Fault-Tolerant CAN 프로토콜

이진선*, 최경희*, 정기현**
*아주대학교 정보통신전문대학원
**아주대학교 전자공학부
e-mail : jstaira@ajou.ac.kr

A Fault-Tolerant CAN Protocol

Jin-Sun Lee*, Kyunghee Choi*, Kihyun Chung**
*Graduate School of Information and Communication, Ajou University
**Dept. of Electronics, Ajou University

요 약

본 논문은 차량 및 공장 자동화 분야에서 널리 쓰이고 있는 Controller Area Network 의 안정성 보장을 위한 Fault-Tolerant 프로토콜을 제안한다. 제안된 Fault-Tolerant 프로토콜은 실시간 Fault-Tolerant 시스템을 대상으로 한 Time-Triggered 프로토콜의 중복 메커니즘을 이용하며 event-triggered 방식인 CAN 에 알맞게 변형하여 이용한다. 본 논문의 프로토콜은 Atmel 사의 AT89C51CC03 을 이용하여 구현하여 가능성을 검증 하였다. 제시한 프로토콜을 이용하여 엔진과 X-by-Wire, ABS 분야와 같은 안정성-중시 시스템에 좀더 높은 안정성을 부여할 수 있을 것이다.

1. 서론

오늘날 기술의 발전으로 다양한 기능이 추가된 형태의 시스템이 요구되고 있다. 차량 시스템 또한 고객의 요구에 대응해 멀티미디어와 자동화를 추구하면서 시스템이 복잡해지고 있는 추세이다. 차량 시스템의 경우 시스템이 복잡해짐에 따라 배선 또한 복잡해져 이를 위한 해결 방법으로 나온 것이 차량 내 네트워크인 Controller Area Network(CAN)이다.

CAN 은 사무실 내의 PC 들을 연결한 것과 같은 방식으로, 하나의 CAN 버스에 여러 시스템 컴포넌트들을 연결하여 차량의 배선 양을 감소시키는 장점이 있다. 이러한 CAN 을 차량 내에 접목시키는 연구가 활발해지면서 CAN 네트워크에서의 Fault-Tolerance 연구가 대두되고 있다. 차량 시스템은 탑승자의 생명과 연관되어 안전성이 중시되므로 엔진과 핸들, 바퀴들에 의한 오류로 인한 시스템의 오 동작을 방지할 수 있어야 한다.

실시간 Fault-Tolerant 시스템을 위한 프로토콜로 Time-Triggered Protocol(TTP)이 있다. 이는 중복 메커니즘을 이용하여 오류에 대한 내구성을 높여서 안정성을 보장한다. 본 논문에서는 TTP 의 중복 메커니즘을 기반으로 하여 Event-triggered 시스템인 CAN 에서의 Fault-Tolerant 를 구현하고자 한다.

본 논문에서는 CAN 의 통신 방식, TTP 의 중복 메커니즘을 설명한 뒤, TTP 의 중복 메커니즘을 CAN 에 어떻게 적용하였는지를 설명하였다.

또한, AT89C51CC03 을 이용하여 제시한 프로토콜을 구현하여 동작을 확인하였다.

2. 관련연구

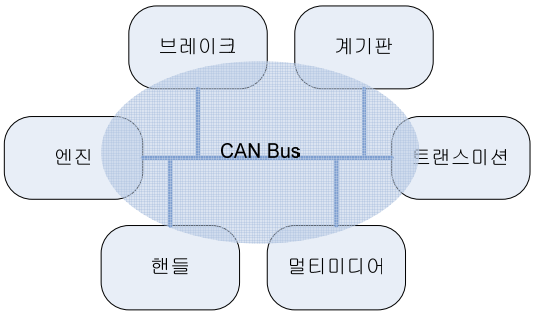
2-1. Controller Area Network (CAN)

Controller Area Network (CAN)은 분산 시스템에서의 빠르고 효율적인 데이터 전달을 위해 제안된 시리얼 통신 프로토콜로 초기에 차량 내 네트워크를 위해 개발되었으며, 현재는 공장 자동화와 의료 장비에 널리 쓰이고 있다.

CAN 네트워크는 (그림 1)과 같이 하나의 버스와 버스에 연결되어 있는 수많은 노드들로 이루어져 있으며, 멀티 마스터 방식을 지원하고 있어 각 노드간의 통신이 가능하다.

CAN 의 버스 통신 방식은 CSMA/CR (Carrier Sense Multiple Access with Collision Resolution)을 사용한다. 이는 ID 비트를 이용한 비트-대-비트 중재를 통해 메시지간의 충돌을 방지한다. 또한, CAN 은 CRC(Cyclic Redundancy Check), 비트 스터핑과 에러 응답 메커니

즘을 통해 에러처리를 제공한다.



(그림 1) CAN 기반의 분산 시스템

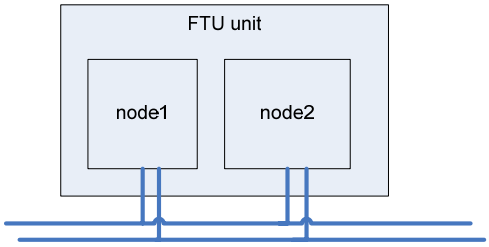
2-2.Time-Triggered Protocol (TTP)

TTP 는 Time-Triggered 구조를 위한 통신 프로토콜로 실시간 fault-tolerant 시스템이 요구하는 다음과 같은 사항들을 만족한다.

- 예측 가능한 메시지 전송
- 클럭 동기화
- Fault tolerance
- 멤버십 서비스
- 중복 메커니즘

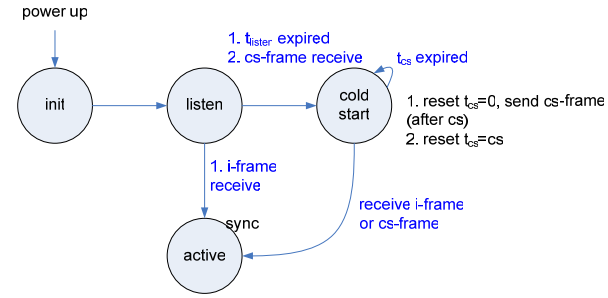
TTP 의 fault tolerance 는 (그림 2)와 같이 노드와 버스를 중복해서 둠으로써 구현된다. 노드를 중복해 둠으로써 한 노드가 오류 상태가 되도 다른 노드가 대신하여 컴포넌트 오류를 보완하고, 버스를 중복함으로써 통신상의 오류를 보완한다. 이때, 동일한 동작을 하는 중복된 노드들의 집합을 FTU(Fault Tolerance Unit)라 부른다.

FTU 의 사용에서 중시되는 것은 중복된 노드 사이의 동작이다. 이는 노드가 추가되었을 때와 오류가 발생하였을 때 각 노드의 동작을 의미한다. TTP 는 Time-Triggered 구조를 기반으로 하므로 시스템 전체의 동작에 대한 정보(a priori)를 각 노드가 예측할 수 있다. 즉, 현 시점에서 어떠한 노드가 어떠한 동작을 할 것이라는 정보를 미리 갖고 있는 것이다. 그러므로, 현 시점에서 예측되는 동작이 이루어지지 않을 경우 동작을 안 한 노드에 오류가 발생하였음을 발견하고 다른 노드가 역할을 대신하게 된다.



(그림 2) Fault Tolerance Unit

또한, 노드가 추가되었을 때는 중복 메커니즘을 따라 (그림 3)과 같은 상태를 보여준다.
각 노드는 시작 시 t_{listen} 시간 동안 기다리면서 i-frame 을 받게 되면 active 상태로 들어간다. i-frame 은 각 노드가 동기화를 맞출 수 있게 주기적으로 보내지는 메시지이다. t_{listen} 시간 동안 i-frame 을 받지 못할 경우 노드는 cold start 상태가 되어 cs-frame 을 보내게된다. cs-frame 을 받은 노드들은 이를 보고 시간을 다시 맞추어 노드간의 동기화가 이루어진다. 이때, t_{cs} 시간은 노드마다 다르게 적용되어 start-up 시의 충돌을 방지한다.



(그림 3) TTP 에서의 중복 메커니즘

3. Fault-Tolerant CAN 프로토콜

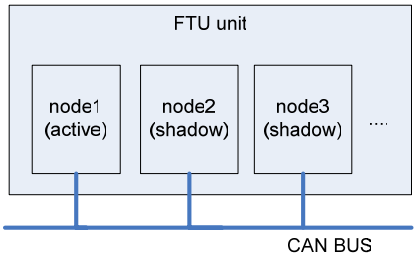
3-1. 프로토콜을 위한 가정

FTU 유닛을 이용한 Fault-Tolerance 구현 시 가장 중요한 것은 중복된 노드들이 동일한 시간에 동일한 동작을 한다는 것이다. 본 프로토콜에서는 [4], [11]와 같은 연구에 의해 클럭 동기화를 구현하여 일정한 시간마다 메시지에 의해 각 노드가 동기화를 맞춘다는 것을 전제로 한다.

또한, 한 노드에서 보낸 메시지를 같은 FTU 내의 모든 노드가 동시에 받고, 순서에 맞게 받음을 보장하기 위해 Consensus protocol[3]을 가짐을 가정한다.

3-2. 시스템 구조

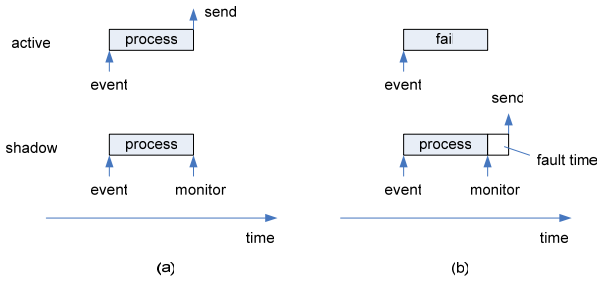
Fault-Tolerant CAN 의 기본적인 구성은 TTP 에서와 같은 FTU 유닛으로 이루어진다. 하지만 중복 노드가 다같이 동작하는 TTP 와 달리 동작(active)하는 노드는 하나, 나머지 노드는 그림자(shadow) 노드로 모니터링을 하게 한다. CAN 의 경우 Event-Triggered 구조를 기반으로 하므로, a priori 정보가 없어서 여러 노드를 동작상태로 둘 경우, 각 노드간의 중복 메세지를 판단하는 메커니즘이 별도로 필요하게 된다. 또한, Fault-Tolerant CAN 에서는 최대 3 개(동작노드 2 개, 그림자 노드 1 개)의 노드를 사용하는 TTP 보다 유동적으로 노드 수에 제한을 두지 않게 프로토콜을 구현하였다.



(그림 4) Fault-Tolerant CAN 에서의 FTU

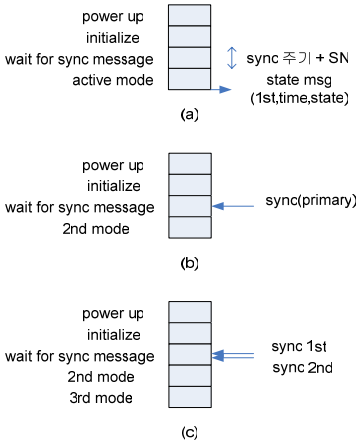
3-3. 상세 프로토콜

앞서 밝힌 바와 같이, CAN 은 event-triggered 구조를 기반으로 하기 때문에, a priori 정보로 오류 발견을 할 수가 없다. 그러므로, Fault-Tolerant CAN 의 그림자(shadow) 노드는 버스를 통해 동작(active) 노드의 동작을 모니터링 하게 된다. CAN 은 event-triggered 방식이므로, 이벤트에 따른 동작을 하게 된다. 그림자 노드는 이벤트와 그에 해당하는 동작을 모니터링 하여 이벤트에 따른 동작이 일정 시간(오류 감지 시간) 동안 없으면 오류라고 간주하고 그림자 노드가 동작 상태가 된다.



(그림 5) (a)정상, (b)오류 발생 시의 노드 동작

중복 메커니즘은 노드의 시작과 재진입 시 동작 노드와 그림자 노드의 구분에 필요하다. 본 논문에서 제안하는 프로토콜은 안정성을 높이기 위해 여러 그림자 노드를 둘 수 있게 하였다. 이와 관련하여 오류 시 동작상태가 될 그림자 노드를 선별하는 메커니즘이 필요하다. 본 논문에서 제안한 프로토콜에서는 그림자 노드간의 순위를 정해서 순위에 따라 동작상태가 될 수 있게 하였다. 순위는 시리얼 번호를 부여하여 시작 시에 결정된다. 각 노드는 시작 시에 초기화를 한 후 클럭 동기화 주기에 시리얼 번호를 더한 시간만큼 기다리게 된다. 기다리는 시간 동안 동기화 메시지가 도착하지 않으면 FTU 내에 다른 노드가 없음을 알고 스스로 동작 상태가 된다. 그러나, 기다리는 동안 동기화 메시지가 도착하면, 그림자 상태로 빠지게 된다 . 이때, 그림자 노드 중의 순위는 도착하는 동기화 메시지의 정보로 판단된다.



(그림 6) Fault-Tolerant CAN 에서의 중복 메커니즘, (a) 동작 노드, (b) 그림자 노드 1, (c)그림자 노드 2

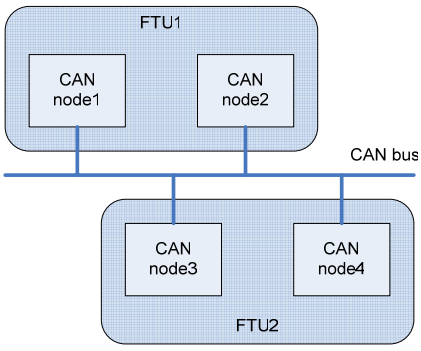
(그림 6)에서 나타나듯이 동기화 메시지에는 각 노드의 순위가 포함되어 보내진다. 이로써, 이 동기화 메시지를 받은 노드는 자신이 어떠한 순위를 갖는지 알게 된다.

재진입 시 또한 동기화 메시지를 이용하여 재진입한 노드의 동기를 맞추고, 동기화 메시지의 정보를 이용하여 자신의 순위를 정하게 된다.

4. 프로토콜 구현

제안한 프로토콜의 구현은 CAN 컨트롤러가 내장된 AT89C51CC03 칩을 이용하여 CAN 노드를 만들어서 테스트하였다.

실험에는 CAN 노드 4 개를 사용하였으며, 다음과 같이 구성하였다.



(그림 7) 실험 환경

어플리케이션은 자동차에서의 X-by-wire 를 가정하여, FTU 1 은 센서 데이터를 전송하고, FTU 2 는 데이터를 받아 처리하는 동작을 한다. 각 FTU 내의 노드들은 동일한 프로그램을 수행하며, 동작 상태일 때와 그림자 상태일 때의 동작이 프로그램에 정의되어있다.

제안한 프로토콜의 구현을 위해, 각 노드를 네트워크에서 분리하였을 때의 결과를 실험해보았다.

<표 1> 시작 시의 각 노드의 상태

	시리얼 번호	시작 시 상태
노드 1	1	동작

노드 2	2	그림자
노드 3	1	동작
노드 4	2	그림자

<표 2> 노드 1 에 오류가 발생했을 경우

	시작 시 상태	오류 발생 후
노드 1	동작	오류
노드 2	그림자	동작
노드 3	동작	동작
노드 4	그림자	그림자

<표 3> 노드 2 에 오류가 발생했을 경우

	시작 시 상태	오류 발생 후
노드 1	동작	동작
노드 2	그림자	오류
노드 3	동작	동작
노드 4	그림자	그림자

<표 4> 노드 3 에 오류가 발생했을 경우

	시작 시 상태	오류 발생 후
노드 1	동작	동작
노드 2	그림자	그림자
노드 3	동작	오류
노드 4	그림자	동작

<표 5> 노드 4 에 오류가 발생했을 경우

	시작 시 상태	오류 발생 후
노드 1	동작	동작
노드 2	그림자	그림자
노드 3	동작	동작
노드 4	그림자	오류

실험 결과를 통해, 각 노드가 시리얼 번호에 따라 동작 상태와 그림자 상태가 정해지고, 동작 노드가 오류시 그림자 노드가 동작상태가 됨을 확인할 수 있다.

5. 결론

본 논문에서는 차량 네트워크에 널리 쓰이고 있는 Controller Area Network(CAN)에 안정성을 보장하기 위해 Fault Tolerance 를 구현하였다. CAN 에 Fault Tolerance 를 구현하기 위해 Time-Triggered Protocol (TTP)에 구현된 FTU 유닛과 중복 메커니즘을 Event-triggered 방식인 CAN 에 알맞게 변경하여 적용시켰다. Fault Tolerant CAN 은 어플리케이션 단에서 수행이 되며, 동작(active) 노드에 그림자(shadow) 노드(동작 노드와 동일한 프로세스를 갖는 다른 노드)들을 두어 동작중인 노드가 오류가 나도 시스템이 제대로 돌아갈 수 있도록 구현되어있다. 실험은 4 개의 CAN 노드로 이루어졌으며, 시작 시 각 노드가 알맞은 상태(동작상태나 그림자상태)를 갖고, 동작 노드가 오류 날 경우 그림자 노드가 동작상태가 됨을 확인하였다.

현재 제안한 프로토콜은 CAN 노드의 오류만을 보장할 수 있기 때문에, CAN 버스를 복제하여 쓰는 경우 또한 생각해야 할 것이다. 또한, 본 논문의 실험에서는 한 FTU 안에 2 개의 노드만을 갖는 환경으로 실험을 하였기 때문에, 더 많은 그림자 노드들로 구성되었을 때의 결과를 추가적으로 실험해야 할 것이다.

또한, 현 프로토콜에서는 event-triggered 방식에서 오류를 발견하기 위해 오류 감지 시간 동안 모니터링을 하여야 하는 문제가 있어 지연을 처리하는 문제가 남아있다.

참고문헌

[1] H.Kopetz, G.Gunter, “TTP – A Protocol for Fault-Tolerant Real-Time Systems”, IEEE Computer, Jan. 1994

[2] J.Ferreira, P.Pedreidas, L.Almeida, J.Fonseca, “Achieving fault tolerance in FTT-CAN”, IEEE, Aug, 2002

[3] G.M.A.Lima, A.Burns, “A Consensus Protocol for CAN-Based Systems”, IEEE Computer, IEEE 2003

[4] G.R-Navas, J.Proenza, “Clock Synchronization in CAN Distributed Embedded Systems”, RTN 2004

[5] H.kopetz, G.Grunsteidl, “TTP – A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems”, IEEE 1993

[6] H.kopetz, A.Damm, C.Koza, M.Mulazzani, W.Schwbl, C.Senft,R.Zainlinger,“The MARS Approach”, IEEE,1989

[7] M.D. Bada, H.Ekiz, A.Kutlu and E.T.Powner. “Toward Adaptable Distributed Real-Time Computer Systems“, IEEE 1996

[8] J.Pimentel, J.Kaniarz, “A CAN-based Application Level Error Detection and Fault Containment Protocol”, 2004 IFAC

[9] J.Pimentel, J.A.Fonseca, “FlexCAN: A Flexible Architecture for Highly Dependable Embedded Applications”, Int. Workshop on Real-Time Networks, Jul. 2004

[10] W.Steiner, J.Rushby, M.Sorea, H.Pfeifer, “Model Checking a Fault-Tolerant Startup Algorithm: From Design Exploration To Exhaustive Fault Simulation”, IEEE Computer, 2004

[11] M.Gergeleti, H.Streich, “Implementing a Distributed High-Resolution Real-Time Clock using the CAN bus”, Can in Automation, 1994