기밀성을 제공한다. EG 메커니즘은 [6]에서 랜덤 그래프 이론을 통해 일정 확률 이상이면 네트워크가 연결됨을 증명하였으며 간단한 방식으로 키가 설립되기 때문에 계산 오버헤드가 적다. 따라서 제약 사항이 많은 센서 네트워크에 적합한 방식이라고 할 수 있다. 또한, TD구조를 구성하기 위해서 이루어져야하는 이웃 노드들 간의 부모 자식 관계 파악을 EG메커니즘을 도입함으

로써 한 번에 해결할 수 있다. 본 논문에서 제한한 클러스터 기반의 TD 구조에서는 EG 메커니즘을 기반으로 한 키 설정 과정을 먼저 수행하여 로컬 키 그래프를 구성함으로써 TD를 구성하는데 필요한 트리 노드에 필요한 부모 자식 간 노드의 관계와 멀티 패스 노드에 필요한 이웃 노드 파악도 용이하게 수행한다.

4. 구현 내용 및 결과 분석

4.1 구현 환경 및 시나리오

4.1.1 구현 환경

본 논문을 통해 제안된 클러스터 기반의 Tributaries-Deltas 구조를 구현하기 위해 사용된 개발 환경은 다음과 같다.

4.1.1.1 TinyOS 2.0

TinyOS는 UC Berkeley 대학에서 초기 연구를 진행하여 현재는 오픈 소스 프로젝트로 계속 발전되고 있다. 센서 네트워크를 위해 컴포넌트 기반의 프로그래밍 언어인 NesC를 기반으로 태스크(Task)의 동시(concurrency) 작업 지원에 초점을 두고 있다. 이를 위해 하위 레벨의 메시지 전달과 실행 배정 프리미티브(dispatching primitive)를 제공한다.

4.1.1.2 TmoteSKY

Moteiv사는 UC Berkeley 대학의 센서 네트워크 운영체계인 TinyOS와 Mote를 기반으로 센서 네트워크의 연구개발 및 애플리케이션을 위한 Telos 시리즈의 상용 플랫폼을 제공하고 있다. Berkeley Mote를 토대로 발전되어 온 Moteiv사의 Mote는 현재 Telos revision A, Telos revision B 등의 여러 유형의 플랫폼이 개발된 상태이다. TmoteSKY는 msp430 마이크로 콘트롤러와 CC2420 RF Chip을 탑재하고 있으며, 센서가 보드에 통합되어 있는 on-board 센서노드이다. IEEE 802.15.4 표준과 호환되며 250 kbp 의 높은 데이터 전송률을 보인다.

4.1.2 구현 시나리오

클러스터 기반의 TD 구현을 위해 다음과 같은 시나리오로 클러스터 헤드의 내용을 구성하였다

- 1단계 : 로컬 키 그래프 구성
  - ① 다이렉트 키 설정
  - ② 패스 키 설정
- 2단계 : Tributary 영역과 Delta 영역 설정
  - ③ 홉 카운트 결정
- 3단계 : 네트워크 재구성
  - ④ 확장 명령
  - ⑤ 축소 명령

진행 단계에 맞추어 전송되는 패킷의 형태는 <표 1>과 같다. 각 패킷의 packet_flag 변수의 값에 따라 패킷의 속성이 정해지며 변수 값에 따른 패킷의 속성은 <표 2>와 같다.

센서 노드는 자신만의 변수를 유지하고 있으며 그 변수 값을 통해 수신된 패킷의 명령대로 동작을 취하게 된다. <표 3>은 센서 노드가 가지는 주요 변수의 의미를 보여주고 있다.

구현에는 총 7개의 노드가 이용되었으며, 각 노드가 가지는 키의 ID는 <표 4>와 같다. <표 4>에 명시된 키의 ID에 따라 클러스터 헤드를 포함한 센서 노드는 키 설정 단계에서 공통키를 찾아내어 키 그래프를 완성하게 되고 이를 통해 Tributaries-Deltas 구조를 구성하기 위한 기초 자료를 수집하게 된다. 최종 완성 된 노드의 계층 관계는 (그림 5)과 같다.

〈표 1〉 패킷 형식

변수명	의미
id	송신자 ID
count	패킷 카운트
packet_flag	현재 패킷의 속성
readings [NREADINGS]	센싱 데이터
hopcount	수신자의 홉 수
carrier [KEY_RING_SIZE]	송신자가 가진 키 ID (초기에 해당)
carrier2 [PATH_ANS_SIZE]	사용 안함 (초기에 해당)

〈표 2〉 packet_flag 값에 따른 속성

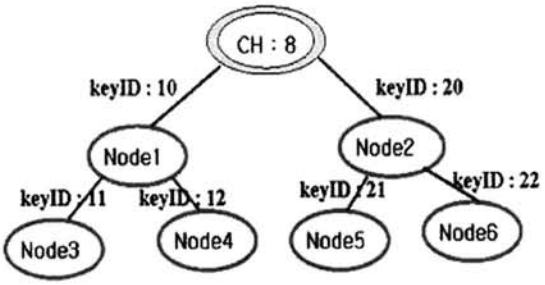
변수값	의미
1	다이렉트 키 설정
2	패스 키 요청
3	패스 키 응답
4	홉 수 결정
5	멀티 패스 영역 확장
6	멀티 패스 영역 축소
7	데이터 통합 명령
8	데이터 통합 응답
9	네트워크 재구성 명령
10	휴면 상태

〈표 3〉 각 센서노드가 가지는 변수의 의미

변수명	의미
keyid[MAX_KEY_RING_SIZE]	센서 노드가 가진 키 ID
keyinfo[MAX_KEY_RING_SIZE]	센서 노드가 가진 키 값
keyused[MAX_KEY_RING_SIZE]	키링의 키가 쓰였는지 여부
nodeid[MAX_NODE_NUMBER]	공통키 설정을 마친 노드의 ID
directkey[MAX_NODE_NUMBER]	공통키 설정에 쓰인 키의 ID
keyflag[MAX_NODE_NUMBER]	해당 인덱스의 노드와 키를 설정했는지 여부
pathphase[MAX_NODE_NUMBER]	패스키 설정 단계 여부
max_hopcount	네트워크 내 허용최대 홉 카운트
my_hopcount	자신의 홉 카운트
my_parent	부모 노드 ID
treenode	자신이 트리 노드에 속하는지 여부

〈표 4〉 각 노드가 가진 키 ID

노드ID	1	2	3	4	5
CH : 8	58	10	20	41	42
1	51	10	11	12	30
2	52	11	31	43	44
3	53	12	31	45	46
4	54	12	31	47	48
5	55	21	32	49	60
6	56	22	32	61	62



(그림 5) 노드간 계층 구조

각 센서 노드는 트리 노드로 설정 시, 부모 노드와 공유하는 키로 데이터를 암호화하여 전송하고 멀티 패스 노드로 설정 시에는 부모 노드 뿐만 아니라 키를 공유하는 이웃 노드 모두에게 암호화된 데이터를 전송하게 된다.

4.2 구현 내용 및 동작

이 장에서는 실제 구현 내용과 동작을 클러스터 헤드인 8번 노드의 명령 중심으로 1계층에 해당하는 노드 1번과 2계층에 속하는 노드 3번의 수신 되는 패킷에 따른 상태 변화에 대해서 알아본다. TinyOS 2.0에서는 Printf 라이브러리를 제공하고 있으며 본 논문에서는 노드의 상태 변화를 알아보기 위해 printf() 함수를 이용하여 변수의 변화를 알아보았다.

4.2.1 클러스터 헤더의 초기화 및 다이렉트 키 설정 단계

센서 노드가 부팅 될 때에는 Boot() 모듈에 명시된 내용에 따라 초기화를 수행하게 되어 있으며 본 구현 에서는 초기화 직 후 다이렉트 키 설정을 수행하게 된다.

클러스터 헤더의 초기화 및 다이렉트 키 설정 단계

```
event    void Boot.booted() {
    local.interval = DEFAULT_INTERVAL;
    local.id = 8; //클러스터헤드
    local.packet_flag = 1;
    local.hopcount = 1;
  for(i=0; i<MAX_NODE_NUMBER+1; i++)
  //에러를 방지하기 위해 배열 1번부터 사용함
  //1 이면 키 설정한 적 없음 , 2이면 다이렉트 키 설정됨
  //3이면 공통키 없어서 다이렉트 키 설정 안됨
      keyflag[i]=1;
  for(i=0; i<MAX_NODE_NUMBER+1    ; i++)
    pathphase[i]=1;
    for(i=0; i<MAX_KEY_RING_SIZE ; i++)
  //키 사용됐는지 여부를 초기화
            keyused[i]=0;
}
```

노드ID	실행 결과 / 내용
CH : 8	/opt/tinyos-2.x/apps/TD_EG/1129/TD_EG_CH count: 1 flag: 1 hop : 1 1_1: 10 2_1 : 0 count: 2 flag: 1 hop : 1 1_1: 10 2_1 : 0 count: 3 flag: 1 hop : 1 1_1: 10 2_1 : 0 클러스터 헤드는 부팅과 동시에 다이렉트 키를 요청하는 1번 패킷을 송신 한다.
1	/opt/tinyos-2.x/apps/TD_EG/1201/EG_scheme_node1 nodeid[1] : 3 , directkey[1] : 11, keyflag[1] : 1 Tree : 1 nodeid[2] : 8 , directkey[2] : 10, keyflag[2] : 1 Tree : 1 1번 노드는 CH 노드와 3번 노드의 다이렉트 키 설정 요청 메시지를 받아서 공통키를 찾아내고 이를 자신의 변수에 저장하여 보관 한다.
3	/opt/tinyos-2.x/apps/TD_EG/1201/EG_scheme_node3 0 parent : 1 nodeid[1] : 1 , directkey[1] : 11, keyflag[1] : 2 Tree : 0 parent : 1 3번 노드는 1번 노드의 다이렉트 키 설정 요청 메시지를 받아서 공통키를 찾아내어 자신의 변수에 저장하여 보관한다.

4.2.2 패스키 요청 단계

각 센서 노드는 다이렉트 키 설정 후 패스키 설정이 필요한 노드의 리스트를 작성하여 자신의 이웃 노드에게 전송한다. 본 구현에서는 노드 개수의 제약으로 인해 의도한 형태의 로컬 키 그래프를 구성하고자 패스키 설정이 제대로 동작하는 사항만을 확인하였다

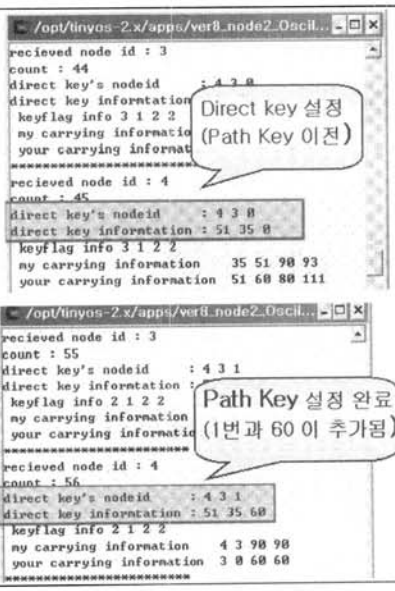
패스키 요청 단계

```
//   path key establishment 단계
 if(countnum==50   && pathflag==1){ //countnum은 충분한 숫자로 결정해야 함
     local.packet_flag    = 2;
     j=1;
     for(i=1 ; i <MAX_NODE_NUMBER+1 ; i++) {
   // keyflag를 돌면서 path key설정 필요한 노드들(값=3)을 모두 저장
              if(keyflag[i]==3){
                       local.carrier[j]=i;
                       j++;}
              local.carrier[j]=0;} //마지막을 표시하기 위해 0을 넣음
     local.flag_path_key=1;}
```

패스키 처리 단계

```
//   path key msg를 받으면 동작
if( omsg->flag_path_key == 1 && pathphase[omsg->id] == 1 ){
      pathphase[omsg->id] = 0;
      i=1;
      if( keyflag[omsg->id] == 2 ) {
     // 받은 패킷의 노드 id에 해당하는 다이렉트 키가 있어야 함
          while( omsg->carrier[i] != 0 ){
            if((keyflag[omsg->carrier[i]] == 2) && (omsg->id < omsg->carrier[i])){
              //받은 패킷에서 path key 원하는 노드에 해당하는 키가 있으면
              //local.carrier2[1]에서부터   정보 담아서 보내야 함
              // (keyid 정보 + key 정보 )
              //keyid   와 나누어진 key 정보를 암호화 하여 담음
                     local.packet_flag = 3;  }
    }}}
```

패스키 테스트



4.2.3 홉 수 결정 단계

홉 수 결정 단계에서는 클러스터 헤드로부터 시작하는 홉 수 결정 패킷이 생성되며 각 노드는 자신의 이웃노드로부터 수신된 메시지를 바탕으로 자신의 홉 수를 결정하고 다시 자신의 이웃노드에게 홉 수를 1만큼 증가시킨 메시지를 송신한다.

홉 수 결정 단계의 클러스터 헤드의 동작

```
if( countnum >=70 && countnum < 80){
    local.packet_flag = 4;
    local.hopcount = 1;
    j=1;
    temp10 = 1; //멀티패스노드 범위
    for(m=1 ;   m<MAX_NODE_NUMBER ; m++) {
        if (keyflag[m]==2){
            local.carrier[j]=nodeid[m];
            temp8=directkey[m];
            for(n=1 ; n < MAX_KEY_RING_SIZE ; n++){
               if (keyid[n] == temp8) {
                  temp9 = keyinfo[n];
                  n = MAX_KEY_RING_SIZE;}            }
            local.carrier2[j]=endecrypt(temp9, temp10);
            j++; }
      }
       //패킷의 끝부분 표시
       local.carrier[j]=0;
       local.carrier2[j]=0;}
```

노드ID	실행 결과 / 내용
CH:8	/opt/tinyos-2.x/apps/TD_EG/1129/TD_EG_CH count: 56 flag: 10 hop : 1 1_1: 10 2_1 : 0 count: 57 flag: 10 hop : 1 1_1: 10 2_1 : 0 count: 58 flag: 10 hop : 1 1_1: 10 2_1 : 0 count: 59 flag: 10 hop : 1 1_1: 10 2_1 : 0 count: 60 flag: 10 hop : 1 1_1: 10 2_1 : 0 count: 61 flag: 10 hop : 1 1_1: 10 2_1 : 0 count: 62 flag: 4 hop : 1 1_1: 1 2_1 : 1 count: 63 flag: 4 hop : 1 1_1: 1 2_1 : 1 클러스터 헤드는 홉 수가 1이 되는 자신의 이웃노드에게 홉 수 결정 메시지를 전송한다
1	/opt/tinyos-2.x/apps/TD_EG/1129/EG_scheme_node1 count: 53 flag: 10 hop: 10 tree : 1 parent : 0 count: 54 flag: 10 hop: 10 tree : 1 parent : 0 count: 55 flag: 10 hop: 10 tree : 1 parent : 0 count: 56 flag: 10 hop: 10 tree : 1 parent : 0 count: 57 flag: 10 hop: 10 tree : 1 parent : 0 count: 58 flag: 10 hop: 10 tree : 1 parent : 0 count: 59 flag: 4 hop: 1 tree : 1 parent : 8 count: 60 flag: 4 hop: 1 tree : 1 parent : 8 클러스터 헤드로부터 홉 수 결정 메시지를 받으면 자신의 홉 수를 1로 저장하고 부모를 클러스터 헤드로 설정한다
3	/opt/tinyos-2.x/apps/TD_EG/1201/EG_scheme_node3 0 parent : 1 nodeid[1] : 1 , directkey[1] : 11, keyflag[1] : 2 Tree : 0 parent : 1 nodeid[2] : 0 , directkey[2] : 0, keyflag[2] : 3 Tree : 0 parent : 1 nodeid[3] : 0 ,nodeid[0] : 0 , directkey[0] : 0, keyflag[0] : 1 Tree 0 parent : 1 1번 노드로부터 홉 수 결정 메시지를 받으면 자신의 홉 수를 2로 저장하고 부모를 1번 노드로 설정한다

4.2.4 멀티 패스 영역 확장 단계

클러스터 헤드는 네트워크의 상황에 따라 멀티 패스 영역 확장을 명령하는 메시지를 발행하여 이웃노드에게 암호화하여 전송하고 이를 바탕으로 각 노드는 복호화 후 다시 이웃 노드에게 명령 메시지를 송신한다.

확장 단계의 클러스터 헤드의 동작

```
// 보내는 패킷의   packet_flag를 5 로 설정
if( countnum >=100 && countnum < 120){
    local.packet_flag = 5;
    local.hopcount    = 1;
    j=1;
    temp10 = 2; //멀티패스노드 범위
    while( nodeid[j] != 0 && j < MAX_NODE_NUMBER ){
                local.carrier[j]=nodeid[j];
                temp8 = directkey[j];
                for(n=1 ; n <  MAX_KEY_RING_SIZE ; n++){
                        if (keyid[n] == temp8) {
                            temp9 = keyinfo[n];
                            n = MAX_KEY_RING_SIZE;}     }
                local.carrier2[j]=   endecrypt(temp9, temp10);
                            j++; }
          //패킷의 끝부분 표시
          local.carrier[j]=0;
          local.carrier2[j]=0;}
```

노드 ID	실행 결과 / 내용
CH:8	/opt/tinyos-2.x/apps/TD_EG/1129/TD_EG_CH count: 97 flag: 10 hop : 1 1_1: 1 2_1 : 1 count: 98 flag: 10 hop : 1 1_1: 1 2_1 : 1 count: 99 flag: 10 hop : 1 1_1: 1 2_1 : 1 count: 100 flag: 10 hop : 1 1_1: 1 2_1 : 1 count: 101 flag: 5 hop : 1 1_1: 1 2_1 : 2 count: 102 flag: 5 hop : 1 1_1: 1 2_1 : 2 멀티패스영역을 홉수가 2인 노드까지 확장하라고 명령하는 메시지를 클러스터의 이웃 노드에게 암호화하여 송신한다.
1	/opt/tinyos-2.x/apps/TD_EG/1129/EG_scheme_node1 count: 93 flag: 10 hop: 1 tree : 0 parent : 8 count: 94 flag: 10 hop: 1 tree : 0 parent : 8 count: 95 flag: 10 hop: 1 tree : 0 parent : 8 count: 96 flag: 10 hop: 1 tree : 0 parent : 8 count: 97 flag: 5 hop: 1 tree : 0 parent : 8 count: 98 flag: 5 hop: 1 tree : 0 parent : 8 클러스터 헤드로부터 확장명령을 받은 1번 노드는 자신의 홉 수와 비교하여 멀티 패스 영역으로 지정되었음을 확인하고 자신의 treenode 변수를 0으로 세팅하여 멀티패스노드로 전환한다.
3	/opt/tinyos-2.x/apps/TD_EG/1201/EG_scheme_node3 0 parent : 1 nodeid[1] : 1 , directkey[1] : 11, keyflag[1] : 2 Tree : 0 parent : 1 nodeid[2] : 0 , directkey[2] : 0, keyflag[2] : 3 Tree : 0 parent : 1 nodeid[3] : 0 ,nodeid[0] : 0 , directkey[0] : 0, keyflag[0] : 1 Tree 0 parent : 1 1번 노드로부터 확장명령을 받은 3번 노드는 자신의 홉 수와 비교하여 멀티 패스 영역으로 지정되었음을 확인하고 자신의 treenode 변수를 0으로 세팅하여 멀티 패스 노드로 전환한다.

4.2.5 멀티 패스 영역 축소 단계

클러스터 헤드는 네트워크의 상황에 따라 멀티 패스 영역 축소를 명령하는 메시지를 발행하여 이웃 노드에게 암호화하여 전송하고 이를 바탕으로 각 노드는 복호화 후 다시 이웃 노드에게 명령 메시지를 송신한다.

축소 단계의 클러스터 헤드의 동작

```
// 보내는 패킷의 packet_flag를 6으로 설정
if( countnum >=140 && countnum < 160){
    local.packet_flag = 6;
    local.hopcount = 1;
    j=1;
    temp10 = 1; //멀티패스노드 범위
    while( nodeid[j] != 0 && j < MAX_NODE_NUMBER ){
            local.carrier[j]=nodeid[j] ;
            temp8 = directkey[j];
            for(n=1 ; n < MAX_KEY_RING_SIZE ; n++){
                    if (keyid[n] == temp8) {
                            temp9 = keyinfo[n];
                            n = MAX_KEY_RING_SIZE;} }
            local.carrier2[j]= endecrypt(temp9, temp10);
            j++;}
    //패킷의 끝부분 표시
    local.carrier[j]=0;
    local.carrier2[j]=0;}
```

노드 ID	실행 결과 / 내용
CH:8	/opt/tinyos-2.x/apps/TD_EG/1129/TD_EG_CH count: 144 flag: 10 hop : 1 1_1: 1 2_1 : 2 count: 145 flag: 10 hop : 1 1_1: 1 2_1 : 2 count: 146 flag: 10 hop : 1 1_1: 1 2_1 : 2 count: 147 flag: 10 hop : 1 1_1: 1 2_1 : 2 count: 148 flag: 10 hop : 1 1_1: 1 2_1 : 2 count: 149 flag: 10 hop : 1 1_1: 1 2_1 : 2 count: 150 flag: 10 hop : 1 1_1: 1 2_1 : 2 count: 151 flag: 6 hop : 1 1_1: 1 2_1 : 1 count: 152 flag: 6 hop : 1 1_1: 1 2_1 : 1 멀티 패스 영역을 홉 수가 1인 노드까지 축소하라고 명령하는 메시지를 클러스터의 이웃 노드에게 암호화하여 송신한다.
1	/opt/tinyos-2.x/apps/TD_EG/1129/EG_scheme_node1 count: 172 flag: 6 hop: 1 tree : 0 parent : 8 count: 173 flag: 6 hop: 1 tree : 0 parent : 8 count: 174 flag: 6 hop: 1 tree : 0 parent : 8 클러스터 헤드로부터 축소 명령을 받은 1번 노드는 자신의 홉 수와 비교하여 멀티 패스 영역으로 지정 되었음을 확인하고 자신의 treenode 변수를 0으로 세팅하여 멀티 패스 노드으로 유지한다.
3	/opt/tinyos-2.x/apps/TD_EG/1201/EG_scheme_node3 1 parent : 0 nodeid[1] : 1 , directkey[1] : 11, keyflag[1] : 2 Tree : 1 parent : 0 nodeid[2] : 0 , directkey[2] : 0, keyflag[2] : 3 Tree : 1 parent : 0 nodeid[3] : 0 ,nodeid[0] : 0 , directkey[0] : 0, keyflag[0] : 1 Tree 1 parent : 0 1번 노드로부터 축소 명령을 받은 3번 노드는 자신의 홉 수와 비교하여 트리 영역으로 지정 되었음을 확인하고 자신의 treenode 변수를 1로 세팅하여 트리 노드로 전환한다.

4.2.6 데이터 수집 및 통합 단계

클러스터 헤드는 데이터 수집 및 통합을 명령하는 메시지를 발행하여 이웃 노드에게 암호화하여 전송하고 이를 바탕으로 각 노드는 복호화 후 다시 이웃 노드에게 명령 메시지를 송신한다. 수신된 메시지를 바탕으로 각 노드는 자신의 노드 성격에 맞추어 정의된 통합과정을 수행하고 부모노드 혹은 이웃 노드에게 암호화 하여 전송한다.

<table>
<tr><td colspan="2">데이터 수집 및 통합 명령시 클러스터 헤드의 동작</td></tr>
<tr><td colspan="2">//보내는 패킷의 packet_flag를 7로 설정
if(countnum >=180 && countnum < 190){
local.packet_flag = 7;
local.hopcount = 100;}</td></tr>
<tr><td>노드 ID</td><td>실행 결과 / 내용</td></tr>
<tr><td rowspan="2">CH:8</td><td>/opt/tinyos-2.x/apps/TD_EG/temp_1129/EG_scheme_node
count: 172 flag: 10 hop : 0 1_1: 0 2_1 : 0
count: 173 flag: 10 hop : 0 1_1: 0 2_1 : 0
count: 174 flag: 10 hop : 0 1_1: 0 2_1 : 0
count: 175 flag: 10 hop : 0 1_1: 0 2_1 : 0
count: 176 flag: 10 hop : 0 1_1: 0 2_1 : 0
count: 177 flag: 10 hop : 0 1_1: 0 2_1 : 0
count: 178 flag: 10 hop : 0 1_1: 0 2_1 : 0
count: 179 flag: 10 hop : 0 1_1: 0 2_1 : 0
count: 180 flag: 10 hop : 0 1_1: 0 2_1 : 0
count: 181 flag: 7 hop : 100 1_1: 0 2_1 : 0
count: 182 flag: 7 hop : 100 1_1: 0 2_1 : 0</td></tr>
<tr><td>클러스터 헤드는 데이터 수집을 명령하는 7번 패킷을 전송하고 데이터 통합 유형을 100으로 지정하여 MAX 값을 구하는 Aggregation 수행을 명령한다.</td></tr>
<tr><td rowspan="2">1,3</td><td>/opt/tinyos-2.x/apps/TD_EG/temp_1129/EG_scheme_node1
node1__packet_flag : 8 , hopcount : 100 carrier: 8 0 carrier2
node1__packet_flag : 8 , hopcount : 100 carrier: 8 0 carrier2
node1__packet_flag : 8 , hopcount : 100 carrier: 8 0 carrier2</td></tr>
<tr><td>데이터 수집을 명령 받으면 미리 정의 되어있는 MAX 함수에 의해서 데이터 통합을 실시하고 수신자를 명시한 carrier2에 맞게 암호화 된 MAX값을 carrier에 담아서 멀티 패스 노드의 경우엔 모든 이웃노드에게, 트리 노드인 경우에는 부모노드에게 전송한다.</td></tr>
</table>

4.3 결과 분석

4.3.1 에너지 사용량 분석

본 논문에서는 에너지 사용량 분석을 위해 총 N개의 센서를 랜덤하게 A(n×n)라는 field에 분포한다고 가정한다. 데이터가 다음 노드로 전달되면 이를 받은 노드는 자신이 받은 데이터들을 모아서 통합 과정을 수행하며 이를 수행하는 방식은 Tributaries-Deltas에서 제안되어 있다. 이 네트워크에서 베이스 스테이션은 고정되지 않은 모바일 에이전트(MA : Mobile Agent)를 가정한다. 이 에이전트는 네트워크 상에서 이동할 수 있으며 데이터는 최종적으로 에이전트에게 보내진다. 에이전트가 실제로 중앙에 있지 않더라도 센서 노드들은 센싱 및 데이터 통합 요청이 네트워크의 중앙에서 이루어진 것으로 보고 라우팅 테이블을 구성한다. 센서 노드는 조밀하게 분포되므로 하나의 홉을 이동할 때마다 에너지 소모량은 동일하게 1로 계산한다. 그러나 클러스터 헤드와 모바일 에이전트 간의 사이와 one-hop 멀티 패스 노드와 모바일 에이전트 사이의 에너지 소모량은 거리의 제곱으로 계산된다. 최종 단계에서 모바일 에이전트로 데이터를 전송하는 것은 클러스터 헤드와 one-hop 멀티 패스 노드이다. [2]에서 제안된 것처럼 네트워크 형성 과정에서 이미 정의된 비율에 의해 멀티 패스 노드가 정해지고 네트워크의 상황에 따라 [2]논문에서 제안한 확률로 계산된 값만큼 확장하거나 감소시킬 수 있다. 전체 네트워크에서 one-hop 멀티 패스 노드의 수는 같다고 가정했을 때 이곳의 에너지 소모량을 제외한 나머지 에너지 (Tributary 영역 + multi-hop Delta 영역)는 전체 노드 수가 동일하기 때문에 에너지 소모량도 동일하다고 간주한다. TD구조에서 one-hop 멀티 패스 노드는 중앙에 모여 있으므로 모바일 에이전트까지의 거리가 같다고 간주한다.

- *클러스터 기반의 Tributaries-Deltas 에너지 계산*

: 총 소모 에너지량 E_c 계산은 다음과 같이 이루어진다.

$$E_c = E_i + \sum_{k=1}^{c} (r_c)^2$$

E_i : CH와 MA 사이의 소모 에너지를 제외한 노드들 사이의 총 소모 에너지.

initial value

c : 클러스터 수

r_c : 각 CH와 MA 사이의 거리

- *Tributaries and Deltas*

: 총 소모 에너지량 E_t 계산은 다음과 같이 이루어진다.

$$E_t = E_i + \sum_{k=1}^{om} (r_t)^2$$

E_i : one-hop 멀티패스노드와 MA 사이의 소모에너지를 제외한 노드들 사이의 총 소모에너지.

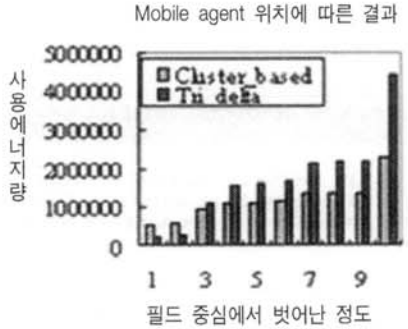
initial value

om : one-hop 멀티패스노드 수

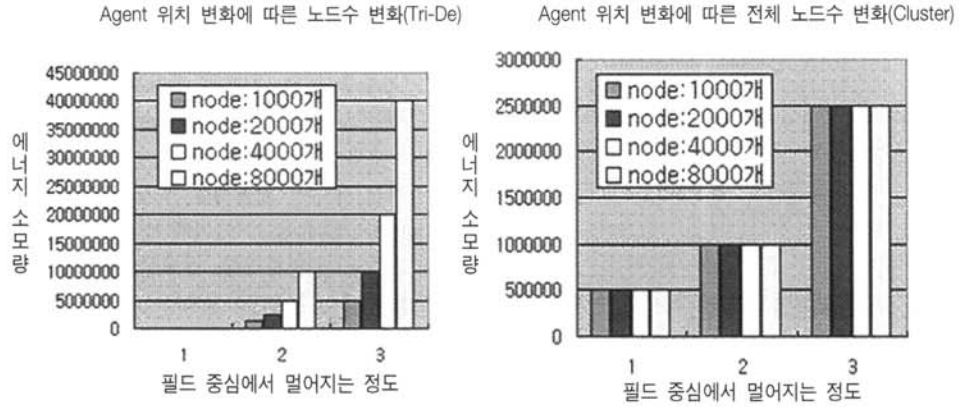
r_t : 필드 중앙 - MA 사이의 거리

Case 1. 모바일 에이전트의 위치 변화에 따른 결과

(그림 6)을 통해 모바일 에이전트가 중앙에서 조금만 벗어나도 본 논문을 통해 제안된 클러스터 기반의 TD 구조가 [2]의 구조보다 현격하게 에너지 효율적임을 알 수 있다. 만약 모바일 에이전트가 네트워크 바깥 영역에 있다면 [2]의 TD구조에서 소모되는 에너지는 클러스터 기반의 TD 구조에 비해 두 배 이상이 될 것이다.



(그림 6) MA 위치에 따른 결과



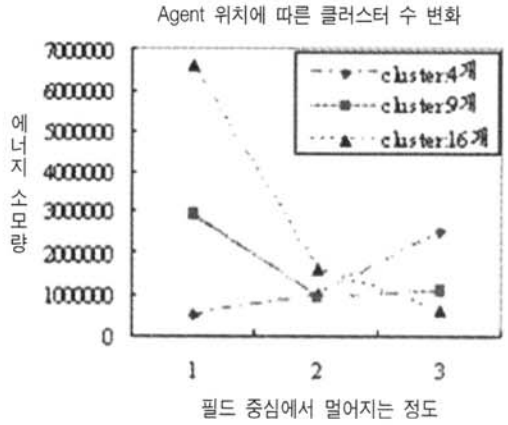
(그림 7) MA 위치에 따른 노드 수 변화

Case 2. 전체 노드 수 증가에 따른 결과

TD 구조에서는 네트워크 내의 전체 노드 수가 증가하면 one-hop 멀티 패스 노드의 수가 증가하게 되고, MA에게 직접 전송하는 노드 수가 증가하게 되는 것이므로 에너지 소모량은 노드 수에 비례해서 증가하게 된다. 그러나 본 논문에서 제안하는 구조에서는 최종 라우팅이 클러스터 헤더에서만 이루어지므로 네트워크의 자체 노드 수가 증가하는 것은 크게 영향을 받지 않는 것을 (그림 7)을 통해 볼 수 있다.

Case 3. 클러스터 수 변화에 따른 결과

클러스터의 수는 무조건 많을수록 좋은 것이 아니라 모바일 에이전트의 위치에 따라 가장 적합한 수가 결정된다. 위의 결과를 보면 에이전트가 중심에서 가까이 있을 경우 클러스터의 수가 적은 것이 유리하며, 멀리 있을수록 클러스터의 수가 많은 것이 좋음을 알 수 있다. 이 사실을 통해 네트워크의 지리적 상황이나 특성을 고려하여 클러스터링을 계획하는 것이 좋다는 것을 확인 할 수 있다. 예를 들어, 험난한 지형의 골짜기와 같이 모바일 에이전트가 접근하기 힘들어서 멀리 떨어진 곳에서 센싱 데이터를 받아야 하는 경우에는 클러스터의 수가 많은 것이 좋고, 모바일 에이전트가 네트워크의 중심까지도 자유롭게 이동할 수 있다면 클러스터의 수가 적은 것이 에너지 효율 측면에서 더 유리할 것이다.



(그림 8) MA 위치에 따른 클러스터 수 변화

4.3.2 Synthetic analysis

베이스 스테이션이 이동성을 가지는 경우 이 모바일 에이전트의 위치에 따라 에너지 소모량은 크게 달라진다. 중심에서 멀어질수록 [2]에서 제안된 TD의 경우 에너지 소모가 매우 심하기 때문에 클러스터 기반의 TD 구조가 효과적임을 알 수 있다. 실제 TD 구조에서 모바일 에이전트의 위치 변화를 모든 노드들이 추적할 수 있다고 가정한다면 delta 영역은 모바일 에이전트의 위치를 향해서 형성될 것이고, 위치가 변하는 대로 delta영역도 변하게 될 것이다. 그러나 만약 안전하지 않은 환경에서 센서 네트워크를 구성하는 경우 TD 구조를 사용한다면 베이스 스테이션의 위치가 대략적으로라도 노출된다는 치명적인 문제점이 발생한다. 게다가 delta 영역이 계속해서 변하기 때문에 트리 노드와 멀티 패스 노드의 변화가 일어나야 하고, 이것은 네트워크에 심각한 지연을 발생시킬 수 있다. 따라서 이러한 환경에서 본 논문에 제안된 클러스터 기반의 방식을 이용한다면 베이스 스테이션의 위치를 노출시키지 않으면서 에너지 효율적인 라우팅이 가능해진다. 클러스터는 미리 구성되어질 수 있지만 동적으로 구성하는 것도 가능하다. 이에 대해 많은 메커니즘이 제안되어 있다. 본 논문의 제안 구조는 에이전트가 이동성을 가지고 움직이는 것은 물론 네트워크 영역 밖에 있는 경우에도 효과적으로 이용될 수 있다.

베이스 스테이션은 전송된 부분적인 통합의 결과를 보고 종합적인 판단을 하게 된다. 만약 전송된 부분 결과의 신뢰성에 문제가 발생했을 경우 본 논문에서 제안하는 방식과 TD 방식은 다르게 동작할 것이다. TD 방식의 경우 전체 네트워크에게 데이터를 다시 요청할 것이고 본 논문에 제안된 방법에서는 해당 지역의 클러스터에게 재전송을 요청할 것이다. 이러한 경우 전체 에너지 소모량을 알아보면 다음과 같다.

• *Tributaries and Deltas*

$$E_t' = 2 \times E_t$$

• *클러스터 기반의 Tributaries-Deltas 에너지 계산*

$$E_c + (E_i / c) + MIN (r_c)^2 \leq E_c' \leq E_c + (E_i / c) + MAX (r_c)^2$$

따라서 위의 결과에서도 모바일 에이전트가 중앙에 있는 경우를 제외하고 대부분의 경우에 본 논문에서 제안하는 클러스터 기반의 TD 방식이 효율적임을 예상할 수 있다.

4.3.3 구현 결과 분석

본 논문에서는 초기의 키 설립이 이루어지기 까지 소요되는 메모리와 RF 프레임 양에 대해서 분석해보았다.

4.3.3.1 RF 프레임량 분석

- Send messages

  2 x Total_Node_Number + Path Key 요청에 대한 답변
- Received messages

  2 x (Total_Node_Number - 1) ≤ Total

  ≤ 2 x (Total_Node_Number - 1) + Path Key 답변 수

4.3.3.2 메모리 사용량 분석

각 노드가 가지는 변수는 MAX_KEY_RING_SIZE와 MAX_NODE_NUMBER와 관련되는 변수, 프로그램을 위해 필수적으로 필요한 변수와 계산을 위해 필요한 임시 변수 등으로 나누어 볼 수 있다. 이 변수들을 토대로 본 구현에서 사용된 메모리량을 분석한 결과는 다음과 같다.

- Total : 774 bits
- RAM : 1014 Bytes
- ROM : 23240 Bytes
- Program size : 23272 Byte

5. 결론 및 향후 연구

본 논문을 통해 [2]에서 제안되었던 TD 구조에 계층성을 높인 클러스터 기반의 Tributaries-Deltas를 제안하여 TD 구조의 효율성과 견고성 뿐만 아니라 계층적 센서 네트워크가 가지는 에너지 효율성을 추가 하였으며 클러스터 기반의 Tributaries-Deltas가 두 가지 상황에서 TD보다 더 좋은 성능을 가짐을 증명하였다. 첫째로 베이스 스테이션이 네트워크에 데이터 재전송 및 통합을 요구할 때이며, 둘째는 BS가 이동성을 가진 모바일 에이전트인 경우이다. 또한 제안된 구조에 적합한 키 설립 메커니즘을 제안하여 에너지 효율성 뿐만 아니라 보안성도 강화시킨 새로운 센서 네트워크 구조를 제안하였다. 제안된 네트워크 구조는 전체 네트워크 영역을 미리 정해진 cluster로 나누어서 CH를 중심으로 하는 TD 구조의 네트워크를 구성하도록 함으로써 TD를 통해서 데이터 전송시 손실율이 작도록 네트워크를 운용하면서 보다 정확한 정보를 클러스터 헤드가 받을 수 있도록 하였고, Delta 영역의 멀티 패스 노드들이 각자 베이스 스테이션과 통신하는 것 대신, 정해진 수의 클러스터 헤드가 베이스 스테이션과 통신하도록 구성해서 데이터 전송시 소비되는 전체적인 에너지를 줄일 수 있었다. 이는 베이스 스테이션이 정해진 위치의 고정 노드가 아닌 이동성을 가진 모바일 에이전트일 경우에도 네트워크 운영이 가능하도록 응용 영역을 확장시키는 결과를 보여준다. 제안된 구조의 에너지 소모량 을 기존의 TD와 비교 분석하였고, 구현을 통해 실현 가능성을 증명하여 Tributaries-Deltas 구조에서 가지고 올 수 있는 장점과 계층적인 구조에서 나타나는 장점을 통합하고 보안성을 제공하는 토대를 마련한 것에 본 논문의 의의

(그림 9) 소모되는 메모리량

가 있다고 할 수 있다. 향후에는 이를 발전 시켜서 보안성을 분석하고 기존의 다른 메커니즘도 도입하여 구현이나 시뮬레이션을 통해 보안성을 검증할 예정이다.

참 고 문 헌

[1] 김신효, 강유성, 정병호, 정교일, "u-센서 네트워크 보안 기술 동향", 전자통신동향분석, 제20권 제1호, pp. 93-99, 2005. 2.

[2] Amit Manjhi, Suman Nath, Phillip B. Gibbons, "Tributaries and Deltas : efficient and robust aggregation in sensor network streams", SIGMOD 2005, pp.287-298, Jun. 2005.

[3] Fei hu, Jason Tillett, Jim Ziobro, neeraj K. Sharma, "An Energy-efficient Approach to Securing Tree-zone-based Sensor Networks," GLOBECOM 2003, vol 3, pp.1430-1434, Dec. 2003.

[4] Jamil Ibriq, imad Mahgoub, "Cluster-based routing in wireless sensor networks : issues and challenges", SPECTS 2004, pp.759-766, Jul. 2004.

[5] J. Spencer, The Strange Logic of Random Graphs, Algorithms and Combinatorics 22, Springer-Verlag, 2000.

[6] L. Eschenauer and V. Gligor, "A Key Management Scheme for Distributed Sensor Networks," Conference on Computer and Communications Security 2002, pp.41-47, 2002.

[7] Mahdi Lotfinezhad, Ben Liang, "Energy efficient clustering in sensor networks with mobile agents", Wireless Communications and Networking Conference 2005, Vol.3, pp.1872-1877, Mar. 2005.

[8] Seema Bandyopadhyay, Edward J. Coyle, "An energy efficient hierarchical clustering algorithm for wireless sensor networks", INFOCOM 2003, Vol.3, pp.1713-1723, Mar. 2003.

[9] Stefan Schmidt, Holger Krahn, Stefan Fischer, and Dietmar W¨atjen, "A Security Architecture for Mobile Wireless Sensor Networks," European Workshop on Security in Ad-Hoc and Sensor Networks 2004, Aug. 2004.

[10] Ruay-shiung Chang, Chia-jou Kuoan, "An Energy efficient routing mechanism for wireless sensor networks", International conference on Advanced Information Networking and Applications(AINA) 2006, Vol.2 pp.308-312, Apr. 2006.

[11] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks," Conference on Computer and Communications Security 2003, pp.42-51, Oct. 2003.

[12] J. Spencer, The Strange Logic of Random Graphs, Algorithms and Combinatorics 22, Springer-Verlag, 2000.

[13] R. Blom "An optimal class of symmetric key generation systems. Advances in Cryptology," Proc. of EUROCRYPT 84, LNCS 209, pp.335 - 338, Apr. 1985.

김 은 경

e-mail : merilin@ewhain.net

2006년 이화여자대학교 컴퓨터학과(학사)

2008년 이화여자대학교 대학원 컴퓨터정보통신학과(공학석사)

2008년~현 재 LG CNS 기술연구부문 전임연구원

관심분야 : 네트워크 보안, 센서네트워크 보안, 유비쿼터스 네트워크보안

서 재 원

e-mail : seojw@ewhain.net

2006년 이화여자대학교 컴퓨터학과(학사)

2008년 이화여자대학교 대학원 컴퓨터정보통신학과(공학석사)

관심분야 : 센서네트워크, 보안, 키 관리 기법 등

채 기 준

e-mail : kjchae@ewha.ac.kr

1982년 연세대학교 수학과(학사)

1984년 미국 Syracuse University 컴퓨터학과(이학석사)

1990년 미국 North Carolina State Univ. 컴퓨터공학과 (공학박사)

1990년 9월~1992년 2월 미국 해군사관학교 컴퓨터학과 조교수

1992년 3월~현 재 이화여자대학교 컴퓨터학과 교수

관심분야 : 네트워크 보안, 인터넷/무선통신망/고속통신망 프로토콜 설계 및 성능분석, 센서네트워크, 홈 네트워크, 유비쿼터스 컴퓨팅

최 두 호

e-mail : dhchoi@etri.re.kr
1994년 성균관대학교 수학과(학사)
1996년 한국과학기술원(KAIST) 수학과 (이학석사)
2002년 한국과학기술원(KAIST) 수학과 (이학박사)
2002년~2007년 한국전자통신연구원 선임연구원
2008년~현 재 한국전자통신연구원 RFID/USN보안안연구팀 팀장
관심분야 : 암호 알고리즘 분석, RFID 보안, USN 보안, 부채널 분석 및 대응법

오 경 희

e-mail : khoh@etri.re.kr
1999년 연세대학교 컴퓨터과학과(학사)
2001년 연세대학교 컴퓨터과학과(공학석사)
2001년~현 재 한국전자통신연구원 RFID/USN 보안안연구팀 선임연구원
관심분야 : USN 보안, RFID 보안, 인증 프로토콜