DCUP: 무선 센서 노드를 위한 새로운 동적 코드 갱신 프로토콜

김종화 [O1], 이상호 [2], 최종무 [1]
단국대학교 정보 컴퓨터 과학과[1], 서울대학교 컴퓨터 공학과[2]
{zcbm4321, choijm}@dankook.ac.kr[1], shyi@ssrnet.snu.ac.kr[2]

DCUP: A New Dynamic Code Update Protocol for Wireless Sensor Nodes

Jonghwa Kim[1], Sangho Yi[2], Jongmoo Choi[1]
Division of Information and Computer Science Dankook University[1]
School of Computer Science and Engineering, Seoul National University[2]

센서 네트워크에서 운영체제는 모듈을 동적으로 교체할 수 있는 동적 재구성 기능이 필요하다. 동적 재구성이 필요한 이유는 다음과 같다. 첫 번째, 센서 네트워크를 구성할 때 센서 장치는 수십 개에서 수천 개씩 설치되므로 버그가 발생했을 때 수정이 불가능하다면 모든 센서 장치들이 정상적인 동작을 기대하기 힘들다. 두 번째, 센서 네트워크를 운용함에 있어서 다른 서비스를 제공하거나 구조의 효율적 변경이 필요하다. 그래서 최근에 등장한 센서 운영체제들은 동적 재구성 기능을 제공하고 있다.

본 논문이 제안하는 방법은 세가지 이점을 갖는다. ***최소한의 링크정보*** 데이터의 크기는 센서의 에너지와 관계가 많다. 데이터를 함수 호출 관계 그래프를 이용하여 경로 값을 생성하여 크기를 줄였다. ***운영체제 내 Symbol Table 미사용*** 함수 호출 관계 그래프는 메모리 안의 명령어를 분석하여 검색하므로 심볼 테이블을 필요로 하지 않는다. ***함수 단위 업데이트*** 본 논문은 함수 관계 가중치 그래프를 이용하여 함수 단위 업데이트를 지원한다.

센서 네트워크에서 여러 운영체제가 동적 재구성을 지원하고 있으며 각 운영체제가 지원하는 방식은 장단점이 존재한다. 장단점은 두 가지 축으로 구분할 수 있는데 재구성의 유연함과 전송되는 파일의 크기이다. TinyOS는 유연함과 크기가 가장 높고 Contiki와 SOS는 중간 정도의 위치를 가지고 있다. 스크립트나 환경변수와 같은 방법은 유연함과 전송 크기가 가장 작다.

본 논문에서 제안하는 업데이트 방법은 Contiki와 SOS의 위치에서 유연함을 높이고 크기는 더 작은 위치에 있으며 또, 일반 동적 재구성과 다르게 심볼테이블을 사용하지 않고 독자적인 링크 방식을 사용하여 최소한의 크기로 전송한다.

앞서 언급한 3가지 목표를 이루기 위해서 함수 호출관계를 이용한 그래프를 구성하였다. ***그래프 구성하기***: 함수와 함수의 호출 관계를 연결하면 그래프를 구성할 수가 있다. **그림 1**을 보면 a함수에서 b함수와 c함수를 호출할 때 a함수는 b함수, c함수와 연결을 그릴 수 있으며 또 b함수는 d함수와 e함수를 호출할 때 연결을 그릴 수 있다. 이런 방법을 반복하여 연결된 모든 함수를 그래프로 구성한다. 함수의 호출 명령어를 검색할 때는 프로그램 코드를 일일이 비교하여 call 명령어를 찾는다. 원하는 call 명령어를 찾기 위한 비교 횟수가 존재하는데 그 값은 해당 call 명령어의 가중치이다. 이러한 가중치 값을 기반으로 **그림 2**와 같은 그래프를 구성할 수 있다. ***링크정보 만들기***: 구성된 그래프와 경로가 구해지면 이 데이터를 바탕으로 링크 정보를 구성한다. 예를 들어 d함수를 업데이트한다면 **그림1**과 **그림2**를 바탕으로 a→b→d 의 경로가 최적임을 알 수 있다. d함수의 위치를 찾기 위해서는 시작함수에서 1번째 호출 명령어를 찾고 주소를 추출하여 그 지역을 검색하여 1번째 호출 명령어를 찾아 주소를 추출하게 되면 d함수

의 위치를 찾을 수 있게 된다. 따라서 전송되는 데이터는 1, 1 이 된다.

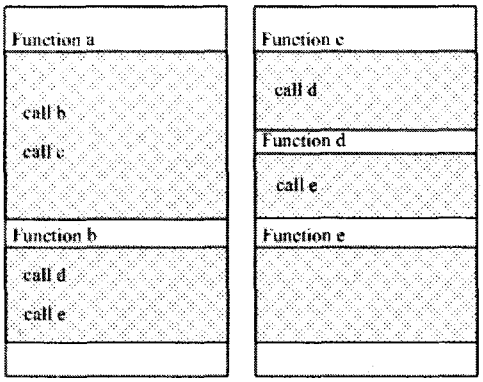

그림 1 메모리 안의 함수 호출 관계

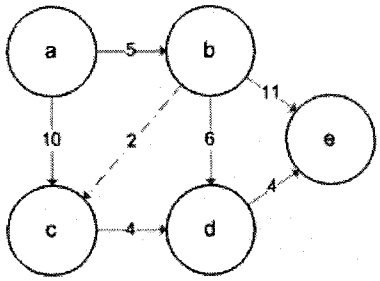

그림 2 함수 호출 관계 가중치 그래프

본 논문에서 제안하는 함수 관계를 이용한 가중치 그래프를 구성하는 프로그램을 만들어서 실험하였다. 실험 환경은 MSP430 MCU[6]를 사용하는 Tmote Sky[7]를 대상으로 하였다. Contiki의 전체 이미지를 이용하여 그래프를 구성하였으며 그 정보를 바탕으로 DCUP 전송 포맷을 만들었다.

표 1을 보면 Contiki 전송 모듈의 ELF 포맷(Contiki-2.x/platform/sky/loadable_prg.ko)과 DCUP 포맷의 크기를 비교 하였다. 해당 모듈은 재배치함수 7개로 구성되었다. 아직 커널과 링크되지 않은 함수이므로 재배치에 필요한 부가적인 정보가 기록되어있다.

포맷	Text + bss + data	Link info
ELF	248 Byte	758 Byte
DCUP	248 Byte	30 Byte

표 1. ELF와 DCUP의 크기 비교

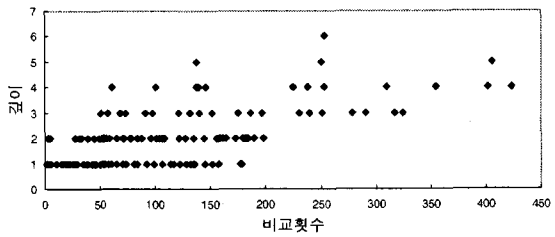

그림 5. Contiki 커널 함수의 가중치 그래프 구성 결과 깊이(depth)와 명령어 비교 횟수

그림 5는 Contiki 함수 가중치 그래프의 결과를 나타내었다. y축은 함수의 깊이이고 x축은 해당 함수를 찾기 위한 명령어 비교 횟수이다. 함수들의 평균 비교 횟수는 약 121번이었으며 깊이는 1~2가 대부분이었다. DCUP는 메모리 안에 명령어를 비교하는 방식으로 RISC 프로세서가 대부분인 센서 장치는 RISC방식의 특징으로 인해 코드가 길다. 따라서 심볼 테이블에서 해당하는 함수를 비교하는 비용은 심볼들을 정렬하여 이진검색을 사용하여 log(*심볼의 수*)의 비용이 들지만 경로 방식은 코드의 길이N의 영향을 받으므로 심볼 테이블 방식보다 평균적으로 비교횟수가 많다. 하지만 심볼테이블 비교방식은 단순 명령어 비교방식보다 좀 더 복잡한 코드를 가지고 있고, **그림 5**의 데이터를 보면 함수의 깊이와 비교 횟수가 비교적 좋은 구간에 집중해있어 실제 실험 결과 평균적으로 걸리는 시간은 심볼 테이블 검색방식과 별다른 차이가 없었다.