DOI: 10.3745/KIPSTC.2009.16-C.4.423

센서 네트워크에서 코드분배 메커니즘에 대한 조사 연구

김 미 희[†] · 김 지 선[††] · 김 지 현[††] · 임 지 영[†††] · 채 기 준[††††]

요 약

센서 네트워크는 유비쿼터스 컴퓨팅 구현을 위한 기반 네트워크 중의 하나로 그 중요성이 점차 부각되고 있으며 센서 네트워크 및 노드 특성상 효율성 및 안전성을 제공하기 위한 다양한 기반 기술이 연구되고 있다. 특히 센서 네트워크는 많은 노드 수로 구성되어 있고 많은 응용에서 외부 환경에 무작위 배포로 배치되어 사용되므로 센서 노드에서 실행되는 코드의 기능에 대한 업그레이드나 버그 수정을 위한 코드분배 방법이 원격으로 수행되어야 하며 무선 환경으로 실행코드를 배포해야 하므로 안전성이 특히 중요한 분야라고 할 수 있다. 본 논문에서는 최근 센서 네트워크에서의 새로운 연구 주제로 주목 받고 있는 이러한코드분배 메커니즘들을 조사하여 요구 사항 및 그 특징에 대하여 비교 분석하였다. 이를 통해 센서 네트워크에서의 코드분배 메커니즘에 대한 향후 연구 방향을 제시함으로써 효율성 및 안정성을 제공할 수 있는 코드분배에 관한 연구를 촉진하고 센서 네트워크의 활용도를 제고하자 한다.

키워드 : 센서 네트워크, 코드분배 메커니즘, 효율성, 안전성

A Survey of Code Dissemination Mechanisms on Sensor Networks

Mihui Kim[†] · Jisun Kim[††] · Jeehyun Kim[††] · Jiyoung Lim[†††] · Kijoon CHae[††††]

ABSTRACT

The sensor network is highlighted because it is one of the essentialbase networks in the ubiquitous computing realization. Researches for providing security and efficiency are being perfomed in the various isuues because of the characteristics of sensor nodes and sensor networks. Recently, code dissemination mechanism is recognized as an important research issue since sensor nodes are in the need of updating new software or the need of modifying bugs in dynamically. Generally lots of nodes are in the sensor networks and they are ramdomly deployed in hostile environments. Thus it is especially important that the code dissemination from the base station to nodes should be processed efficienctly and securely. In this paper, we check up the recent existing code dissemination mechanisms, and comparatively analyze the requirements of the code dissemination and the characteristicsof existing mechanisms. Through the analysis, we present future research issues for the code dissemination area. This research can expedite the research on the code dissemination and improve the usability of sensor networks with efficiency and security.

Keywords : Sensor Network, Code Dissemination, Efficiency, Security

1. 서 론

최근 다양한 분야에 주요한 주제로 떠오르고 있는 유비쿼터스 환경의 핵심 기술로서 무선 센서 네트워크의 기술이 주목 받고 있으며, 이에 대한 연구로서 기반 구조에 대한 연구와 효율성 측면의 연구, 그리고 내포된 보안 취약성을 보완하기 위한 안전한 운용 측면의 연구가 진행되고 있다. 특히 이러한 센서 네트워크는 노드 자원의 한계와 많은 노드 수의 구성, 배치 환경 특성으로 인하여 효율성과 안전성을 모두 고려한 메커니즘 고안이 성공적인 무선 센서 네트워크 적용을 좌우할 수 있다. 다양한 센서 네트워크 운용 메커니즘 중에 이미 배치되어 있는 다수의 노드에 수행되고 있는 코드를 원격으로 재분배하는 코드분배 메커니즘이 중요 메커니즘 중의 하나로서 최근 주목받고 있다. 이는 코드의 오류 수정 혹은 버전 업그레이드 등의 목적으로 코드를 분배하는 메커니즘으로 리프로그래밍 메커니즘이라고도 하며, 다수의 노드로 구성되어 외부 환경에 배치된 센서 네트워크에서는 특히 지속적인 운용을 위하여 필수적인 메커니즘이 된다.

센서 네트워크에서의 초기 코드분배 메커니즘은 효율성을 증진하는 측면의 연구가 진행되었다. 해당 연구로서, 첫째 코드분배 노드에서 전송 코드의 양을 줄이기 위한 코드 사이즈 최소화 방안[4, 5], 둘째 효율적인 코드분배[6-8] 프로

※ 이 논문은 2008년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(NO.R01-2008-000-20062-0).
† 정 회 원 : 미국 North Carolina State University Postdoc Researcher
†† 정 회 원 : 이화여자대학교 컴퓨터공학과 석사과정
††† 정 회 원 : 한국성서대학교 정보과학부 조교수
†††† 정 회 원 : 이화여자대학교 컴퓨터공학과 교수
논문접수 : 2008년 12월 26일
수 정 일 : 1차 2009년 4월 16일, 2차 2009년 6월 7일
심사완료 : 2009년 6월 7일

토콜로서 전송 시 에너지 효율을 증진하고 불필요한 제어 메시지 전송을 지양하며 무선 에러율을 고려한 코딩 방법, 마지막으로 센서 노드에서 수신한 코드를 실행 시 동적 링킹을 활용한 효율적인 실행 방법[9,10] 등이 있다.

또한 최근에는 안전한 코드분배 메커니즘에 대한 연구가 진행되어 왔다. 초기에는 분배되는 코드의 무결성과 송신자 인증을 위해 해시체인과 전자서명을 이용한 방법들[11-14]이 제안되었고, 이에 발전된 방법으로서 전자서명의 계산 오버헤드를 줄이기 위한 방법들[15,16]이 제안되었으며, 다양한 공격을 방어하기 위한 메커니즘들[17-20]이 제안되었다.

본 논문에서는 이러한 코드분배 메커니즘들의 특징을 살펴보고 상호 비교를 통해 발전 동향을 분석하며, 이를 통해 향후 연구되어야 할 코드분배 메커니즘들의 주제들을 제시하고자 한다. 이로써 효율성 및 안정성을 동시에 제공할 수 있는 코드분배에 관한 연구를 촉진하고 센서 네트워크의 활용도를 제고하자 한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어 2장에서는 전체적인 센서 네트워크의 특징 및 연구 동향을 정리하고, 3장에서는 본 논문에서 조사 분석하고자 하는 코드분배 메커니즘의 기존 연구 동향에 대해 효율성에 대해 분석 기술하고 4장에서 코드분배 메커니즘의 안전성, 기타 응용에 대해 자세히 분석 기술한다. 5장에서는 향후 연구 방향 및 제언에 대하여 기술하고 마지막으로 6장에서 결론을 맺고자 한다.

2. 센서 네트워크 연구 동향

2.1 센서 네트워크의 특징

무선 센서 네트워크에서는 비교적 저렴한 센서 노드를 대량으로 유기적으로 배치하는 대규모 센서 네트워크의 구현이 용이하다. 센서 노드는 메모리와 저장공간, 에너지가 모두 제한적인 장치이므로 센서로 구성된 네트워크도 자원 제약적인 특성을 갖게 되어 센서 네트워크의 자원 효율성 운용은 중요한 주제 중의 하나이다.

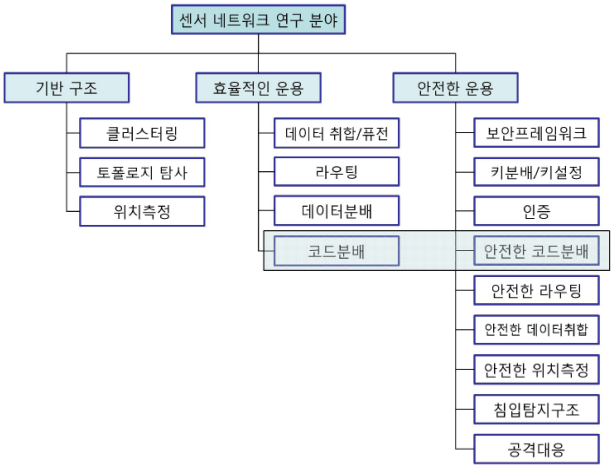
센서 네트워크는 신뢰성이 낮은 무선 통신 채널을 사용함으로 인하여 채널 에러 또는 노드 혼잡으로 인해 패킷이 손실되거나 손상될 수 있고, 전송 중 충돌로 인한 전송 실패가 빈번할 수 있다. 또한 멀티 홉 라우팅과 네트워크 혼잡, 노드 프로세싱은 네트워크 안에 더 많은 대기시간을 만들 수 있다. 이로 인해 센서 노드 사이에 동기화를 하는데 어려움을 주며 결함 발생 시 다른 경로를 통하여 네트워크 기능을 유지해야만 한다.

특히 응용에 따라 무선 센서 노드는 외부에 무작위로 배포되어 배치될 가능성이 있으므로 다양한 보안 측면의 취약성을 내포하고 있다. 그러나 이를 대비한 보안 메커니즘은 센서 노드의 특성으로 인해 경량화 되어야 하며 에너지 효율을 함께 고려해야 한다.

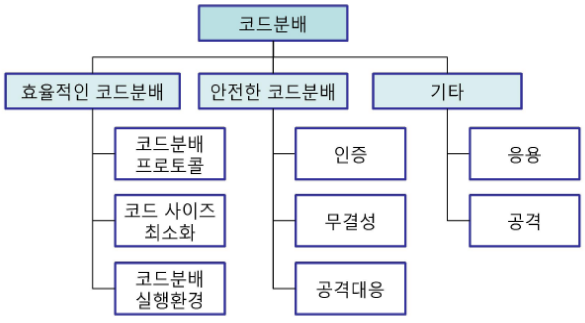
2.2 센서 네트워크의 연구 분야

앞서 설명한 센서 네트워크 특성에 맞추어 진행되어 왔던 주요 연구 주제는 (그림 1)과 같이 크게 3개의 부분으로 나누어 볼 수 있으며 이는 기반 구조 연구, 효율적인 운용, 그리고 안전한 운용에 대한 연구 부분이다. 기반 구조 연구는 많은 노드 수로 구성된 센서 네트워크의 기반 구조를 설립하기 위해 작은 그룹에 해당하는 클러스터링을 구성하는 방법과 스스로 토폴로지를 구성하는 방법, 자신의 위치를 추정하는 방법들이다. 둘째로는 효율적인 운용 방법으로서 많은 센서로부터의 감지된 데이터의 에너지 효율적 전송을 위해 데이터 취합(aggregation)/퓨전(fusion) 방법, 라우팅 그리고 적합한 노드로의 데이터 분배와 코드분배 방법들이 연구되어 왔다. 마지막으로 센서 특성을 고려하여 안전성을 제공하기 위한 보안 프레임워크와 키분배/키설정, 인증, 침입탐지구조, 공격대응 방법을 포함하여 효율성에 기반한 연구들에 안전성을 더하는 안전한 라우팅과 안전한 데이터 취합, 안전한 위치측정, 안전한 코드분배 방법이 연구되고 있다.

본 논문에서 고려하고자 하는 코드분배 방법은 지금까지 (그림 2)와 같이 효율성 측면과 안전성 측면, 기타 부분으로 나뉘어 연구 진행되어 왔다. 3, 4장에서 이에 대해 각 특징을 소개하고 비교 분석하고자 한다.



(그림 1) 센서 네트워크의 주요 연구 주제



(그림 2) 센서 네트워크에서 코드분배 메커니즘의 연구 주제

3. 센서 네트워크의 효율적인 코드분배 메커니즘 분석

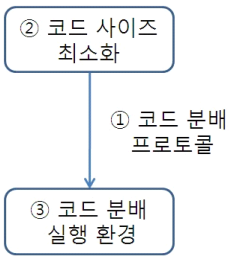
3.1 요구사항

센서 네트워크에서 효율적인 코드분배를 위해서는 오버헤드를 낮추고 남은 리소스에 따라 적절히 운용되어야 하며, 플래시 메모리 리라이팅(rewriting) 및 센서 네트워크 라이프타임의 영향, 프로세싱 등이 최소화되어야 한다. 또한 메모리 리소스 사용을 제한하고 응용프로그램에 대한 인터럽트 및 제어메시지의 전송을 제한해야 한다[23]. 이러한 요구사항은 센서 노드의 특징에 의한 것으로 제한적인 자원의 효율적인 활용에 초점이 맞추어져 있다.

3.2 기존 메커니즘의 특징

코드 분배 시에 에너지 사용에서의 효율성은 자원 제약적인 센서 네트워크 환경에서 중요한 주제 중의 하나이다. 효율성과 관련하여 지금까지 연구된 분야는 (그림 3)과 같다. 첫 번째 연구 분야는 코드 분배를 위한 제어메시지나 분배되는 코드의 양을 줄이고 손실이 발생하더라도 재전송 되는 코드를 최소화하여 전송 에너지를 줄여주는 방법에 관련된 분야이다. 두 번째 연구분야는 전송 에너지를 줄이기 위하여 전송 전에 분배할 코드의 사이즈를 최소화하기 위한 방법에 관련된 분야이다. 세 번째 연구분야는 코드 분배 실행 환경에서 에너지 효율성을 위한 동적 링킹에 관련된 분야이다.

지금까지 제안된 대부분의 코드분배 메커니즘[11-20]은 멀티홉으로 구성된 TinyOS 기반 센서 노드들에 새로운 코드분배를 수행하는 모듈인 Deluge[24]를 기본으로 하고 있다. Deluge는 전송하고자 하는 전체 코드 이미지를 일정한 크기의 페이지로 나누고, 각각의 페이지는 n개의 패킷으로 나누어 전송한다. 페이지 레벨에서는 순서대로 한 페이지씩 전송 완료한 후 다음 페이지 전송을 수행하고 패킷 레벨에서는 전송 노드가 자신의 이웃 노드들에게 페이지 내의 모든 패킷들을 브로드캐스트한다. 이웃 노드들로부터 SNACK (Selective Negative ACK) 메시지를 받은 후 받지 못한 패킷의 집합에 대해서 다시 브로드캐스트하여 전송에 누락된 패킷이 없도록 신뢰성 있는 데이터 전송을 보장한다.



(그림 3) 효율성 관련 연구 주제

3.2.1 코드분배 프로토콜

코드분배 프로토콜에서의 연구는 Deluge에서는 사용하지 않는 Sleep 모드를 이용하여 전송메지시가 없을 때 안테나를 꺼서 에너지를 절약하는 방법과 두 번째로 센서 네트워크에서 각 노드들이 매체를 공유하고 있다는 특성을 활용하여 광고, 요청 메시지 등 제어 메시지전송을 억제하는 방법, 세 번째로 네트워크에 동일한 프로그램을 가지고 있는 다수의 코드 분배자를 이용하여 네트워크를 분할 하여 빠른 시간안에 프로그램을 전송하는 방법, 마지막으로 손실 데이터가 있을 때 엿듣기나 슬라이딩 윈도우를 이용하고, Fountain Code를 이용한 인코딩 방식을 통해 전체 손실된 데이터를 다시 주고 받지 않아도 에러를 복구할 수 있는 방법이 있다. 위와 같은 방식들을 통해 에너지 제약적인 센서 네트워크의 특성을 고려하여 에너지 소비를 줄이는 코드분배 프로토콜에 대한 연구가 진행되었다.

■ *Freshet [6]*

Freshet은 세 단계로 코드분배가 진행된다. 첫 번째 단계에서 노드들은 실제 코드를 보내기 전에 버전정보, 페이지 수, 홉 카운트를 포함한 메타 데이터를 주고 받아 중복 전송을 줄인다. 메시지가 이동할 때마다 홉 카운트는 하나씩 증가하고 임계치 이상이 되면 광고 전달을 멈추고, 더 이상 업데이트 내용을 위한 광고가 들리지 않으면 다음 단계에 들어가기 전에 Sleep 상태로 들어간다. 두 번째 단계는 Deluge와 비슷하게 3방향 핸드셰이크 단계로 코드를 전송하는 단계이다. 이 때, 다른 노드가 전송하는 내용을 엿듣는 방식을 사용하여 광고 메시지나 요청 메시지의 중복 전송을 줄이고 자신이 가장 최근에 들은 광고 메시지로 코드 분배 노드를 선택하여 선택한 노드에게만 메시지를 받는다. 세 번째 단계에서는 노드들이 자신의 전송범위 안에서 완벽하게 코드분배가 끝나면 잠시 동안 새롭게 업데이트 할 내용이 없을 것이라고 예상하고 Sleep 상태로 들어간다.

Freshet은 위에서 설명한 Sleep을 통한 에너지 효율성 이외에 부가적인 특징을 추가였다. 첫 번째는 코드 분배 시간의 효율성을 위해 GPS 같은 하드웨어를 통한 정확한 위치 정보를 사용하여 코드를 분배하는 방식이다. 두 번째는 한 번의 요청으로 다수의 페이지를 전송하는 방식으로 각 페이지에 대한 요청을 계속 해주지 않아도 코드 분배 노드가 적절한 다수의 페이지를 전송해 줌으로써 제어 패킷의 양을 줄여준다. 세 번째는 다수의 코드 분배자가 업데이트 내용을 전송해 주는 방식으로 네트워크에 같은 코드를 가지고 있는 다수의 노드들이 넓은 네트워크를 분할하여 업데이트 내용을 전송하는 방식을 제안하고 있다. 따라서 Freshet은 큰 규모의 센서 네트워크에 적용하였을때 좋은 성능을 보인다.

■ *E3NP [7]*

E3NP는 코드분배 모듈을 미리 외부메모리에 상주하게 하여 코드 분배 시 모듈을 제외한 부분만을 전송하여 불필요한 에너지 낭비를 막을 수 있도록 구성한다. 먼저, 코드분배 노드는 브로드캐스트하여 코드분배의 시작을 알리고 응답으로 각 노드는 부트로더를 호출하여 코드분배 모듈을 프로그램 메모리로 로드시킨다. 호스트는 전송할 이미지를 블록으로 나누어 새로운 버전의 이미지와 이전버전과의 차

이점을 비교하고 새로운 프로그램의 코드 캡슐을 브로드캐스트한다. 각 노드들은 캡슐을 받아서 외부 메모리에 저장하고, 계층적 비트맵에 손실된 패킷을 기록하여 패킷을 관리한다.

손실된 패킷은 쿼리를 통해서 다시 전송이 되고 모든 코드를 받은 후에 코드분배 모듈은 부트로더를 호출한다. 부트로더는 이 코드를 프로그램 메모리로 복사한 후 새로운 사용자 응용 프로그램과 함께 재시작한다. 손실된 패킷의 탐지는 슬라이딩 윈도우와 계층 인덱싱 방법을 이용한다. 캡슐이 성공적으로 받아졌다면 비트를 1로 업데이트하고 빠진 캡슐이 있다면 비트를 0으로 표시하며 중복적으로 받아진 패킷은 버린다. 수신된 캡슐은 외부 메모리에 저장하고 다음 패킷을 받기 위해서 슬라이딩 윈도우를 옆으로 옮긴다.

손실된 패킷을 복구하기 위해 이웃 노드가 멀티홉 데이터 전송을 위해 데이터를 브로드캐스트 할때 자신에게 필요한 패킷이 있는지를 엿들어서 필요한 정보를 복구한다.

■ *Synapse [8]*

Synapse는 재전송이 발생하였을 때 추가로 중복 패킷을 전송하는 HARQ(Hybrid ARQ) 에러 복구 방식을 사용한다. 전송 패킷의 코딩을 위해서 센서 네트워크 리프로그래밍에 적합하게 설계되고 효율적인 인코딩과 디코딩 알고리즘으로 알려져 있는 Fountain Code를 사용한다. 이 코딩 방식은 데이터 Rateless의 특성을 가지며 XOR연산을 통해 인코딩과 디코딩을 수행하기 때문에 연산 오버헤드가 적게 든다.

우선 데이터 코드분배 프로토콜을 진행하기 위해 전체 파일을 전송 블록으로 나누고 전송 전에 HARQ를 적용한다. Rateless 코드는 K개의 원본 이미지를 얻기 위해서 N(≥K)개의 인코딩 패킷을 받아야 하기 때문에 코드분배 노드 i는 a만큼의 중복 패킷을 더해 N(K+a)개의 인코딩 패킷을 주변에 있는 노드들에게 브로드캐스트한다. 브로드캐스트 패킷을 수신자 노드 j가 받으면 디코딩을 위해 필요한 행렬 G를 만들기 위해 필요한 추가적인 중복 패킷이 무엇인지에 대한 정보(NACK)를 i에게 보낸다. i는 통신 범위에 있는 수신자들이 보내는 패킷을 수집한 후 이전에 모든 노드들이 전송받아 재전송이 필요하지 않은 패킷 수의 최소값(b)을 계산하여 K-b+a개의 패킷을 재전송해준다. 이때 필요한 수의 인코딩 패킷을 받을 때까지 전 단계를 반복한다. 모든 인코딩 패킷은 전체 입력 파일을 똑같이 대표하기 때문에 어떤 패킷을 수신하지 못했느냐가아니라 얼마나 많은 패킷이 올바르게 도착했느냐가 중요하다. 디코딩 절차는 최적의 디코딩을 위해 Gaussian Elimination을 사용하여 N개의 패킷 데이터로부터 행렬 G를 구한 후 이를 역변환 하여 원본 패킷을 구한다.

3.2.2 코드 사이즈 최소화

전송 전에 코드 사이즈를 최소화하여 전송하는 연구는 먼저 코드분배 모듈과 응용프로그램을 분리하여 코드분배 모듈을 제외한 응용프로그램만 전송하고 전송 코드의 사이즈를 줄여주는 방법, 두 번째로 전체 코드를 전송하지 않고 변경된 코드만을 전송하는 방법, 세 번째로 업데이트 스크립트를 보낼 때 레지스터와 데이터 할당을 효율적으로 하여 전송 코드를 줄이는 연구가 진행되었다.

이 중 코드만을 전송하는 대부분의 코드분배 메커니즘은 Deluge와는 달리 변경된 부분만 전송하여 전송할 데이터의 양을 줄이는 INP[25]를 기본으로 한다.

■ *Stream [4]*

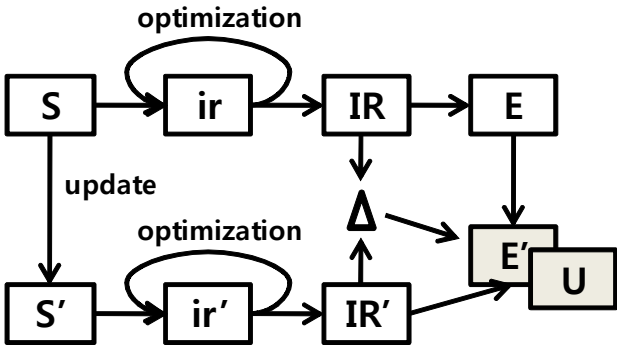
Deluge[24]의 단점을 보완하고자 Stream에서는 프로그램의 이미지를 응용프로그램 이미지와 코드분배 프로토콜 이미지로 나누어서 전송하는 방법을 제안하였다. 응용 이미지는 Stream-AS로 표시하고 이미지 0에 설치되어 작동하며 모든 TinyOS상의 응용에 간단하게 삽입될 수 있다. Stream-AS(Application Support)는 사용자가 재시작 명령을 하거나 새로운 노드가 네트워크상에 도입되었을 때 이미지 0부터 다시 시작하는 기능이 있다. 코드분배 프로토콜 이미지는 Stream-RS(Reprogramming Support)로 표시하고, 이미지 1에 설치되어 작동하며 Deluge에서 사용되었던 3방향 핸드셰이크 과정을 포함하고 노드에 미리 설치되어 있다.

사용자로부터 재시작 명령이 생기면 모든 노드는 이미지 0으로부터 시작된다. 재시작 명령이 모든 노드에게 전해지면 모든 노드는 Stream-RS를 시작하고 Stream-RS는 코드분배를 시작한다. 모든 노드는 자신이 가지고 있는 코드의 페이지를 브로드캐스트하고 이웃노드가 자신의 노드보다 새로운 버전을 가지고 있다면 이웃노드에게 요청 메시지를 보내서 새로운 버전을 받는다. 각 노드는 자신에게 새로운 버전을 요청한 노드의 아이디를 가진 집합 S를 유지한다. 모든 코드가 다운로드 된 후에는 ACK메시지를 보내어 모든 다운로드가 완료되었다는 것을 알린다. 집합 S에 저장되어 있는 아이디로부터 ACK을 받으면 그 아이디를 삭제하고 S가 빈 공간이 되면 이미지 1로 재시작하여 새로운 프로그램으로 실행될 수 있게 된다.

■ *UCC [5]*

UCC는 업데이트된 내용을 전송할 때 변경된 바이너리 코드와 이전버전 사이에 코드 유사성을 향상시켜 전송 업데이트 스크립트를 최소화시킴으로써 에너지 효율성을 증대화하기 위한 컴파일 기술이다.

소스코드를 바이너리코드로 변경해 주기 위한 컴파일은 (그림 4)와 같은 단계를 거친다. 먼저 컴파일러는 소스코드 S를 ir(intermediate representation)로 변환해 주고 ir을 몇 번이고 반복하여 최적의 상태인 IR로 변환해 준다. 마지막으로 IR에 데이터 할당, 코드 배치, 레지스터 할당 등을 적용하여 바이너리 코드 E를 생성한다. 만약 소스코드가 업데이트되어 S′가 되면 위와 같은 똑같은 단계를 거쳐 E′를 생성한다. E와 E′의 차이를 업데이트 스크립트 U라고 하며 코드 분배 노드는 U를 전송하고, 주변 노드가 U를 받으면 U를 이용하여 E′를 생성할 수 있다. 이 때 U의 크기를 줄



(그림 4) 소스코드를 바이너리코드로 변환하는 단계

일 수 있다면 전송량이 줄어들기 때문에 에너지 효율성을 증대시킬 수 있다. 따라서 U를 줄이기 위한 방식으로 IR에서 E로 변환될 때 적용되는 데이터 할당과 레지스터 할당을 효율적으로 하여 Δ을 줄여줌으로써 코드 유사성을 향상시킨다.

3.2.3 코드분배 실행환경

전송된 코드를 받아서 실행하는 코드 분배 실행 환경에 대한 연구는 먼저 UCC[5]처럼 변경된 부분을 받아 노드에서 컴파일하여 이미지를 생성해 내는 방법, 두 번째로 FlexCup[9]처럼 노드에서 베이스스테이션에게 컴파일 된 이미지를 받아 런타임시 링킹을 통해 인스톨을 실행하는 방법, 세 번째로 두 번째 방식에 동적으로 링킹을 추가하여 실행하는 연구가 진행되어왔다.

■ *FlexCup [9]*

FelxCup은 실제로 바뀐 프로그램의 컴포넌트만 전송하기 때문에 전송 시간과 에너지를 절약할 수 있는 방식으로 코드생성단계와 연결단계의 두 단계로 진행된다. 코드생성단계에서는 컴파일된 컴포넌트들을 나타내는 메타 데이터를 생성하여 코드를 업데이트 하는 동안 메타 데이터를 사용해서 실행중인 응용프로그램에 새로운 컴포넌트를 위치시킨다. 연결단계에서는 알맞은 위치에 함수 호출을 재연결시킨 후 데이터 오브젝트의 주소 바인딩을 수행한다. 주소 바인딩은 먼저 응용프로그램을 바이너리 컴포넌트의 형태로 나누고 이 컴포넌트들은 노드 자체에서 연결이 가능하도록 만든다. 새로운 컴포넌트를 기존의 프로그램코드에 통합하기 위해서는 바이너리 컴포넌트의 수와 오프셋, 심볼테이블과 재배치테이블 주소가 포함되어 있는 메타 데이터가 필요하다. 심볼테이블은 모든 컴포넌트에서 쓰이는 글로벌데이터와 심볼기능에 대한 정보가 들어있으며 재배치테이블에는 컴포넌트 코드에서 참조하는 데이터나 기능에 대한 리스트가 있다.

바이너리 컴포넌트를 업데이트할 때 노드에서 수행되는 동작의 링킹과정은다섯 단계로 나누어진다. 첫 번째 단계는 업데이트 데이터를 받아서 외부메모리에 저장하는 단계이다. 이 단계에서는 두 가지의 동작 모드를 가지며 모든 컴포넌트와 메타 데이터를 함께 전송하는 FlexCup Basic 모드와 기존에 저장되어있는 정보와 비교해서 새롭게 바뀐 부분만 전송하는 FlexCup Diff 모드이다. 두 번째 단계는 기존의 심볼 테이블에 새로 받은 심볼 데이터를 합치는 단계로서 램의 임시버퍼를 사용하여 모든 변경된 심볼 정보를 저장한 후에 외부 메모리에 한번에 쓰여진다. 세 번째 단계는 재배치 테이블을 재위치 시키는 단계이다. 각 바이너리 컴포넌트에는 각각의 업데이트된 재배치 테이블을 가지고 있어서 이 새로운 재배치테이블을 알맞은 위치에 복사한다. 네 번째 단계는 참조를 패치하는 단계이다. 모든 컴포넌트의 재배치 테이블의 엔트리를 조사하면서 참조값의 업데이트가 필요한지 체크하여 업데이트가 필요하다면 재배치 테이블에 쓰여져 있는 주소로 이동하여 새로운 목적지 주소값을 쓴다. 이 단계를 거치고 나면 모든 컴포넌트의 참조값들은 올바른 주소를 가리키게 된다. 마지막 단계는 설치와 재부팅 단계로서 프로그램 코드를 외부 메모리에서부터 복사하여 프로그램 메모리에 배치한 후에 재부팅을 하면 업데이트된 프로그램이 실행된다.

■ *Run-Time Dynamic Linking [10]*

Run-Time Dynamic Linking은 동적인 적재 모듈을 사용하여 가상머신코드와 네이티브코드를 함께 사용하여 에너지 효율을 높이는 방법에 대하여 제안하였다. 동적인 적재 모듈에는 Contiki가 있으며 시스템에 적재되는 네이트브 기계코드를 가지고 있다. 이 기계코드는 시스템에서 사용되는 함수나 변수에 대한 참조값을 가지고 있고 이 참조값들은 기계코드가 실행되기 전에 물리주소로 변형되어야 하며 이렇게 변형되는 과정을 링킹이라고 한다. 동적링킹은 시스템의 함수와 변수에 대한 심볼명을 가지고 있으며 모듈이 로드될 때 링킹이 일어난다. 동적링킹의 예로 대부분의 유닉스기반에서 쓰이는 표준 형식인 ELF(Executable and Linkable Format)와 CELF(Compact ELF)가 있다.

동적링킹의 과정은 다음과 같은 네 단계를 거친다. 첫 번째로 ELF/CELF 파일의 코드와 데이터, 심볼테이블, 재배치 엔트리들을 분석하고 관련 있는 정보를 추출하여 저장한다. 두 번째로 코드와 데이터를 위한 메모리를 각각 플래시 ROM과 RAM에 할당하고, 세 번째로 코드와 데이터 조각이 링크되고 알맞은 메모리에 위치가 재배치된다. 네 번째로 코드는 플래시 ROM에 쓰여지고 데이터는 RAM에 쓰여진다.

또한, 자바 가상 머신을 이식하여 네이티브 매소드를 플랫폼에서 실행 가능한 네이티브 코드로 처리할 수 있도록 구성하고 응용프로그램과 가상머신 사이의 중간단계로 Contiki 가상 머신을 두어서 하위 시스템에서제공되는 모든 네이티브 함수를 호출할 수 있다

3.3 기존 메커니즘의 분석

<표 1>은 효율적인 코드분배 메커니즘에 대해 코드분배 프로토콜, 전송코드 사이즈 최소화, 실행시간 최소화로 나누어 비교한 것이다.

먼저 코드분배 프로토콜은 에너지 절약과 빠른 코드분배,

〈표 1〉 효율적인 코드분배 메커니즘의 비교

	코드분배 프로토콜			전송 코드 사이즈 최소화			실행 시간 최소화
	에너지 절약	빠른 코드분배	에러 감내	변경 부분만 전송	응용프로그램만 전송	코드유사성 향상	런타임 링킹
	sleep	다수의 코드 분배자					
Freshet[6]	o	o	o	-	-	-	-
E3NP[7]	-	-	o	o	o	-	-
Synapse[8]	-	-	o	-	o	-	-
Stream[4]	-	-	-	-	o	-	-
UCC[5]	-	-	-	-	-	o	-
FlexCup[9]	-	-	-	o	-	-	o
Run-Time Dynamic Linking[10]	-	-	-	-	-	-	o

에러 감내 세가지 요소로 나눌 수 있다. 에너지 절약을 위해서 Freshet은 다수의 코드 분배자가 새로운 버전 데이터를 전송해 주고 각 단계에서의 조건을 만족하면 노드가 Sleep에 들어가는 것이다. 따라서 이전 방식들에 비해 에너지가 절약되는 효과를 얻을 수 있고 다수의 코드 분배자를 이용하여 빠른 코드분배가 가능하다. 그러나 Sleep에 들어가면 이벤트에 대한 응답을 하지 못하게 될 가능성이 있다는 단점이 있다.

Freshet과E3NP, Synapse에서는 에러 감내를 위해 공통적으로 엿듣기를 사용하고 있다. 특히 E3NP에서는 에러감내를 위해 슬라이딩 윈도우와 인덱싱을 사용하여 메모리 사용량과 패킷손실을 최소화할 수 있는 방안을 제안하였다. 또한 많은 논문의 기본이 되는 Deluge는 ARQ방식을 이용해 에러감내를 하고 있지만 Synapse는 Deluge에서 발전하여 코딩방식과 HARQ를 이용하여 에러 감내를 하고 있다. 코딩방식과 HARQ를 이용하면 하나의 중복 패킷이 파일의 일부를 나타내는 것이 아니라 전체 입력파일을 똑같이 대표하기 때문에 하나의 중복 패킷을 가지고 여러 수신자의 다른 손실을 회복할 수 있다는 장점이 있다.

두 번째로 전송 코드 사이즈 최소화를 위해서 변경된 부분만 전송하는 방식, 응용프로그램만 전송하는 방식, 코드유사성을 향상시켜 전송을 최소화 하는 방식으로 나누어 볼 수 있다. 변경된 부분만 전송하는 방식에는 E3NP와 FlexCup 방식이 있으며 새로운 프로그램의 코드를 전송할 때 전송 사이즈를 줄일 수 있어 에너지 측면에서 매우 효과적이다. 그러나 새로운 버전의 프로그램을 만들기 위해서는 단순히 프로그램에 변경된 부분만을 가지고는 어렵기 때문에 스크립트나 메타데이터와 같은 추가 데이터를 필요로 한다. FlexCup에서는 메타 데이터를 사용하기 때문에 코드 분배 모듈을 미리 저장해놓은 E3NP보다 더 적은 양을 보내게 되어 메모리 사용량이나 전송량에 있어서 보다 더 효율적이다.

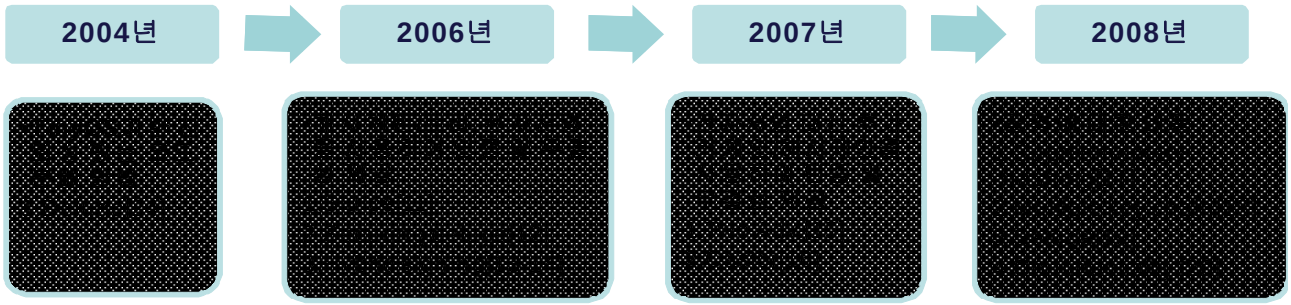
응용프로그램만 전송하는 방식은 Stream과E3NP, Synapse (Stream 사용)에서 사용된다. 코드분배 프로토콜을 노드에 미리 설치해놓기 때문에 전송 시에 작은 부분의 응용 이미지만 전송하여 코드분배 시간과 전송되는 바이트의 수를 줄일 수 있어 에너지 소비가 줄어들며 프로그램 메모리의 사용량을 줄일수 있다는 장점이 있다. 하지만 네트워크 코드분배 모듈이 항상 외부메모리에 상주하기에는 크기가 크다는 단점이 있다.

UCC는 센서 노드에서 수신한 코드와 기존 코드를 컴파일하여 코드 유사성을 향상시킨다. 이 기술을 사용하여 레지스터를 할당할 경우 데이터 전송과 mov와 같은 명령어 실행 사이에 어느쪽이 에너지 소비가 더 많이 이루어지는지를 고려해서 UCC 기술을 적용 할지 여부를 결정할 필요가 있다. 이 기술을 사용할 경우 업데이트 코드 부분의 실행이 적거나 업데이트가 드물게 발생하는 경우에만 효율적이다. 따라서 업데이트 상황을 잘 고려하여 기술을 적용해야 하므로 그 상황을 고려하는 것이 오버헤드가 될 수 있다.

세 번째로 실행시간 최소화를 위해 런타임 링킹을 사용하는 방식은 FlexCup과 Run-Time Dynamic Linking이 있다. FlexCup은 응용프로그램과 시스템 소프트웨어 컴포넌트를 실행 시에 교환할 수 있는 유연성을 제공하고 각 노드에서 전송되는 바이트 수를 줄여서 코드를 업데이트에 소비되는 에너지의 양을 최소화 할 수 있다는 장점이 있다. 하지만 외부메모리와 프로그램 메모리에 충분한 공간이 확보되어 있을 경우에만 가능하다. Run-Time Dynamic Linking의 특징은 자바와 비슷한 가상기계를 사용하는 것이다. 일반적으로 가상머신을 사용하는 것은 수행시간이나 속도 등에서 많은 오버헤드가 들지만 이 메커니즘에서 제안한 네이티브 코드와 가상머신 코드를 함께 사용하면 비교적 에너지 효율성이 좋다. 또한, 동적인 실행시간의 링킹은 네이티브 코드를 업데이트 하는데 사용될 뿐만 아니라 이기종의 네트워크에서도 사용 가능하다는 장점이 있다. 두 방식의 차이는 Run-Time Dynamic Linking이 런타임시에동적으로 링킹이 이루어 진다는 점이다. Run-Time Dynamic Linking방식에서는 JVM(Java Virtual Machine)대신 CVM(Contiki Virtual Machine) 구현하였다. JVM과 CVM 모두 가상기계이므로 노드 수가 늘어나면 에너지 소모량이 선형으로 늘어난다는 단점이 있지만 CVM이 JVM보다는 에너지 소모량이 훨씬 적다는 장점이 있다. 반면, FlexCup은 특정 하드웨어에 의존적이기 때문에 CVM보다 메모리 크기가 작고 높은 속도를 보장해 준다는 장점이 있다.

3. 센서 네트워크의 안전한 코드분배 메커니즘 분석

지금까지 제안된 안전한 코드분배 메커니즘 대부분



(그림 5) 안전한 코드분배 매커니즘의 연구 동향

[11-20]은 Deluge[24]를 기반으로 안전성을 제공하기 위해 기능이 추가되었다. 초기의 메커니즘들[11-14]은 인증 및 무결성을 제공하기 위해 해시트리/해시체인 및 전자서명을 사용하였고, 이후 전자서명의 계산 오버헤드를 줄이기 위해 추가 해시트리나 키체인을 사용하는 방안[15, 16]이 제안되었다. 최근에는 공격에 대한 대응이 가능한 메커니즘[17-20]으로 향상되었다. (그림 5)는 각 연도별 안전한 코드분배 메커니즘의 연구 동향을 도시화한 그림이고, 본 장에서는 지금까지 제안된 각 메커니즘의 특징을 소개하고 비교 분석하고자 한다.

4.1 요구사항

센서 네트워크에서 안전한 코드분배를 제공하기 위해서는 코드분배 메커니즘 자체가 다른 공격이 이용할 수 있는 트랩도어(trapdoor)를 제공해서는 안되며, 무결성, 인증, 프라이버시, 안전한 위임(delegation), 침입 탐지 및 코드분배 실패(failure, corruption)의 탐지 및 감내와 같은 보안 사항이 요구된다[23].

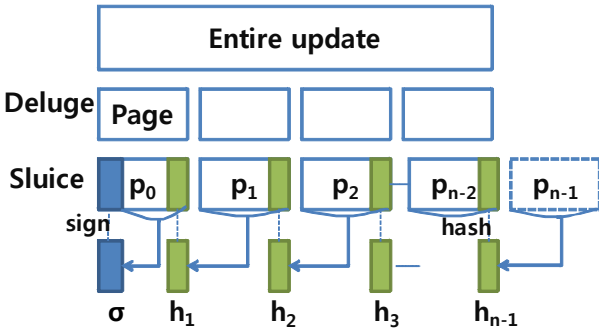
4.2 기존 메커니즘의 특징

4.2.1 인증 및 무결성

■ *Sluice [11]*

코드분배 시, 악의적인 노드에 의해 변형된 코드가 전파되지 않도록 보장하기 위해 단방향 해시체인을 사용하여 코드 이미지의 무결성을 입증하고, 공개키 암호 기법을 사용한 전자서명을 통해 전송자의 인증을 제공하는 코드분배 기법이다.

(그림 6)은 Deluge기반으로 설계된 Sluice의 전자서명과 해시체인을 이용한 업데이트 과정을 보여준다. 먼저 베이스 스테이션은 같은 사이즈로 페이지를 나누고, 각 페이지의 해시값을 그 이전 페이지의 끝에 붙여준다. 그리고 가장 앞에 있는 페이지인 p_0의 경우 p_1의 해시값인 h_1와 합하여 전자서명 σ를 작성하고 이를 맨 앞에 붙인다. 업데이트 페이지를 받은 각 센서는 베이스 스테이션의 공개키를 이용하여 자신이 받은 페이지가 신뢰하는 베이스 스테이션으로부터 전송되었다는 것을 점검할 수 있다. 또한 페이지 p_1부터는 해시함수를 이용하여 각 페이지의 무결성을 검증하며, 단방향 해시체인의 특성상 첫번째 페이지에 포함되어 있는 h_1이



(그림 6) Sluice의 업데이트 과정 - 해시체인과 서명

신뢰할 수 있는 값이라면 나머지 페이지에 포함되어 있는 해시 값도 신뢰할 수 있는 값임을 알 수 있다.

■ *Securing Deluge [12]*

Sluice와 흡사하게 Deluge를 기반으로 인증과 무결성을 제공하기 위한 전자서명과 해시체인을 사용하였지만, 페이지 단위로 해시를 적용하지 않고 각 패킷의 해시값을 바로 앞 패킷에 붙여 보내는 점이 차이점이다.

또한 안전한 코드전파를 위해 서버는 1단계 버전확인 과정을 수행하며, 이는 버전번호를 요청하는 메시지 쿼리 M_{ver}을 브로드캐스트하고, 해당 프로그램 아이디 X_{pid}와 동일한 프로그램을 수행하고 있는 노드가 이전 코드분배 시 서버로부터 받은 M_{adv} 패킷을 전송하여 수행된다. M_{adv} 패킷은 서버 아이디 S, 난수 N, 그리고 난수와 첫번째 데이터 메시지를 합해서 해시한 값인$H(N, M_{D,1})$를 포함시켜, 서버의 개인키로 암호화되어 있어 서버 자신으로부터 전송된 패킷임을 확인할 수 있다. 서버는 수신된 M_{adv} 패킷 중 가장 큰 버전번호의 코드로 2단계 코드분배를 시작한다. 서버로부터의 M_{adv}를 수신한 노드는 전자서명을 검증하고 프로그램의 ID와 패킷의 버전번호를 비교해서, 프로그램의 정보가 일치하며 버전번호가 현재 실행중인 것보다 상위 버전이라면, 이후 패킷은 해시값을 계산하여 비교하고 무결성을 검증하여 코드를 업데이트한다. 두 단계에서 전송되는 패킷은 다음 식과 같다.

- 1단계: 버전확인

$$M_{ver} \triangleq S \rightarrow A: [S, X_{pid}]$$

$$M_{adv} \triangleq A \rightarrow S: [S, X_{pid}, X_{ver}, N, H(N, M_{D,1})]_{SKs}$$

- 2단계: 코드분배

$M_{adv} \triangleq A \rightarrow B$: $[S, X_{pid}, X_{ver}, N, H(N, M_{D,1})]_{SKs}$

$M_{D,1} \triangleq A \rightarrow B$: $X_1, D_1, H(N, M_{D,2})$

$M_{D,i} \triangleq A \rightarrow B$: $X_i, D_i, H(N, M_{D,i+1})$

■ *Hybrid signing [13,14]*

서명된 해시체인(Siged hash chain)에서 손실 패킷(lost packet)으로 인한 인증 지연 및 많은 메모리 사용의 단점을 보완하고자 페이지 단위의 해시트리와 해시체인을 함께 사용한 하이브리드 서명 스킴을 제안하여 인증 및 무결성을 제공하였다.

역시 Deluge를 기반으로 하고 있고, 안전한 데이터 전송을 위해 한 페이지는 (그림 7)에서 보이듯이 g_i패킷을 선두로하여 해시트리로 구성한다. (그림 7)의 page3의 해시트리를 예를 들어 설명하면, 실제 보내고자 하는 코드이미지의 각 패킷은 트리의 최하위 레벨에 순차적으로 배치하고, 각 패킷의 해시값을 수 개씩(이 예제에서는 2개씩) 연결하여 트리의 중간노드인 인덱스 패킷을 작성한다. 이를 반복하여 최종 트리의 루트 $P_{1,0}$까지 완성하면, 이를 해시한 값인 HL_3과 다음 페이지의 g_4의 해시값 HC_3을 연결하여 g_3을 완성한다. 첫 패킷이 되는 g_0는 첫 페이지에 대한 HC_0와 공개키 암호화에 의한 시그니처를 포함하고 있다. 수식에서 *other_info*는 추가되는 부가 정보이다. 관련 식은 다음과 같다.

$g_i = HL_i||HC_i||Other\_page\_info$

$g_0 = E_{Ks}(HC_0||other\_info)||HC_0||other\_info$

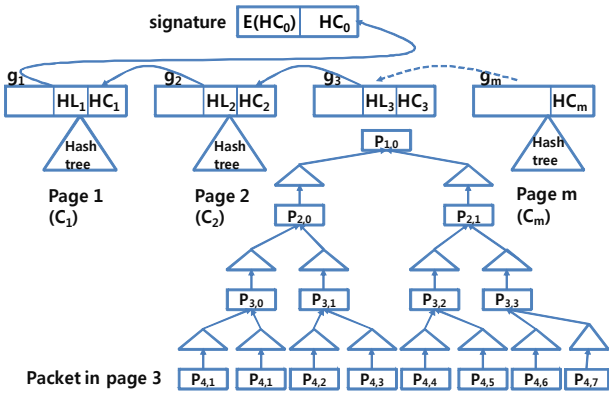
$P_{i,j} = Hash(P_{i+1,j*w})||\cdots||Hash(P_{i+1,j*w+w-1})||other\_info$

$P_{0,0} = E_{Ks}(Hash(P_{1,0}))||Hash(P_{1,0})||other\_info$

4.2.2 계산 오버헤드를 줄인 인증

■ *Two Adv. [15]*

안전한 코드의 전송을 위해서 기존 연구에서는 전자서명과 해시함수를 사용하여 전송을 하였다. 그러나 무선 센서 노드에서 공개키 기반의 전자서명 검증 수행은 많은 계산적 오버헤드를 요구하므로 이를 대신해 해시함수만을 사용하는 메커니즘을 제안하였다.



(그림 7) 하이브리드 서명 스킴

해시체인은 초기값 s로부터 시작하여 체인의 형태로 해시함수를 거쳐서 만들어지며, 마지막으로 만들어진 $p(=h_n)$값은 공개하고 h_{n-1}은 비밀로 가짐으로 해시체인이 완성된다. 모든 노드들은 공개정보인 p값과 해시함수를 가지고 있다. 베이스 스테이션이 노드의 새로운 프로그램에 대한 코드분배를 수행할 때 두 개의 광고 패킷 M_{adv1}, M_{adv2}를 생성하여 전송한다. M_{adv1} 패킷에는 베이스 스테이션의 정보, 프로그램 아이디, 버전 넘버, 난수값, 해시체인의 순서에 대한 정보가 들어있고 이 모든 정보들과 데이터 메시지를 합하여 해시함수를 적용시킨 해시값을 가지고 있다. M_{adv2} 패킷에서는 h_{n-1}과 M_{adv1} 패킷에서 만들어져서 사용되었던 해시값으로 이루어지며 다음과 같이 표현된다.

$M_{adv1}=S, X^{pid}, X^{ver}, N, n-1, hash(S, X^{pid}, X^{ver}, N, n-1, h_{n-1}, M_1)$

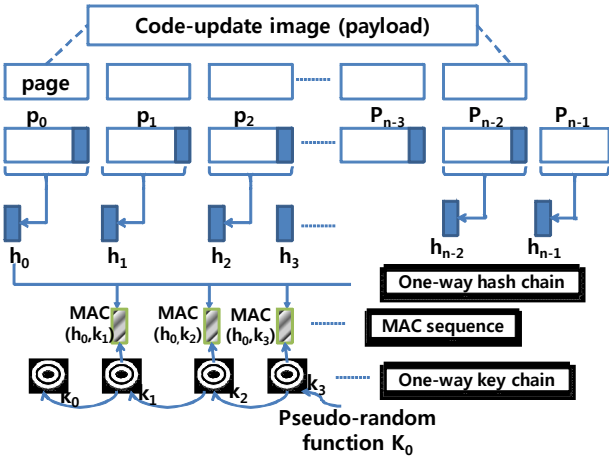
$M_{adv2}=h_{n-1}, hash(S, X^{pid}, X^{ver}, N, n-1, h_{n-1}, M_1)$

만약 노드가 M_{adv1}을 받으면 해시체인의 순서 계수인 n-1을 저장하고, 다른 n-1을 가지고 있는 Madv1들은 모두 무시한다. M_{adv2}를 받으면 해시값을 비교함으로써 검증을 한다. 무결성 체크는 기존 논문과 동일하게 Deluge를 기반으로 이미지를 고정된 사이즈의 페이지로 나눈 후 이 페이지를 해시한 값을 이전 페이지에 포함시킴으로써 수행한다.

■ *Castor: Using Hash chain and Key chain [16]*

인증을 위해 전자서명을 사용하는 Sluice의 계산 오버헤드를 줄이기 위해 비대칭 암호 방식이 아닌, 단방향 해시체인, 단방향 키체인, 그리고 인증을 위한 MAC(Message Authentication Code)을 혼합하여 보안을 제공하는 메커니즘을 제안하였다.

(그림 8)은 h_0의 인증을 위해 다수의 MAC을 사용하는 Castor 메커니즘을 보이고 있다. 이 메커니즘에서는 베이스 스테이션과 모든 센서 노드가 *T*(key가 오픈되는 시간 간격)와 K_0(Castor 키체인의 head), 키함수 *K()*, 해시함수 *H()*를 미리 가지고 배치된다고 가정한다. 베이스 스테이션은 Sluice방식과 같이 코드 업데이트 이미지를 같은 사이즈의 페이지로 나눈 후에 코드 업데이트 이미지의 페이지를 단방향 해시체인을 포함하여 구성하고, 단방향 키체인을 가지고 MAC의 시퀀스를 계산한다. 첫번째 페이지 p_0를 해시한 값에 단방향 키체인으로부터의 유일한 키를 사용하여 MAC을 만든다. 업데이트 과정을 시작하기 전에 베이스 스테이션은 센서 노드에게 MAC을 전송하고, 그 후에 이미지의 페이지를 전송한다. 또한, 베이스 스테이션은 미리 정해져 있는 간격 T에 따라 지연 방식으로 하나씩 키를 공개하고, 센서 노드는 공개된 키를 가지고 받은 MAC을 검증하는 방식으로 보안을 제공한다.



(그림 8) Castor의 업데이트 과정

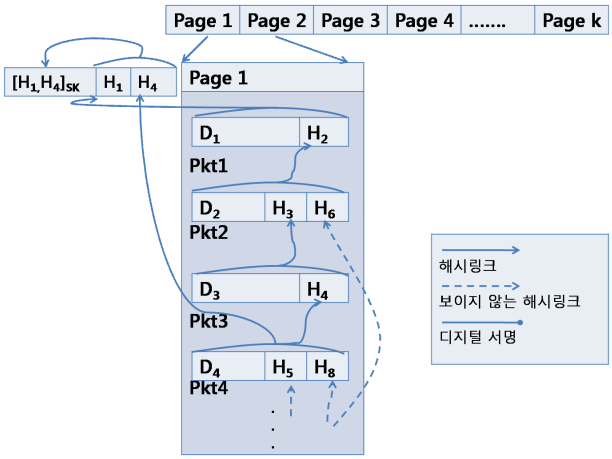
4.2.3 공격에 대한 대응

■ *Supplymentary Hashing [17]*

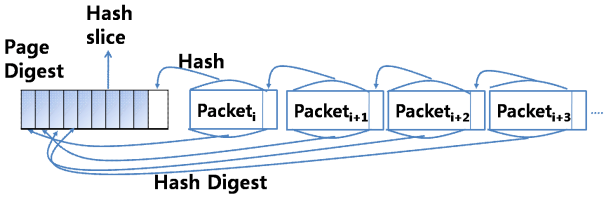
기존의 연구는 오직 프로그램 이미지의 인증과 무결성에 초점을 맞추어서 진행이 되었다. 하지만, 손실된 패킷에 의해 혹은 이를 유발하는 공격에 의해 인증 지연 문제가 발생할 수 있으므로, 이 문제를 해결하기 위해 중복 해시(Redundant Hash) 스킴과 페이지 다이제스트(Page Digest)를 적용한 스킴을 제안하였다.

중복 해시 스킴은 (그림 9)와 같이, Secuing Deluge[12]와 흡사하게 패킷단위 해시체인을 형성하되, 한 패킷의 해시값을 이전 패킷에 붙이고, 또한 어떤 패킷은 자신의 해시값을 이전 패킷뿐 아니라 더 앞의 패킷에도 중복하여 붙여서 패킷을 만든다. (그림 9)에서는 s=2, d=4일때의 경우로, s번째 패킷인 pkt_2, pkt_4, …은 다음 패킷의 해시값 뿐아니라 d번째 다음 패킷의 해시값을 붙여 패킷을 형성한다. 첫 번째 패킷에서는 해시값에 대하여 전자서명을 한 후에 전송하여 인증을 수행한다. 이렇게 중복으로 해시를 만들어 붙임으로써 패킷 손실에 따른 인증 지연을 줄이게 된다.

페이지 다이제스트 스킴은 (그림 10)과 같이 패킷단위 해



(그림 9) 중복 해시 스킴



(그림 10) 페이지 다이제스트 스킴

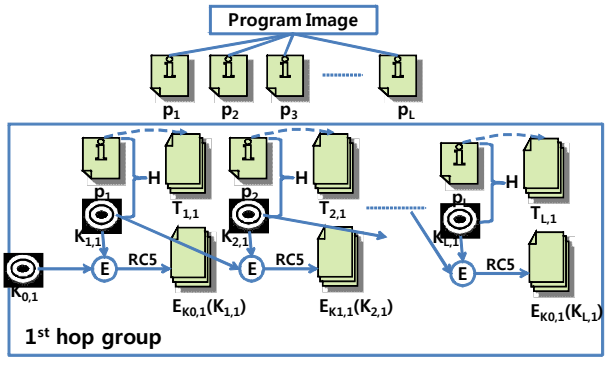
시체인을 통해 패킷을 형성하되, 이러한 해시값들을 모아 페이지 다이제스트를 만들고, 한 페이지의 전송 마지막에 다음 페이지의 페이지 다이제스트를 먼저 전송하여 패킷 손실에 의한 전송지연을 피할 수 있다.

■ *Muliple Key Chain [18]*

Supplymentary Hashing[17]과 마찬가지로 전자서명의 생성 및 검증의 오버헤드를 없애고, 멀티홉 시나리오에서 송신자의 완전한 인증이 가능하도록 안전한 멀티홉 네트워크 프로그래밍 프로토콜을 제안하였다.

센서 노드가 배치되기 전에, 베이스 스테이션은 S개의 랜덤 시드(IV: Initial Value)를 생성하고, IV에 L번 해시함수 H()를 적용하여 각 IV별, L+1 길이의 해시체인 생성한다. i번째 키체인의 CV(Committed Values)인 $K_{0,i}$는 i번째 홉그룹의 노드들에게 선분배 되어 키체인의 확인값으로 사용된다. 센서 노드는 베이스 스테이션에서의 홉 거리에 따라 S개의 그룹으로 나뉘게 된다.

베이스 스테이션에 새로이 전송할 업데이트 프로그램이 생기면, 코드분배를 위해 패킷 선처리와 검증과정을 거친다. (그림 11)과 같이 먼저 프로그램 이미지를 L개 패킷으로 분할하고, 패킷 선처리를 위해 키업데이트 세그먼트 $U_1=E_{K0}(K_1)$과 패킷인증 세그먼트 $T_1=H(P_1||K_1)$를 만든다. 각 그룹을 위한 패킷을 선처리한 후, $P_1'=(P_1||\{Ui||Ti\}||U_1||T_1)$를 첫 번째 홉 그룹에 있는 노드들에게 전송한다. 첫번째 홉 그룹의 각 노드는 P_1'중에서 U_1을 통해 K_1을 검증하여 송신자 노드의 인증을 수행하고, 검증된 K_1을 가지고 T_1를 통해 첫 번째 패킷인 P_1의 무결성을 체크할 수 있다. 해당 패킷이 검증되면 K_0을 K_1로 대체하여 다음 패킷의 검증을 위해 사용한다. 자신이 두번째 홉그룹에 포워딩 노드가 되면 P_1'에



(그림 11) 다수의 단방향 키체인 생성

서 ($U_1||T_1$)를 제거하여 두번째 홉그룹으로 전송하여 같은 검증과정을 거치게 된다.

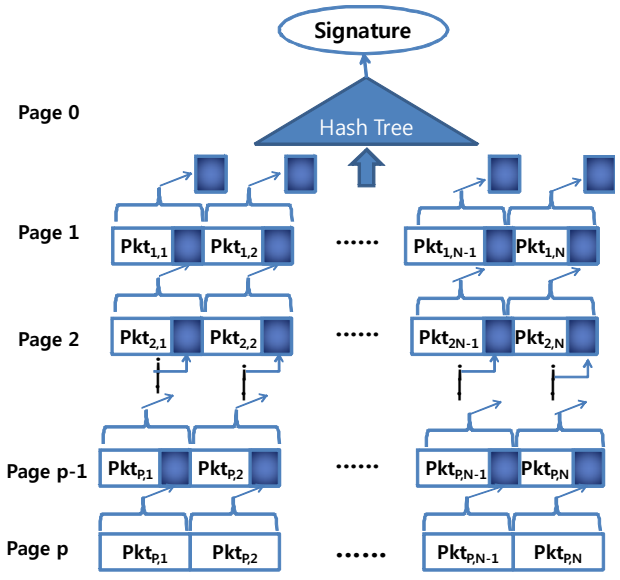
■ *Seluge [19]*

다른 기존 메커니즘에서 고려하지 못한 시그니쳐 패킷과 광고 패킷, SNACK 패킷에 대한 DoS공격에 방어 가능하고, 한 페이지의 각 패킷 해시값을 바로 앞 페이지의 각 패킷에 포함시킴으로써 순서화되지 않은 패킷 수신으로 인한 인증 지연이 없다.

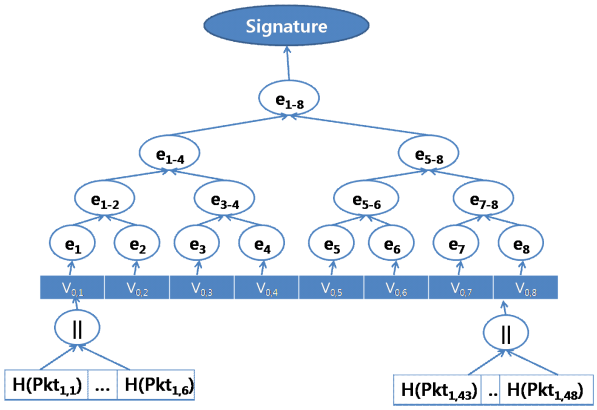
(그림 12)와 같이 코드 이미지를 고정된 사이즈의 페이지로 나누고, 각 페이지를 고정된 사이즈의 패킷으로 나눈다. 각 패킷들의 해시값들을 이전 페이지의 해당 패킷에 포함되며, 이 과정을 반복적으로 시행한다. 첫 번째 페이지의 패킷들을 해시한 후 이 해시결과를 가지고 머클해시트리를 형성한다. 이 머클해시트리의 각 부모 노드들은 자식 노드들을 연결한 후 해시함수를 적용시킨 값들로 구성되어 있으며, 트리의 루트에서 전자서명이 일어난다. 페이지 0의 패킷 $Pkt_{0,i}$는 $V_{0,i}$와 머클해시트리의 인증패스의 노드값으로 구성되며 예를 들어 (그림 13)에서 $Pkt_{0,1}$은 $V_{0,1}$과 e_2, e_{3-4}, e_{5-8}로 구성된다.

페이지 광고(Page Advertisement)와 SNACK 패킷의 인증을 위해 클러스터키를 이용한다. 이웃한 노드들끼리는 공유키를 가지고 배치되고, 헬로우 패킷에 의해 이웃 노드로서 인식이 되면 클러스터키를 생성하여 공유키를 통해 안전하게 전달한다.

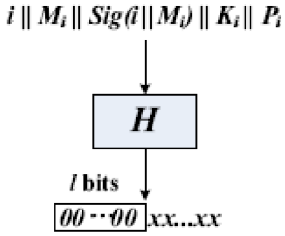
시그니쳐 패킷에 대한 DoS공격을 막기 위한 방법으로 (그림 14)와 같은 Message Specific Puzzle을 도입하였다. 이 메커니즘의 준비단계에서 베이스 스테이션은 랜덤한 키값 Kn을선택하고, 그 Kn값에 단방향 해시 키체인을 적용시켜서 Puzzle Key인 Ki를 형성하여 배치전 센서에게 K0를 선분배한다. 각 코드이미지의 버전과 서명된 패킷들의 집합



(그림 12) 해시체인



(그림 13) 머클 해시트리



(그림 14) Message Specific Puzzle

을 합쳐서 서명을 한다. 준비단계에서 생성되었던 Ki값과 연결하여 Message Specific Puzzle을 생성한다. 여기에서 Mi는 (그림 13)의 머클해시트리에 의해 생성된 시그니쳐 패킷 필드들의 집합이고 Pi는 i||Mi||Sig(i||Mi)||Ki||Pi 값을 해시 함수에 넣었을 때 1개의 0비트가 나오도록 만드는 값이다. Message Specific Puzzle을 해시 함수에 집어넣고, 그 해시의 결과로 나오는 0의 길이에 따라서 퍼즐의 세기가 결정된다. 마지막으로 베이스 스테이션은 서명된 이 i||Mi||Sig(i||Mi)||Ki||Pi을 브로드캐스트 한다. 서명된 패킷을 받은 후에는 각 노드들은 해시 함수 H와 Ki을 이용하여 받은 Puzzle Key Ki가 유효한 키인지 Puzzle 패킷을 해시하였을 때, 1비트의 0값이 나오는지 검증한다. 이로써 위조된 시그니쳐 패킷을 이용한 DoS공격에 대응할 수 있다.

■ *Confidenciality [20]*

Sluice[11]의 공개키 연산의 오버헤드와 업데이트 이미지의 노출이라는 단점을 보완하기 위해, 기밀성과 효율성을 고려하여 랜덤키 사전분배 방식을 통해 그룹키 분배를 이용한 코드분배 방법을 제안하였다. 이 메커니즘은 노드간의 안전한 인증을 보장하기 위해 랜덤키 사전분배 방식을 사용하였다. 각 노드는 배치되기 전에 복수의 키를 가진 키 풀(pool)로부터 키의 집합을 랜덤하게 할당받고, 모든 노드는 자신의 키 집합의 키ID를 브로드캐스트한다. 이웃 노드와 자신이 공유하고 있는 키를 알게 되고, 이 공유키를 사용하여 안전한 링크를 설정한다. 만약 키를 공유하고 있지 않다면 노드 쌍들은 그들만의 유일한 키를 생성하여 나누어 가진다.

랜덤 키 사전분배 방식은 클러스터링 과정에 포함이 되어 진행된다. 첫번째 과정은 클러스터 헤더와 센서 노드간에 공유할 수 있는 그룹키(SK_{intra})를 분배하는 과정이다. 베이스 스테이션이 모든 노드에게 업데이트를 알리는 광고 메시지를 전송하고, 클러스터 헤더는 가장 많은 노드들이 공유하는 사전키를 그룹키로 정한다. 두 번째 과정은 베이스 스테이션과 클러스터 헤더간에 공유하는 그룹키(SK_{inter})를 분배하는 과정이다. 이 그룹키는 베이스 스테이션에서 생성되며 사전 분배된 키를 사용하여 암호화 된 후에 클러스터 헤더에게 전송된다. 이때 전송되는 그룹키는 베이스 스테이션에서 생성된 128비트의 난수를 이용한다. 세번째 과정은 분배된 그룹키를 사용하여 업데이트를 암호화하여 전송하는 과정이다. 베이스 스테이션은 SK_{inter}를 사용하여 클러스터 헤더에게 암호화된 코드를 보내고, 이를 받은 클러스터 헤더는 이를 복호화한 뒤 SK_{intra}를 사용하여서 암호화하여 노드들에게 전송한다.

4.3 기존 메커니즘의 분석

지금까지 소개한 기본 메커니즘을 정리하여 분석해 보면, 2004년 동일한 크기의 페이지단위의 전송 확인 및 SNACK 패킷을 통해 전송 실패된 패킷의 신뢰성 있는 패킷 전송을 가능하게 하는 Deluge[24]를 기반으로 안전성을 제공하기 위해 기능들이 추가 보완되어 제안되었다. 이러한 연구의 시초로 Sluice[11]는 무결성 및 인증을 제공하기 위해 해시 트리와 전자서명을 사용하며, 모든 페이지가 아닌 맨 앞의 페이지만 전자서명을 하여 검증의 오버헤드를 줄이고자 하였다. 그러나 이 방법은 한 페이지의 모든 패킷을 수신한 뒤 무결성 체크 및 인증을 수행할 수 있어서, 플러딩(flooding) DoS공격에 취약하게 된다. 이와 비슷하게 Secuing Deluge[12]에서는 인증 및 무결성을 제공하며 각 페이지 단위가 아닌 각 패킷 단위의 해시값을 제공하기 때문에 패킷이 순서대로 전달된다면 수신 뒤 바로 인증이 수행될 수 있지만, 그렇지 않다면 바로 앞의 패킷이 전송될 때까지 인증 지연 및 버퍼링 오버헤드가 요구된다. Hybrid signing[13-14]에서는 수신 순서에 따른 인증 지연을 방지하기 위해 패킷 전송 시 각 트리의 레벨 전송이 완료된 이후 다음 레벨의 패킷을 전송하도록 하였다. 그러나 많은 인덱스 패킷의 생성 및 전송에 의해 소비 에너지가 증가될 수 있다.

초기 메커니즘에서 인증 제공에 사용되었던 전자서명 검증의 많은 계산 오버헤드를 제거하기 위해 별도의 해시체인을 사용하는 Two Adv.[15]을 제안하였고, 이는 계산 측면에서 더 효율적일 수 있다. 이와 비슷하게 전자서명 검증을 제거하고 별도의 키체인을 사용하는 Castor[16]는 오버헤드를 많이 줄일 수 있지만, 간격 T마다 키를 공개하기 때문에 네트워크에 있는 모든 노드의 시간 동기화가 필요한 단점이 있다. 그리고 T를 적절히 조정하는 것이 중요한데, 만일 T가 크다면 키가 공개 되기 전에 MAC을 풀어보고 검증할 수 없으므로 지연이 생기게 되고, T가 작다면 모든 노드가 MAC을 받기 전에 키가 미리 공개되어 MAC을 인증할 수 없다.

최근에는 무결성 및 효율적인 인증을 포함하여 다양한 공격에 대응할 수 있는 메커니즘이 제안되었으며, Supplymentary Hashing[17]은 중복적으로 해시를 패킷에 포함시켜 패킷 손실이나 순서화되지 않은 수신과 이를 유발하는 DoS공격에 대해 인증 지연과 저장 오버헤드를 줄일 수 있게 하였다. 그러나 중복된 해시값에 의해 전송량이 증가하게 된다. 또한 멀티홉으로 코드분배가 수행될 때 베이스 스테이션 뿐만 아니라 포워딩을 수행하는 송신자의 인증을 지원하기 위해 Muliple Key Chain[18]이 제안되었다. 각 홉그룹에 대한 검증값을 차례로 붙여 전송하여 각 홉그룹에서는 자신의 검증값들을 제거하여 전송하기 때문에, 시빌(Sybil)공격이나 웜홀공격을 방어할 수 있다. 그러나 이로 인해 베이스 스테이션에 가까운 홉그룹에서는 더 많은 전송량을 요구하고, 순서화되지 않은 패킷 전달에 의한 지연은 고려하지 않고 있다는 단점이 있다. 다른 메커니즘에서 고려하지 못했던 페이지광고 메시지와 SNACK 메시지에 대한 인증을 제공하고, 시그니쳐 패킷에 대한 DoS공격을 방어하기 위해 Seluge[19]가 제안되었으며, 순서화되지 않은 패킷 수신에 의해 인증 지연은 없지만 인증을 위해서 각 노드마다 이웃노드와의 pairwise 클러스터키가 필요하다.

그러나 지금까지의 공격 대응 방안들은 재밍공격이나 배터리소모공격과 같은 DoS공격은고려하지 않았고, 전송되는 코드의 기밀성은 제공하지 못한다. 이에 기밀성 제공을 위해 공개키 기반의 인증 방식을 사용하지 않고 랜덤키 사전분배 방식을 사용하여 코드의 노출을 막는 Confidenciality[20]기법이 제안되었다. 그러나 이 방법도 하나의 클러스터 헤더 노드가 탈취되면 전체 센서 네트워크의 보안이 위협받을 수 있고, 노드마다 관리해야하는 키의 수가 증가되었다는 단점이 있다.

특히 최근에는 공격들에 대한 대응이 가능하도록 설계되었으며, Supplymentary hashing[9]은 패킷 손실 즉 순서화되지 않은 패킷 수신을 유도하여 인증 지연을 유발하는 공격을, Multiple key chain[10]은 시빌공격과웜홀공격을, Seluge는 패킷손실공격 및 제어 패킷과 시그니쳐 패킷에 대한 DoS공격을, Confidentiality[12]는 도청에 의한 코드노출에대응할 수 있도록 추가 기능이 고안되었다. 각 메커니즘들이 대응 가능한 공격에 대해서는 <표 2>에 정리하였다.

앞서 이야기 한 것처럼 초기의 안전한 코드분배 메커니즘들은 Deluge를 기반으로 전자서명과 해시체인/트리를 이용하여 인증과 무결성 체크 기능을 추가하였다. 이로 인해 추가된 계산 오버헤드는 다음 <표 3>과 같다.

전자서명 및 해시코드 검증시간은 사용된 알고리즘과 구현방식, 사용된 센서노드의 처리용량에 따라 다르지만, Sluice에서의 실험한 시간에 의해 ECDSA에 의한 전자서명 검증시 30초, 160bit SHA-1에 의한 해시코드 검증 시 0.2초가 소요된다고했을 때, Deluge에 비해 (그림 15)와 같은 계산 오버헤드 시간이 소요된다. 단 Hybrid signing는 해시체인과 해시 트리를 동시에 사용한 메커니즘을 가정했고, w는 해당 논문의 예문처럼 2로 가정하였다. Deluge에 비해 추가

〈표 2〉 안전한 코드분배 메커니즘들이 대응 가능한 공격들

공격	대응가능 메커니즘
	[11~19]
위조된 페이지 전송	[11~19]
패킷손실 즉 순서화되지 않은 패킷 수신을 유도하여 인증지연을 유발하는 공격	[17],[19]
시빌/웜홀 공격	[18]
시그니쳐패킷/광고패킷/SNACK패킷에 대한 DoS공격	[19]
도청	[20]

〈표 3〉 초기 안전한 코드분배 메커니즘의 계산 오버헤드 및 추가 메시지 수

안전한 코드분배 메커니즘	추가된 계산 오버헤드	추가된 메시지 수
Sluice[11]	tSig+tHash*nPage	(nPage*lHash+lSig)/lPkt
Securing Deluge[12]	tSig+tHash*nPage*nPkt	(nPage*nPkt*lHash+lSig)/lPkt +1
Hybrid signing[13]	tSig+tHash*nIdx ≈tSig +tHash *(nPage*nPkt/(w-1))	nPage+nIdx+1 ≈nPage*(1+nPkt/(w-1))+1
Castor[16]	2*tHash*nPage	(nPage*lHash)/lPkt+nPage

- tSig: 전자서명 검증시간
- tHash: 해시코드 검증시간
- nPage: 페이지 수
- nPkt: 페이지당 패킷 수
- nIdx: 전체 인덱스 패킷 수
- w: 인덱스 패킷 생성 시, 한 인덱스 패킷당 포함되는 해시코드 수

된 메시지는 Sluice에서 가정한 ECDSA에 의한 시그니쳐 44Bytes, 해시코드는 20Bytes, 한 페이지의 패킷 수는 Deluge의 기본48패킷-페이지로, 한 패킷은 TinyOS의 기본 페이로드(Payload) 사이즈인 29Bytes로 가정했을 때, (그림 16)과 같다. 예상대로 많은 해시코드 생성으로 Hybrid Signing 방식이 페이지 수가 늘어날수록 계산 오버헤드가 커지고 메시지 수도 인덱스 패킷 생성으로 크게 증가하였다. 또한 페이지 당 하나의 해시코드만 추가하는 Sluice 방식이 제일 작은 오버헤드와 추가 메시지 수를 보이고 있다. 그러나 Hybrid의 이러한 오버헤드로 인해 순서화 되지 않은 메시지 수신에도 바로 무결성 체크가 가능하게 된다.

Caster[16]는 Sluice에서 전자서명의 계산 오버헤드를 줄이기 위해 추가 키 체인에 의한 MAC 메시지 전송으로 인증 기능을 제공하고 있다. 이에 대해 Sluice와 비교한 계산 오버헤드와 추가 메시지 수는 (그림 17), (그림 18)과 같다. 계산 오버헤드는 Sluice보다 작지만 MAC 메시지 전송에 의한 추가 메시지 수는 많음을 알 수 있다.

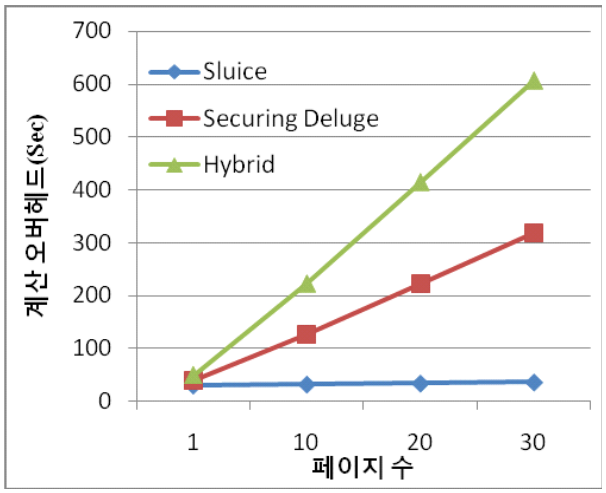
최근 제안된 안전한 코드분배 메커니즘[17-20]들은 다양한 공격에 대응하기 위해 추가적인 오버헤드를 요구하게 되었다. Supplymentary[17]의 경우 순서화되지 않은 패킷 수신에 대비하여, 바로 무결성 체크가 가능하도록 패킷당 해시코드를 추가하는 Securing Deluge보다 더 많은 해시코드를 추가하였고, Seluge[19]의 경우에도 다양한 DoS 공격에 대응 가능하도록 별도의 머클해시트리를 사용하였다. 그러나 이러한 추가적 장치에 의한 오버헤드는 센서노드 장치가 점점 발전할수록미비한 오버헤드로 작용할 수 있으므로 공격에 대한 안전성을 추구하는 추세로 연구가 진행되고 있다.

<표 4>는 최근까지 제안된 안전한 코드분배 메커니즘에서 보안기능 제공 시 사용된 기법들과 각 메커니즘의 단점을 정리 비교한 표이다.

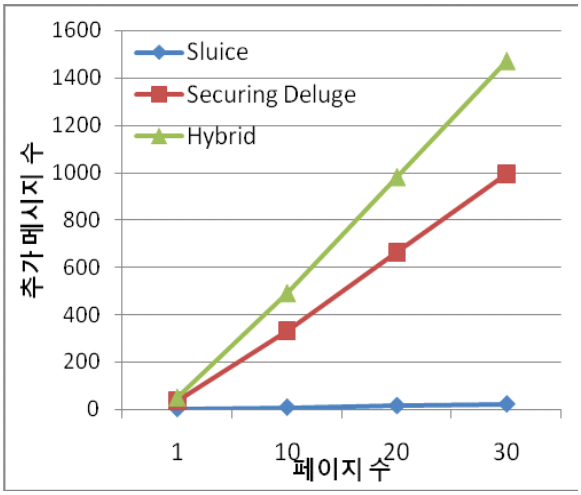
코드분배 메커니즘 자체에 대한 연구 외에도 이를 활용한 코드분배 응용에 대한 연구와 센서노드의 코드 취약점을 악용한 코드삽입공격에 대한 연구도 진행되었다.

■ *응용: 코드분배 이용한 보안 컴퍼넌트 변경 [21]*

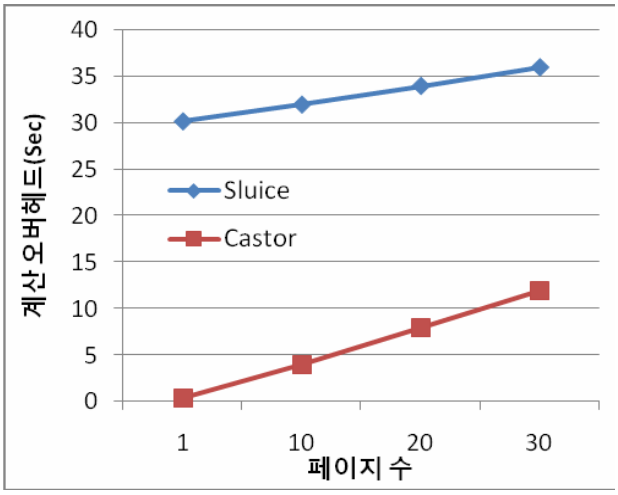
센서 네트워크 운영 시기에 따라 필요한 암호시스템을 기



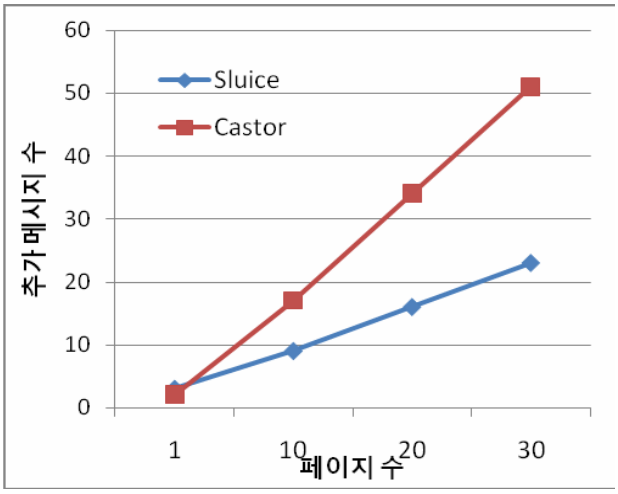
(그림 15) 페이지 수 당 추가 계산 오버헤드



(그림 16) 페이지 수 당 추가 메시지 수



(그림 17) 페이지 수 당 추가 계산 오버헤드



(그림 18) 페이지 수 당 추가 메시지 수

〈표 4〉 안전한 코드분배 메커니즘의 비교

	무결성	인증	기밀성	단점
Sluice [11]	페이지단위 해시체인	첫페이지에 대한 전자서명	-	■ 전자서명의 오버헤드 ■ 전체 페이지 수신후 인증가능하므로 플러딩 DoS공격 취약 ■ 코드 노출
Securing Deluge [12]	패킷단위 해시체인	첫패킷에 대한 전자서명	-	■ 전자서명의 오버헤드 ■ 패킷 수신 순서가 다른 경우 버퍼링 오버헤드 및 인증 지연 ■ 코드 노출
Hybrid Signing [13,14]	패킷단위 해시트리 및 해시체인	첫패킷에 대한 전자서명	-	■ 해시트리 구성 시 생성된 많은 인덱스 패킷 전송으로 오버헤드 ■ 코드 노출
Two Adv.[15]	패킷단위 해시체인	별도의 해시체인	-	■ 패킷 수신 순서가 다른 경우 버퍼링 오버헤드 및 인증 지연 ■ 광고 패킷에 대한 DoS공격 취약 ■ 코드 노출
Castor [16]	페이지단위 해시체인	첫페이지의 해시값에 키MAC 전송 및 주기별 키노출에 의한 인증	-	■ 키노출 주기 설정 값에 따른 인증 지연 발생 가능 ■ 전체 페이지 수신후 인증가능하므로 플러딩 DoS공격 취약 ■ 코드 노출
Supplymentary Hash [17]	패킷단위 해시체인을 중복 적용	첫패킷에 대한 전자서명	-	■ 중복해시에 의한 전송량 증가 ■ 코드 노출
Multiple Key Chain [18]	패킷단위 해시체인	각 홉그룹별 키체인	-	■ 베이스 스테이션으로부터 각 홉그룹별 송신자 인증 및 메시지 무결성 체크를 위한 키 업데이트 세그먼트와 패킷 인증 세그먼트를 붙임으로 상당한 전송량 증가 ■ 코드 노출
Seluge [19]	패킷단위 해시체인 과머클해시트리	머클해시트리루트 패킷에 대한 전자서명	-	■ 각 노드에 이웃노드와의 pairwise 클러스터키 관리를 요구하므로 많은 키 관리 필수 ■ 코드 노출
Confidenciality [20]	-	-	랜덤키 선분배로 암호키 공유	■ 단순히 기존 랜덤키 선분배 적용으로 암호화키 사용했지만, 노드당 많은 키 관리 필수 ■ 인증과 무결성에 대해서는 자세한 언급이 없음

- : 제공하지 않음

존에 제안된 TinyOS기반 플러그인 프로그램인 FlexCup[9]을 이용하여 동적으로 로딩하는 구조를 제안하였다. 즉, 센서 네트워크 초기 운영 시에는 키 설립을 위해 공개키 암호 코드를 로딩하고, 이후 일반모드가 되면 통신의 안전성을 제공하기 위해 자원을 효율적으로 사용하는 대칭키 암호 코드를 로딩하여 사용하는 구조이다. 코드분배를 이용해 적절한 시기에 적절한 코드를 로딩하여 사용하는 한 응용을 제시하였다는데 의의가 있다.

■ *공격: 코드삽입공격 [22]*

많은 상용 센서 노드의 기반 구조인 Harvard-Architecture 기반 센서 노드에 대해, 코드의 취약점을 이용하는 악의적인 패킷을 생성하고 노드가 이 패킷을 수신하였을 때, 원하는 코드를 삽입할 수 있음을 보이고 있다. 악의적인 패킷은

코드내의 버퍼 오버플로우를 유도하고, 오버플로우에 의해 되돌아갈 리턴 주소를 의도된 주소를 넣어 공격자가 원하는 코드가 삽입되게 한다. 센서 패킷의 길이가 짧은 것을 감안하여 Meta-gadget이라는 여러 체인 가젯을 연결하여 메모리에 삽입하고, 이를 통해 원하는 코드를 삽입할 수 있음을 구현을 통해 보이고 있다. 이에 대한 간단한 대응책으로 소프트웨어의 취약점 보완, 코드삽입공격에 악용될 수 있는 Stack-smashing에 대한 보호, Garget 실행에 대한 보호 등을 제시하였다. 본 논문은 정상적인 패킷을 가장하여 원하는 코드로 리로딩의 가능성을 보임으로써 새로운 연구 주제를 제시하였다는데 의의가 있다.

4. 향후 연구 방향 및 제언

지금까지 분석한 기존 코드분배 메커니즘의 특징 및 장단점 비교를 통해 향후 다음과 같은 연구가 진행되어야 할 것이다.

첫째, 효율적인 측면에서 제공되는 코드 사이즈 최소화와 코드분배 프로토콜이 유기적으로 상호 협조하여 운용될 수 있는 메커니즘이 필요하다. 예를 들어 전송되는 코드의 사이즈에 따라 빠른 전송이 될 수 있도록 코드분배 송신지의 노드나 그 수가 결정되는 메커니즘 등이 제공되어야 한다.

둘째, 효율성과 안전성을 동시에 고려한 코드분배 프로토콜 연구가 필요하다. 지금까지 안전성을 제공하는 코드분배 메커니즘들은 코드 최소화 방안을 고려하지 않은 Deluge를 기반으로 하고 있어서 효율성과 안전성을 동시한 고려한 연구가 진행되어야 한다.

셋째, 많은 센서 노드 수를 지원 가능한 코드분배 프로토콜로서, 지금까지는 이를 고려하지 않고 한 베이스 스테이션에서 플러딩을 하듯 말단 센서 노드들까지 코드분배가 수행되는 메커니즘으로 설계되었다. 그러나 이는 많은 노드로 구성된 센서 네트워크에서는 비효율적으므로 많은 노드 수에도 빠르고 에너지 효율적으로 코드분배를 수행하는 프로토콜이 필요하다.

넷째, 지금까지 제어 패킷이나 전자서명 패킷에 대한 DoS공격이나 시빌공격, 웜홀공격에 대응할 수 있는 코드분배 프로토콜이 제공되고 있지만 대응 방안의 효율성을 높여야 한다. 또한 코드분배 시 선택적인 포워딩 공격을 유도하여 한페이지의 전송 시간을 지연시키거나, 한 노드가 코드의 송신 노드 선택 시 공격자가 강한 세기로 헬로우 패킷을 전송함으로써 공격자로부터 코드 수신을 유도하여 새로운 코드분배를 방해하는 공격이 가능하므로 이러한 공격에 대응 가능한 안전한 코드분배 프로토콜 고안이 필수적이다.

마지막으로, 지금까지 제안된 안전한 코드분배 메커니즘의 대부분은 오버헤드의 이유로 기밀성을 제공하고 있지 않지만, 코드 노출은 [22]에서 제시한 것처럼 새로운 공격의 기반으로 사용될 수 있으므로 경량화된 기밀성 제공 메커니즘이 필요하다.

5. 결 론

센서 네트워크의 주요 연구 주제 중의 하나로 최근 활발히 진행되고 있는 코드분배는 많은 노드의 소프트웨어 업그레이드나 버그 수정을 위해 없어서는 안될 중요한 메커니즘이다. 그러나, 분석된 바와 같이 기존 연구들은 안전성 및 효율성 측면이 구분되어 진행되어 왔고 아직 개선해야 할 사항이 남아있다. 본 논문에서는 최근 제안된 기존 센서 네트워크 코드분배 메커니즘을 비교 분석하고 이를 통해 향후 연구 방향에 대해 제안하였다. 이를 통해 센서 네트워크에서의 코드분배 메커니즘 연구 동향을 이해하고 해당 연구를 도모하며, 그 결과를 통해 센서 네트워크 활용도의 제고를 가능하게 할 것이다.

참 고 문 헌

[1] P. Brutch and C. Ko, “Challenges in intrusion detection for wireless ad-hoc networks,” Symposium on Applications and the Internet Workshops (SAINT), pp.368-373, 2003.

[2] 최경진, 윤명준, 심인보, 이재용, “무선 센서 네트워크에서의 에너지 효율적인 클러스터 헤드 선출 알고리즘”, 한국통신학회논문지, Vol.32, No.6, pp.342-349, 2007.

[3] Youtao Zhang, Jun Yang, and Weijia Li, “Towards Energy-Efficient Code Dissemination in Wireless Sensor Networks,” IEEE International Symposium on Parallel and Distributed Processing, pp.1-5, 2008.

[4] R. Panta, I. Khalil, and S. Bagchi, “Stream: Low Overhead Wireless Reprogramming,” IEEE Conference on Computer Communications (INFOCOM), pp.928-936, 2007.

[5] Weijia Li, Youtao Zhang, Jun Yang, and Jiang Zheng, “UCC: Update-conscious Compilation for Energy Efficiency in Wireless Sensor Networks,” ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), San Diego, California, Vol.42, No.6, pp.383-393, June, 2007.

[6] M. D. Krasniewski, S. Bagchi, C-L. Yang, and W. J. Chappell, “Energy-efficient, On-demand Reprogramming of Large-scale Sensor Networks,” Transactions on Sensor Network (TOSN), Vol.4, No.2, 2008.

[7] W Tang, W Kuang, B Wang, and J Yang, “E3NP: An Energy-Efficient, Expeditious Network Reprogramming Mechanism in Wireless Sensor Network,” Embedded Software and Systems (ICESS), pp.516-523, 2008.

[8] Michele Rossi, Giovanni Zanca, Luca Stabellini, Riccardo Crepaldi, Albert F. Harris III and Michele Zorzi, “SYNAPSE: A Network Reprogramming Protocol for wireless sensor networks using fountain codes,” IEEE SECON, pp.188-196, 2008.

[9] P. J. Marron, M. Gauger, A. Lachenmann, D. Minder,

O.Saukh, and K. Rothermel, “Flexcup: A flexible and efficient code update mechanism for sensor networks,” European Workshop on Wireless Sensor Networks (EWSN), pp.212-227, 2006.

[10] Adam Dunkels, Niclas Finne, Joakim Eriksson, and Thiemo Voigt, “Run-Time Dynamic Linking for Reprogramming Wireless Sensor Networks,” Embedded Networked Sensor Systems (SenSys), pp.15-28, 2006.

[11] P. E. Lanigan, R. Gandhi, and P. Narasimhan, “Sluice: Secure dissemination of code updates in sensor networks,” Distributed Computing Systems (ICDCS), pp.53-63, July, 2006.

[12] P. K. Dutta, J.W. Hui, D. C. Chu, and D. E. Culler, “Securing the Deluge network programming system,” Information Processing in Sensor Networks (IPSN), pp.326-333, 2006.

[13] DENG, J., HAN, R., and MISHRA, S, “Secure code distribution in dynamically programmable wireless sensor networks,” Information Processing in Sensor Networks (IPSN), pp.292-300, 2006.

[14] Jing Deng, Richard Han, and Shivakant Mishra, “Efficiently Authenticating Code Images in Dynamically ReprogrammedWireless Sensor Networks,” Pervasive Computing and Communications Workshops (PERCOMW), pp.272-277, 2006.

[15] Sokjoon Lee, Howon Kim, and Kyoil Chung, “Hash-based Secure Sensor Network Programming Method without Public Key Cryptography,” World-Sensor-Web at International Conference on Embedded Networked Sensor Systems, 2006.

[16] D. H. Kim, R. Gandhi, and P. Narasimhan, “Exploring Symmetric Cryptography for Secure Network Reprogramming,” Wireless Ad hoc and Sensor Networks, Canada, pp.17-25, 2007.

[17] Kwangkyu Park, JongHyup Lee, Taekyoung Kwon and Jooseok Song, “Secure Dynamic Network Reprogramming Using Supplementary Hash in Wireless Sensor Networks,” Ubiquitous Intelligence and Computing, LNCS 4611, pp.653-662, 2007.

[18] H Tan, S Jha, D Ostry, J Zic, and V Sivaraman, “Secure multi-hop network programming with multiple one-way key chains,” ACM conference on Wireless network, pp.183-193, 2008.

[19] S Hyun, P Ning, A Liu, and W Du, “Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks,” Information Processing in Sensor Networks (IPSN), pp.445-456, 2008.

[20] 신승목, 최임성, 김광조, “무선 센서 네트워크에서의 안전한 네트워크 재프로그래밍 기법”, 한국정보보호학회 하계 학술대회(CISC), 2008.

[21] D. W. A. Poschmann and A. Weimerskirch, “Dynamic Code Update for the Efficient Usage of Security Components in WSNs,” KiVS Industribeitrage, Kurzbeitrage und Workshops. Berlin: VDE Verlag, pp.445-455, 2007.

[22] Aurelien Francillon, and Claude Castelluccia, “Code Injection Attacks on Harvard-Architecture Devices,” ACM conference on Computer and communications security, pp.15-26, 2008.

[23] S.Brwon, “Updating Software in Wireless Sensor Networks: A Survey,” Dept. of Computer Science, National Univ. of Ireland, Maynooth, Tech. Rep., 2006.

[24] J. W. Hui and D. and Culler, “The dynamic behavior of a data dissemination protocol for network programming at scale,” International conference on Embedded networked sensor systems, pp.81-94, 2004.

[25] Jeong J. and Culler.D, “Incremental network programming for wireless sonsors,” Sensor and Ad Hoc Communications and Networks, pp.25-33, 2004.

김 미 희

e-mail : iceblueee@gmail.com

1997년 이화여자대학교 전자계산학과(학사)

1999년 이화여자대학교 컴퓨터학과(석사)

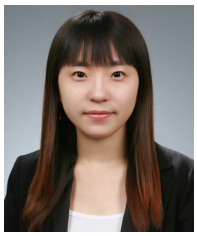
1999년～2003년 한국전자통신연구원 연구원

2007년 이화여자대학교 컴퓨터공학과(박사)

2007년～2009년 이화여자대학교 컴퓨터공학과 전임강사

2009년～현 재 미국 North Carolina State University Postdoc Researcher

관심분야 : 무선 네트워크(센서네트워크, 유비쿼터스네트워크, 메쉬네트워크) 보안, 물리계층 보안



김 지 선

e-mail : 272lovelyjs@ewhain.net

2007년 서울여자대학교 정보보호공학과(학사)

2007년～현 재 이화여자대학교 컴퓨터공학과 석사과정

관심분야 : 센서 네트워크 보안, 침입 탐지 및 대응 기술, 센서 네트워크 리프로그래밍 기술

김 지 현

e-mail : jhkim85@ewhain.net

2008년 서울여자대학교 정보보호공학과(학사)

2008년～현 재 이화여자대학교 컴퓨터공학과 석사과정

관심분야 : 센서 네트워크 보안, 센서 네트워크 리프로그래밍 기술, 메디칼 센서 네트워크 기술

임 지 영

e-mail : jylim@bible.ac.kr

1994년 이화여자대학교 전자계산학과(학사)

1996년 이화여자대학교 컴퓨터학과(석사)

2001년 8월 이화여자대학교 컴퓨터공학과 (박사)

2001년 9월~2003년 2월 이화여자대학교 컴퓨터공학과 전임강사

2003년 3월~현 재 한국성서대학교 정보과학부 조교수

관심분야 : 센서 네트워크 보안, 침입 탐지 및 대응, 네트워크 프로토콜 설계 및 성능분석

채 기 준

e-mail : kjchae@ewha.ac.kr

1982년 연세대학교 수학과(학사)

1984년 미국 Syracuse University 컴퓨터학과(석사)

1990년 미국 North Carolina State University 컴퓨터공학과(박사)

1990년~1992년 미국 해군사관학교 컴퓨터학과 조교수

1992년~현 재 이화여자대학교 컴퓨터공학과 교수

관심분야 : 네트워크 보안, 인터넷/무선통신망/고속통신망/센서네트워크 (보안)프로토콜 설계 및 성능분석