2002년도 대한전기학회 하계학술대회 논문집 2002.7.10-12

론웍스 시스템에서 이벤트 유실을 막기 위한 기법 제안

윤석현, 김영호, 심일주, 박귀태
고려대학교 전기공학과

A Study on the Design Method to Prevent the Event-Loss in LonWorks

Seok-Hyun Yoon, Young-Ho Kim, Il-Joo Shim, Gwi-Tae Park
Department of Electrical Engineering, Korea University

Abstract - 본 논문에서는 론웍스(LonWorks)를 사용할 때 이벤트 또는 메시지가 처리되는 과정을 분석하여 각 과정에 따라 발생할 수 있는 중요 정보의 유실 문제, 그리고 그 해결 방안을 제시하고자 한다. 노드를 개발할 때 우선순위를 갖는 태스크(Task)의 개수를 줄여 비우선순위의 태스크가 원활히 실행되도록 하는 방안을 제시하며, 네트워크를 구성할 때 반복적인 비응답(unacknowledged) 전송 서비스를 사용하는 방법을 제시하고 시뮬레이션을 통해 그 신뢰성을 검증한다. 또한 데이터 기반의 구성 속성(Configuration Property) 값을 설정하여 변화된 입력 값들이 손실없이 반영되도록 하는 방법을 제시한다.

1. 서 론

제어 기술과 이론이 빠른 속도로 발전함에 따라 최근의 제어 시스템은 보다 향상된 제어 성능을 발휘할 수 있도록 개선되어 왔다. 특히 분산되고 네트워크화 된 제어 시스템은 기존의 중앙집중적인 제어 시스템에 비해 많은 장점을 갖고 있다[4]. 이러한 분산형 플랫폼을 가지면서 서로 다른 제조업체에서 만든 말단 기기(노드, node)들이 론마크 협회가 정의한 표준 형식에 맞게 개발되는 것 만으로도 쉽게 상호 운용가능성을 확보하고 다양한 통신 매체를 지원하여 완벽한 분산 구조를 이룰 수 있게 하는 기술이 론웍스(LonWorks) 이다.

론웍스 시스템의 각 노드는 뉴런 칩과 트랜시버를 포함하고 있다. 뉴런 칩은 서로 다른 역할을 하는 3개의 프로세서로 구성되어 있어 마이크로콘트롤러의 핵심 역할과 함께 네트워킹 능력도 갖추고 있다. 트랜시버는 뉴런 칩을 쌍꼬임선, 전력선, RF등 다양한 통신매체와 연결시키는 역할을 한다. 노드 개발자는 3개의 프로세서 중 어플리케이션 처리과정을 담당하는 CPU에 뉴런 C라는 특정 프로그램 언어로 응용 프로그램을 코딩하면 되고, 나머지 두 개의 CPU는 다른 노드들과 통신할 수 있도록 론토크라는 프로토콜을 내재하고 있다.

뉴런 C는 'when()'으로 시작하는 개별 태스크(task)들이 특정 이벤트가 발생했을 때 만 수행되는 방식의 이벤트 반응 기법을 사용한다. 각각의 태스크에는 우선순위를 두어 우선권이 있는 태스크가 전부 실행된 후에 우선권이 없는 태스크를 실행하는 순서를 따른다.

론토크(LonTalk) 프로토콜은 론웍스 노드들 간의 통신을 위해 제안된 프로토콜로 ISO의 OSI 7계층 모델을 모두 따르며, EIA 709.1 표준으로 등재되어 있다. 론토크에서는 네트워크 상의 다른 노드들과 데이터를 효과적으로 교환하기 위해 네트워크 변수를 사용하는 암시적 메시지(Implicit message)와 명시된 메시지(Explicit message)라는 두 개의 통신 객체 클래스를 사용하는데, 이들은 여러 제어 값들의 변화를 네트워크 상의 다른 노드들에 알리는 역할을 담당한다.

본 논문에서는 론웍스 노드를 개발하고 이를 다른 노드들과 네트워크화 하는 과정에서 발생할 수 있는 이벤트나 메시지의 유실 가능성을 분석하고 이를 막기 위한 방법을 제시한다. 2.1절에서는 뉴런 C를 사용해서 노드를 프로그램하는 방법을 분석하고, 이때 발생할 수 있는 이벤트 유실 문제와 그 해결책을 제시한다. 2.2절에서는 각 노드들을 네트워크화 할 때, 메시지를 전송하는 단계와 중요 데이터 값의 변화를 감지하는 단계에서 발생할 수 있는 메시지 유실 문제를 분석하고 그 해결책과 타당성을 제시한다. 3절에서는 본 논문의 필요성을 설명한 후 결론을 내린다.

2. 론웍스 노드의 개발과 통신

2.1 론웍스 노드의 구성

뉴런 칩은 담당하는 역할에 따라 분리된 3개의 8비트 CPU로 구성되어 있다. MAC(Media Access Control) CPU, 네트워크 CPU, 어플리케이션 CPU가 그것인데 이들은 ALU, RAM, ROM, EEPROM 등의 자원을 공유하며 노드의 동작을 관할한다. 이중 MAC CPU와 네트워크 CPU는 론토크 프로토콜을 내장하고 노드간의 통신을 담당하며, 프로그래머는 하부 하드웨어 구조를 크게 신경쓰지 않고 어플리케이션 CPU에 원하는 프로그램을 하기만 하면 된다. 이 어플리케이션 CPU에 사용되는 프로그래밍 언어는 ANSI C를 기반으로 뉴런 칩에서만 사용할 수 있도록 개발된 뉴런 C라는 특별한 프로그래밍 언어이다.

2.1.1 프로그래밍 모델

뉴런 칩의 스케쥴러는 어플리케이션 프로그램이 멀티 태스킹할 수 있도록 한다. 태스크는 하나의 이벤트를 다루는 작은 프로그램 단위로 볼 수 있는데 'when()'으로 시작하는 구문을 통해 아래와 같이 정의된다.

〔우선순위〕 when(이벤트)

'우선순위'는 해당 태스크의 우선권을 부여하여 우선순위 태스크는 우선순위가 없는 태스크를 실행하기 전에 항상 먼저 수행되어야 한다. '이벤트'는 해당 태스크가 실행되기 위해 이뤄져야 할 조건을 담고 있으며 이벤트가 참(true)일 경우에만 when절 이하의 구문이 실행되는데, 네트워크 변수의 변화, 입력 핀 값의 변화, 타이머 소멸, 메시지 수신 등을 이벤트로 사용할 수 있다.

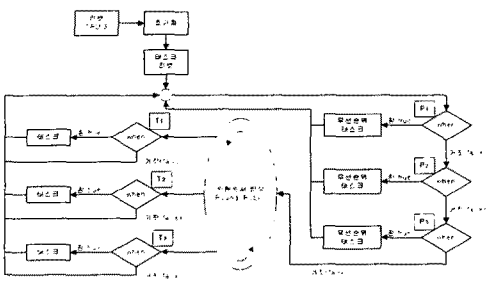

그림 1 스케쥴러의 동작 순서

그림 1은 스케쥴러의 동작 순서를 나타내고 있는데, 스케쥴러는 리셋 태스크에서 시작하며 이때 어플리케이션의 모든 초기화가 이루어진다. 시스템에 따른 초기화가 끝나면 우선순위 태스크가 검사되고 이벤트가 참(true)인 태스크가 있다면 실행된 후 스케쥴러 루프의 시작점으로 돌아간다. 만일 어떤 우선순위 태스크도 참이 아니라면 비우선순위 태스크를 검사한다. 이들 태스크들은 순환 순서(round-robin) 방식으로 선택되며, 각 비우선순위 태스크들이 검사되기 전에는 항상 모든 우선순위 태스크들이 계속 검사된다. 마지막 비우선순위 태스크가 검사된 후, 스케쥴러는 다시 시작점으로 돌가서 첫 번째 우선순위 태스크를 검사한다.

when절의 참/거짓을 판단하기 위해서는 내부클럭이 10MHz일 때 대략 1m sec 정도의 시간이 걸리는데 그 정확한 값은 프로그램 문장의 복잡도에 좌우된다. 만약 1m sec라고 가정하고 어떤 우선/비우선순위 태스크도 참이 아니라고 가정한다면, 스케쥴러가 모든 when절을 검사하는데 걸리는 시간은 다음과 같다[1].

$$PN+N=(P+1)N\ [m\,\mathrm{sec}]$$

P : 우선순위 태스크의 개수
N : 비우선순위 태스크의 개수

2.1.2 이벤트 유실을 막기 위한 스케쥴링 기법

위 그림 1에서 시작점 이후에 검사되는 태스크의 순서는 모든 태스크가 거짓(false)일 경우 다음과 같다.

$$P_1\rightarrow P_2\rightarrow P_3\rightarrow T_1\rightarrow P_1\rightarrow P_2\rightarrow P_3\rightarrow T_2\rightarrow P_1\rightarrow P_2\rightarrow P_3\rightarrow T_3$$

그러나 어떤 우선순위 태스크의 이벤트가 참이라면 해당 태스크가 실행된 후 다시 시작점으로 돌아가므로, 비우선순위 태스크들은 검사받기 위해 상당한 시간을 기다릴 수 밖에 없다. 이때 발생한 이벤트들은 검사받지 못해 유실될 가능성이 있다. 다음이 그런 예이다.

$$P_1\rightarrow P_2\rightarrow P_3\rightarrow P_1\rightarrow P_2\rightarrow P_3\rightarrow T_1\rightarrow P_1\rightarrow P_2\rightarrow \cdots\cdots$$

여기서 P_3가 참일 경우 해당 P_3 태스크를 수행한후 스케쥴러는 다시 시작점으로 돌아가 P_1부터 다시 검사를 하게 되므로 T_1이 검사받기까지 많은 시간이 소요됨을 알 수 있다. 또한 우선순위 태스크가 하나 증가하면 모든 비우선순위 태스크를 검사하기 전에 증가된 우선순위 태스크가 검사되어야 하기 때문에 우선순위 태스크의 개수는 비우선순위 태스크의 개수보다 전체 시스템의 속도에 미치는 영향이 큼을 알 수 있다. 이 경우 모든 비우선순위 태스크가 검사받을 때까지의 시간이 증가되므로, 다른 이벤트가 발생할 가능성이 더 커지며 이때 발생한 이벤트는 유실될 것이다.

따라서 원치않는 이벤트 유실을 막기 위해서는 우선순위 태스크의 개수를 되도록 적게 설계하여 꼭 필요한 경우에만 우선순위 태스크를 사용하는 것이 바람직하다. 스케쥴러의 바이패스(Bypass) 모드를 사용하여 단 하나의 태스크만 갖도록 알고리즘을 구현할 수도 있는데 이 경우 모든 이벤트들은 if 구문을 사용하여 단일 when절 안에 들어가게 프로그램 하면 된다(그림 2).

```
when(TRUE)
{
   while(TRUE)
   {
      if (offline) {.....}
      if (.....) {.....}
      .....
      if (msg_arrives) {.....}
   }
}
```

그림 2 바이패스 모드를 사용한 단일 when절의 예

그러나 하나의 when절 수행 시간이 너무 길게 프로그램되면 해당 when절의 태스크가 실행중인 동안 발생한 다른 이벤트들은 무시될 수 있으므로 적절한 길이로 나눠줘야 한다. 실제로 뉴런 칩은 10MHz의 내부클럭에서 840msec의 왓치독 타이머(Watchdog Timer)를 갖고 있어서 when절의 길이를 제한하는데, 부득이 이보다 긴 실행시간이 필요하다면 이 타이머를 적절히 리셋해줘야 한다.

2.2 론웍스 노드들간의 통신

론웍스 노드들간의 통신은 두가지 방식으로 이뤄진다. 첫 번째 방법은 네트워크 변수를 이용하여 데이터 전송을 하는 방법인데 네트워크 변수 자체가 데이터가 어떤 종류인지 미리 정해 놓고 프로그래머에게 알려져 있으므로 암시적 메시지(Implicit Message)라 한다. 두 번째 방법은 명시된 메시지(Explicit Message)를 이용한 방법인데 데이터 패킷의 송수신을 위해서는 원하는 데이터의 송수신을 요청하는 함수를 호출해야 한다. 후자가 전자보다 적은 EEPROM 메모리를 사용하지만 보다 많은 프로그램 메모리를 필요로 한다.

또한 네트워크 변수를 사용하는 방법은 여러 개발자들이 론마크(LonMark) 협회가 정의한 네트워크 변수의 표준형식에만 맞추어 노드를 개발하면 쉽게 다른 노드들과의 상호 운용 가능성을 확보할 수 있게 하는 큰 장점이 있다.

2.2.1 네트워크 변수 연결과 메시지 유실

네트워크 변수는 입력 네트워크 변수와 출력 네트워크 변수로 나눌 수 있는데, 어플리케이션 프로그램이 새로운 값을 출력 네트워크 변수에 할당하면 이와 바인딩(binding)된 입력 네트워크 변수를 갖는 모든 다른 노드들에 자동적으로 이 값이 전송된다. 이때 전송은 응답(ACKD, acknowledged service)과 비응답(UNACKD, unacknowledged service)의 두가지 방식으로 이뤄지는데, 응답 서비스는 수신 노드로부터 데이터 수신 여부에 대한 확인(ACK)을 요구하는 방식으로 일정 시간 내에 ACK 응답이 오지 않으면 자동적으로 재송신한다. 이러한 방식은 수신 노드가 응답을 하지 않을시 송신측에서 적절한 행동을 취할 수 있는 장점이 있지만 수신 노드의 개수가 늘어남에 따라 전체 네트워크에 실리는 메시지 부하(load)가 많아지게 되는 단점이 있다. 따라서 많은 노드를 갖는 네트워크에서는 수신 노드의 응답을 요구하지 않는 비응답 서비스 방식이 사용되는 것이 일반적이다.

하지만 이러한 비응답 서비스 방식은 메시지의 수신 여부를 확인하지 않기 때문에 메시지 유실이 발생시 해당 네트워크 변수의 값이 갱신되지 못하는 문제가 생길 수 있다. 이를 방지하기 위해 일정 횟수 만큼 반복적으로 비응답 서비스(UNACKD_RPT, unacknowledged /repeated service)를 사용하여 데이터를 보냄으로써 신뢰도를 증가 시킬 수 있다. 론토크 프로토콜은 비응답 서비스 전송의 재시도 횟수를 0~15회로 제한하고 있는데, 재시도 횟수가 메시지 전송의 신뢰도에 미치는 영향은 아래와 같다[3].

$$P(\le k)=\sum_{i=0}^{k}{}_{k+1}C_i\,p^i\,(1-p)^{k-i+1}\,(1-p^{k-i+1})^{n-1}$$

p : 단일 메시지 전송 실패 확률
n : 전체 수신 노드의 개수
k : 재시도 횟수

여기서 $P(\le k)$는 k번의 재시도 안에 전송이 성공할 확률을 나타낸다. 그림 3과 표 1은 p=0.1일 때 이를 나타낸 것이다.

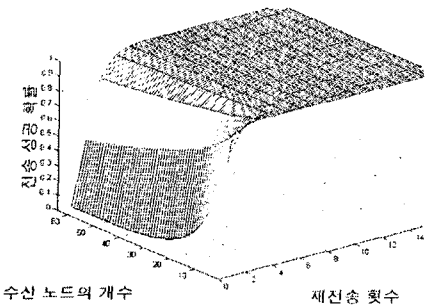

그림 3 재시도 횟수에 따른 전송 성공 확률

표 1 재시도 횟수에 따른 전송 성공 확률

수신 노드의 개수	재전송 횟수		
	3회	4회	5회
2	0.9987	0.9998	0.9999
4	0.9964	0.9993	0.9999
8	0.9922	0.9984	0.9997
16	0.9849	0.9968	0.9994
32	0.9725	0.9940	0.9987
64	0.9516	0.9889	0.9976

여기서 3번의 재시도 만으로도 95% 이상 신뢰할 만한 전송 성공 확률을 갖는 다는 것을 확인할 수 있다. 따라서 많은 수신 노드를 갖는 네트워크에서의 데이터 전송은 3~5번의 반복횟수를 갖는 비응답 서비스를 사용함으로써 메시지 유실을 방지하기 위한 신뢰성을 확보할 수 있다. 이런 메시지 전송 타입은 명시된(Explicit) 메시지를 사용할 경우에도 마찬가지로 사용된다.

2.2.2 구성 속성 설정과 중요 업데이트 값 유실

론마크 협회는 구성 속성(configuration property)의 표준 타입 역시 정의해 놓고 있는데, 구성 속성은 EEPROM에 저장되어 디바이스의 행동 양식을 정의하고 있다. 이 값은 네트워크 변수와는 달리 값이 자주 변하지 않는 정적(static) 데이터인데, 노드를 새로 코딩할 필요없이 파라미터화 된 값들을 변화시킴으로써 디바이스의 동작과 네트워크의 성능을 최적화 시킬 수 있다. 자주 쓰이는 표준 구성 속성 변수는 Heartbeat /Max Receive Time, Throttle, Min Delta 등이 있다.

Heartbeat Time은 네트워크 변수 값이 변경되지 않아도 자동으로 데이터를 출력 네트워크 변수로 전송하는 최대 시간을 지정하며, Max Receive Time은 연결된 입력 네트워크 변수가 업데이트 되는 최대 시간을 지정한다. 이 두 매커니즘은 수신측에서 송신 노드가 여전히 동작중 임을 확실히 하기 위해 사용되는 것으로 최적의 성능을 위해서는 Max Receive Time=4×Heartbeat Time이 되도록 설정한다[2].

Throttle은 출력 네트워크 변수 전송의 최소 단위 시간을 지정하는 시간 기반 필터이다. Throttle 값이 너무 작게 설정되어 있으면 불필요한 업데이트에 의한 네트워크 대역폭 소비가 증가하게 되고 반대로 너무 길게 설정되어 있으면 출력 네트워크 변수가 업데이트 되지 않아 중요한 업데이트 정보를 놓칠 수도 있다(그림 4).

이러한 업데이트 정보 유실을 줄이기 위해서는 데이터 기반의 필터를 사용해야 한다. Min Delta는 델타 값 이상(%단위)의 값이 측정되는 경우 출력 변수의 값을 전송하도록 하여 비교적 중요하지 않은 데이터를 걸러내고 네트워크 트래픽을 줄이면서 중요한 값의 변화를 놓치지 않을 수 있다.

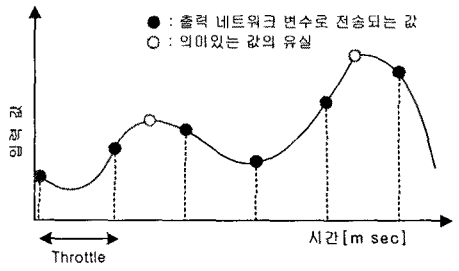

그림 4 Throttle과 중요 업데이트 값 유실

그림 5는 Throttle과 Min Delta 값에 의해 데이터 값이 전송되는 경우를 보여주고 있다. Throttle은 Min Delta보다 우선권을 갖기 때문에 Min Delta에서 지정한 값 이상이더라도 Throttle 시간이 종료되지 않았다면 데이터는 전송되지 않는다. 이렇게 데이터 기반의 구성 속성 변수를 적절히 조합하여 사용함으로써 네트워크 부하를 크게 증가시키지 않으면서도 중요한 데이터의 업데이트를 놓치지 않을 수 있다.

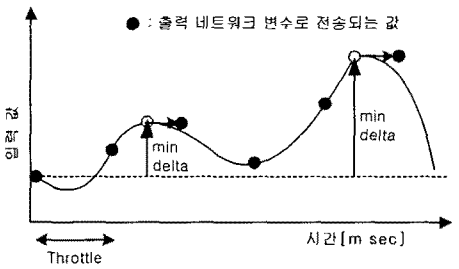

그림 5 Throttle과 Min Delta 구성 속성 변수

3. 결 론

센서 등으로 구성되는 제어 시스템의 특성상 화재 등 비상시 발생할 수 있는 특정 이벤트의 유실은 시스템 전체를 위태롭게 할 수 있는데, 론웍스 기술 역시 주의깊게 설계, 구성하지 않으면 이런 문제를 피할 수 없다.

본 논문에서는 론웍스 기술을 사용한 제어 시스템에서 발생할 수 있는 각종 중요 이벤트 또는 데이터 값 변화의 유실 가능성을 분석하고 그것을 방지하기 위한 방법을 제시한 후 그 타당성을 살펴보았다.

개발자는 노드 프로그램 단계에서 우선순위 태스크의 개수를 줄이고 반복적인 비응답 서비스를 사용하여 네트워크 변수를 연결하며, 최적화 단계에서 구성 속성 값을 적절히 설정하는 노력을 통해 노드와 네트워크가 중요 이벤트/데이터 값의 변화를 놓치지 않도록 할 수 있다.

[참 고 문 헌]

[1] Dietmar Loy, Dietmar Dietrich and Hans-Joerg Schweinzer, "Open Control Network-LonWorks /EIA 709 Technology", Kluwer Academic Publishers, 2001

[2] "LonWorks Network Design", version 1.1, Echelon Corporation, 2000

[3] "ANSI/EIA 709.1-A Standard - Control Network Protocol Specification", EIA, 1999

[4] Zhigang Liu, Huicun Wang, Wengqin Wu and Qingquan Qian, "Problem and Solutions of Electrified Railway Remote Control System Based on LON Technology", Autonomous Decentralized Systems, 2000. Proceedings. 2000 International Workshop, p68-71, 2000