차량네트워크상 신뢰성 테스트를 위한 애플리케이션 개발

강 호 석[†] · 최 경 희[††] · 정 기 현[†††]

요 약

오늘날 차량네트워크(CAN)는 전기적 강인, 저가격과 접근지연 때문에 분산 임베디드 시스템에서 널리 사용되는 버스형 필드이다. 그러나 버스토폴로지에서 발생하는 의존적인 제한 때문에 차량네트워크가 어플리케이션상에서 안전적으로 사용되는지는 논쟁되어왔다. 그래서 차량네트워크(CAN) 디자인 단계 동안 데이터 버스의 부하와 최대 지연, 경쟁 우선순위와 같은 네트워크의 성능을 분석하는 것이 중요하게 되었다. 이 논문은 차량네트워크의 성능을 평가하기 위해 사용된 시뮬레이션 알고리즘과 고장 기법 기술을 적용을 소개한다. 이는 차량네트워크의 어떤 산만한 구현의 줄임과 시스템의 신뢰성을 향상 시켜 줄 것이다.

키워드 : 차량네트워크, 고장 테스트, 부하테스트

Development of an Application for Reliability Testing on Controller Area Network

HoSuk Kang[†] · Kyung Hee Choi[††] · Gi Hyun Jung[†††]

ABSTRACT

Today, controller area network (CAN) is a field bus that is nowadays widespread in distributed embedded systems due to its electrical robustness, low price, and deterministic access delay. However, its use safety - critical applications has been controversial due to dependability limitation, such as those arising from its bus topology. Thus it is important to analyze the performance of the network in terms of load of data bus, maximum time delay, communication contention, and others during the design phase of the controller area network. In this paper, a simulation algorithm is introduced to evaluate the communication performance of the vehicle network and apply software base fault injection techniques. This can not only reduce any erratic implementation of the vehicle network but it also improves the reliability of the system.

Key Words : Controller Area Network, Fault Injection Techniques, Load Test

1. 서 론

최근 자동차에 대한 연비 및 배기에 대한 규제의 강화와 운전자 편의성 증대 욕구에 따라 자동차를 구성하는 전자 시스템 적용이 증가하고 있다. 따라서 차량에 장착된 각종 전자 시스템을 연계시키는 와이어 하네스(wire harness)도 필연적으로 증가하고 있다. 복잡하고 상대적으로 부피가 큰 와이어 하네스는 유지, 보수가 어려울 뿐 아니라 신뢰성 확보에도 문제가 있으며 진단 기능의 구현이 어렵다. 이와 같은 문제점을 근본적으로 해결하기 위해서 사용되기 시작한 것이 CAN(Controller Area Network)이다. 이는 종래의 1대1 네트워크가 아니라 버스형 네트워크 시스템으로 데이터 공유가 가능해질 뿐만 아니라 배선이 말끔해져 중량을 대폭 삭감할 수 있게 되었다.[1] 그러나 네트워크에 적용하게 되면 자료 전송에 있어서 자료의 손실이 야기될 수 있으며 부하로 인하여 제어시스템의 성능이 저하를 가져온다. 또한 운용미숙으로 예기치 않은 조작이 일어날 위험에 많이 노출되어 있다. 이러한 이유로 제어시스템의 신뢰성 및 오작동에 대한 테스트가 필요하다. 이를 위해 본 논문에서는 CAN 인터페이스의 동작을 테스트할 수 있는 프로그램을 구현하기 위하여 2장에서 CAN의 개요를 설명하고, 3장에서는 차량네트워크 시뮬레이션 알고리즘과 실제로 프로그램을 어떻게 구현하였는가에 대해 살펴보고, 4장에서는 차량네트워크상에 소프트웨어 신뢰성 테스트 방법을 소개하며 마지막으로 5장에서 실제 구현한 결과를 보이고 시스템 검증을 하였다.

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 해외교수요원초빙사업의 연구결과로 수행되었음
† 정 회 원 : 파수닷컴 SPA사업본부 연구원
†† 정 회 원 : 아주대학교 정보통신전문대학원 교수
††† 정 회 원 : 아주대학교 전자공학부 교수
논문접수 : 2006년 12월 8일, 심사완료 : 2007년 7월 11일

2. CAN 테스트 시뮬레이션을 위한 알고리즘 개발

2.1 CAN 및 차량 네트워크의 개요

CAN은 분산 전자제어 시스템 상에서 자료를 전달하기 위한 직렬 통신 채널의 하나로 독일의 Bosch에서 처음 제

안하였다. 신호 전달을 위해 두 신호 라인의 전압차를 이용하기 때문에 각종 전기적 노이즈에 강한 내성을 가진다. 또한 강력한 오류 검출 능력과, 최대 1MSbps의 빠른 전송 속도를 갖고 있기 때문에 자동차에의 적용이 늘어가고 있다. CAN 메시지 포맷(그림 1)은 데이터 요청을 위한 원격 프레임, 그리고 오류 발생을 알리기 위한 오류 프레임 등이 있으며 이는 다시 표준형과 확장형의 두 가지로 나뉜다. 표준형과 확장형의 차이점은 중재(Arbitration) 필드를 사용한다. 중재(Arbitration) 필드는 또한 각 메시지의 식별자로도 사용된다. 식별자는 전체 네트워크에서 고유하게 할당된 것으로 숫자가 작을수록 높은 우선순위를 갖는다. 메시지의 데이터(DATA) 필드는 신호을 가지고 있으며 신호 범위를 가지고 있다. 이 범위는 0에서 2^x-1까지 정수 번호를 가진다. 그리고 사용되는 범위를 유효 신호 범위라고 부른다. x는 각각의 메시지 마다 명세서에 정의된 번호의 비트이다.

간헐적인 이벤트는 전송 시마다 데이터가 바뀌는 경우가 많으나, 시스템에 따라 이벤트가 일어날 때마다 같은 메시지가 전송 되어 질 수도 있다[2, 3].

CAN 네트워크 구조(그림 2)는 차내온도, 스로틀 위치,각 조작 스위치의 상태 등과 같은 자료들을 샘플주기 등을 기준으로 정리하고 각각에 고유한 식별자를 부여되며 멀티미디어 부분(MM CAN)과 자동차 부속품 부분(BODY CAN) 사이의 게이트웨이 기능으로 게이트웨이 (듀얼-존 전자동 온도 제어기)가 구성된다. 이는 각 CAN 네트워크로부터 받은 메시지들의 신호을 새로 조합하여 다른 쪽 CAN 네트워크로 전송하는 역할과 자체 메시지를 생성하여 자동차 부속품 부분 또는 멀티미디어 부분 쪽으로 메시지를 주기적으로 보낸다.

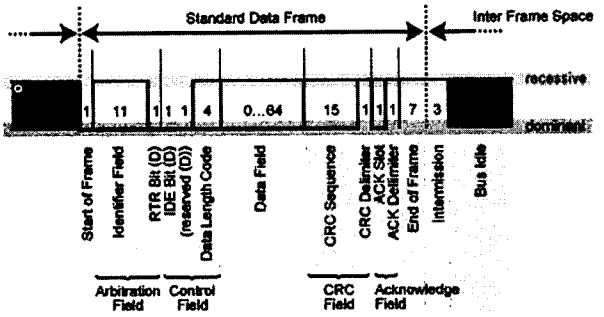
CAN의 버스 통신 방식은 CSMA/CR(Carrier Sense Multiple Access with Collision Resolution을 사용한다. 이는 식별자 비트를 이용한 비트-대-비트 중재를 통해 메시지간의 충돌을 방지한다. CAN의 식별자 비트는 메시지 식별자를 뜻하며, 모든 메시지는 어떠한 식별자를 갖게 된다. 이 식별자로 버스상의 노드들은 자신이 받을 데이터인지 아닌지를 필터링하여 메시지를 받을 수 있게 된다. ID 비트를 이용한 비트 대 비트 중재는 '0'비트를 우선시하여 전송시키며, 이로 인해 식별자가 낮은 메시지가 우선순위를 갖게 된다.

예를 들어 (그림 3)과 같이 여러 노드들이 버스에 아이들 상태 확인한 후 같은 시간에 메시지를 보내려고 한다면 메시지의 충돌은 비트와이즈에 의해 피할 수 있다. 각각의 노드들은 메시지의 식별자를 보내고 버스 레벨에서 모니터링 한다. 위의 그림처럼 노드A와 C가 도미넌트 식별자 비트를 보낸다. 노드 B는 리세시브 식별자 비트를 보낸다. 노드 B는 버스 중재(arbitration)에서 지고 수신모드로 바뀐다. 노드 C의 식별자 비트중에 A노드에 지게 된다. 이것은 노드 A의 식별자가 낮은 바이너리 값을 가지고 있으며, 노드 B와의 메시지보다 우선순위가 높다는 의미이다.

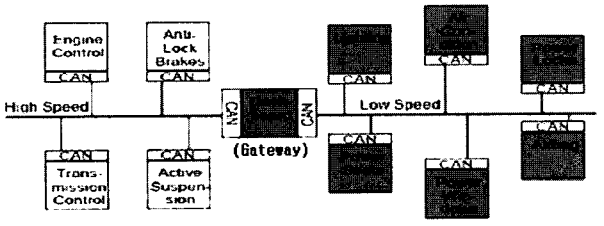
2.2 시뮬레이션 알고리즘

시뮬레이션 알고리즘은 기본적으로 양방향 송수신 구조인 큐의 모델과 시간순서에 의해 메시지정보가 저장된 로그파일을 바탕으로 하는 이산 시간 시뮬레이션 이론을 사용하였다. 시간의 흐름은 연속적이지 않고 이벤트의 발생에 따르게 된다.

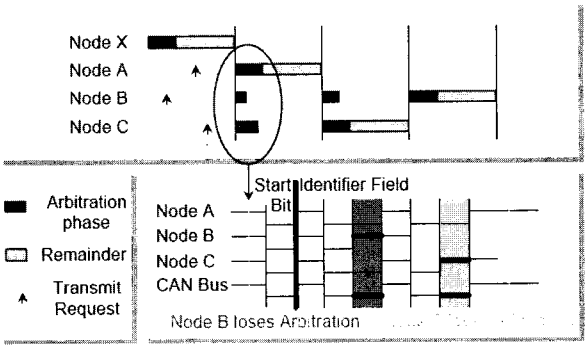
(그림 4)에서 알 수 있듯이 두 가지의 큐를 사용하였다. 송신(Tx) 큐는 각 컨트롤러 내부의 버퍼로써 송신될 메시지를 저장하며 수신(Rx) 큐는 각 컨트롤러 내부의 버퍼로써 수신될 메시지를 저장된다. 따라서 각 컨트롤러 모두에 메시지를 송수신이 가능하게 되어 양방향 인터페이스 테스트 구조로 모델링 된다. 여기서 큐의 크기를 임의로 설정할 수 있도록 하여 사용하는 컨트롤러의 사양에 맞추어 적절히 조절할 수 있다. 그리고 메시지를 분석하기 위해서 송수신된 메시지정보가 저장된 로그 파일을 사용하였다. 메시지 분석 방법(그림 5)은 로그파일을 읽어 들여 식별자별로 메시지들을 추출하여 식별자별 메시지-Set파일들을 구성하며 이렇게 구성된 파일을 이용하여 형태별 메시지 분석 알고리즘을 통해 게이트웨이가 올바르게 반응하였는지에 대한 분석을 하게 된다. 여기서 식별자별 메시지-Set파일은 로그파일에 있는 송신된 메시지와 그것의 결과에 해당하는 수신메시지 집합들이다.



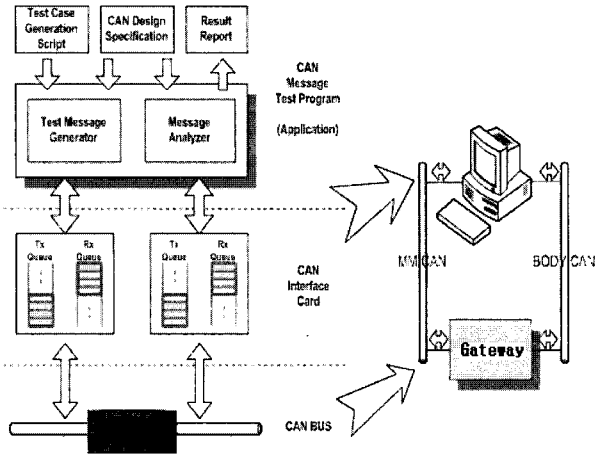
(그림 1) CAN 메시지 포멧



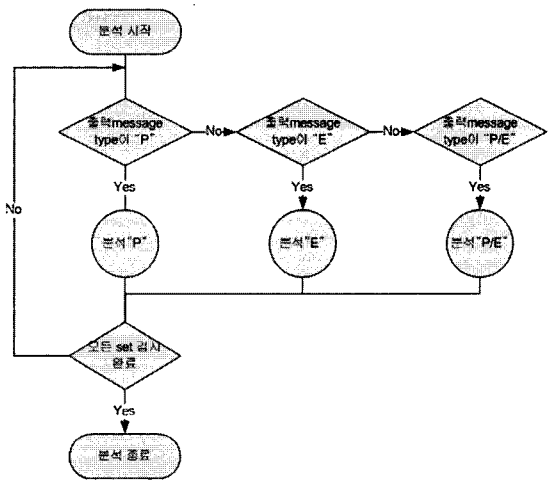
(그림 2) CAN 네트워크 구조



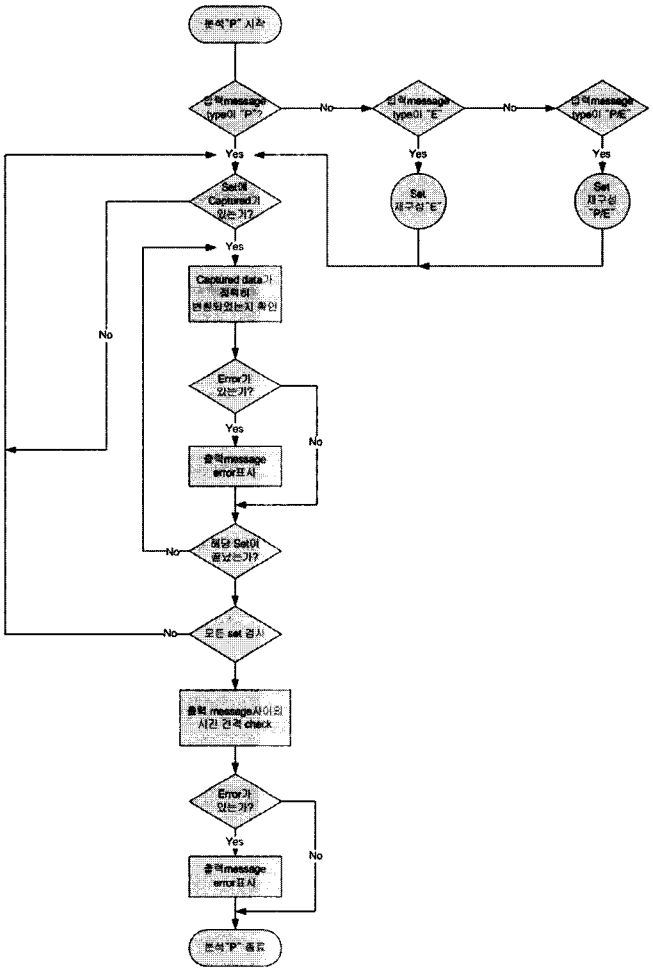
(그림 3) CSMA/CD 방식



(그림 4) 송수신 큐(Queuing) 이론



(그림 5) 메시지 분석 알고리즘



(그림 6) 주기성 이벤트 분석 알고리즘

형태별 분석방법은 크게 주기성 이벤트, 간헐적 이벤트 형태로 구분된다.

주기성 이벤트일 경우에 (그림 6)알고리즘과 같이 200ms 간격으로 메시지가 발생하기 때문에 전송 메시지 사이의 간격이 200ms인지를 확인된다 즉 전송 메시지의 데이타가 수신 메시지의 데이타에 반영이 되었는지 확인 후 수신 메시지간에 200ms 확인한다. 이 중 하나의 메시지에 오류가 있을 경우 오류가 표시된다. 간헐적 이벤트일 경우는 전송 메시지의 데이타가 수신 메시지의 데이타에 반영이 되었는지 확인 후 수신 메시지간에 40ms 간격으로 3회 메시지가 발

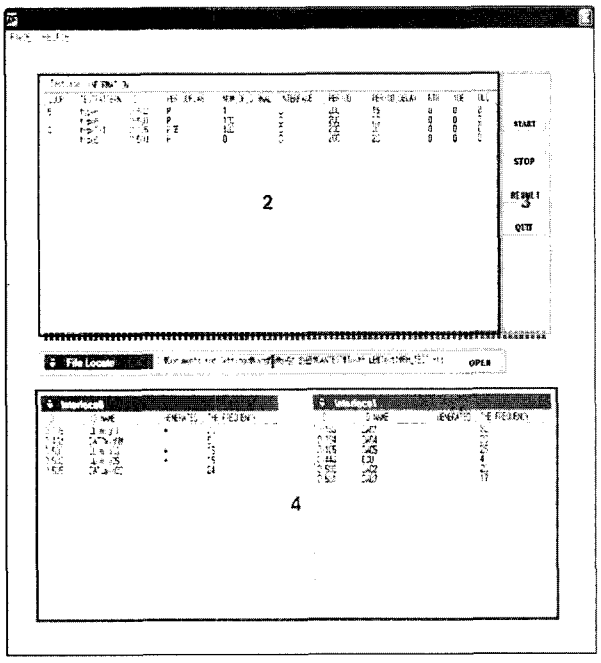
생했는지 확인된다.

3. 시뮬레이터 구성

이 도구는 CANTester라고 불리어지며, 시스템의 CAN 인터페이스 동작을 테스트할 수 있는 목적으로 제작되었다. 이 도구의 구성은CAN 메시지 생성 및 전송, 설정한 메시지 스케줄링, 송수신 정보저장 및 분석 부분으로 되어진다. 이는 위의 알고리즘을 바탕으로 MS의 Visual C++을 사용하여 시뮬레이터를 제작하였다. CAN 컨트롤러에 해당하는 노드 버퍼의 크기 각 메시지의 속성 등의 설정이 가능하며 노드와 메시지 추가, 삭제가 및 설정 내용의 저장 기능도 갖추고 있다. 또한 전송주기를 설정할 수 있어서 부하테스트가 가능하다. (그림 7)은 차량네트워크 설정 윈도우의 모습이다.

3.1 CAN 메시지 생성 및 전송

CANTester는 CAN 포멧이 정의된 테스트 스크립트를 통해 하나의 패킷 인스턴스들을 생성시킨다. 이것은 다음과 같은 방법으로 진행 된다. 첫 번째로 메시지 공간이 할당된다. 그리고 나서 할당된 공간에 테스트 스크립트에 의한 조건들이 CAN 포멧 헤더에 적용된다. 다음은 시그널 값의 범위가 명세서에 의해 인정되는 경우 그 값은 유효한 값의 범위로부터 랜덤하게 선택되어 패킷 Payload에 복사 되어진다. 마지막으로 검사합계 계산되고 전송형태에 맞게 패킷이 IXXAT library를 통해 전송된다.[6] 예를 들어 (그림 8)에



번호	설 명
1	연결된 Test Script 위치정보
2	연결된 Test Script의 TestCase에 대한 정보를 나타내는 패널
3	Message 생성 및 송신 Control 장치
4	CAN 인터페이스별 송수신 Message 개수 및 정보를 나타내는 패널

(그림 7) 시뮬레이터 메인화면

있는 테스트 스크립트가 있다면 500개의 0x500 메시지가 주기적으로 시그널 값이 바뀌면서 생성되며 콘트롤 0번을 통해 버스에 전송된다.

3.2 메시지 스케줄링

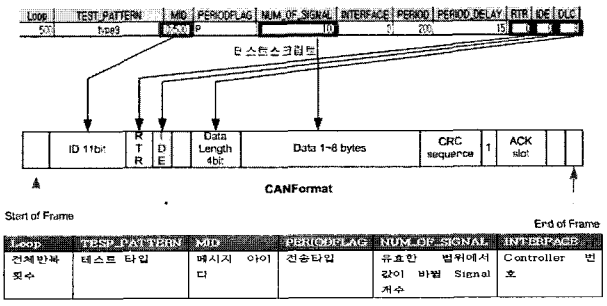
스케줄링은 메시지 리스트의 각 메시지의 전송형태, 시작시간 주기를 바탕으로 동작한다. 메시지 전송형태가 주기적 이벤트이면 이벤트 발생시점의 메시지가 주기시간으로 순차적으로 더해진다. 즉 주기적으로 메시지를 전송되다가 이벤트가 발생하면 신호값이 바뀐메시지를 주기적으로 전송하게 된다. 만약 간헐적 이벤트이면 이벤트 발생시점에 시간에 랜덤값만 더해지게 된다. 이벤트 시간이 계산된 메시지들은 시간순서로 정렬되어 파일에 저장된다. 파일에 저장된 시간은 타이머에 셋팅 되면 타이머가 파일에 저장된 시간순서로 전송함수를 인터럽트 시킨다.

3.3 송수신한 메시지 분석

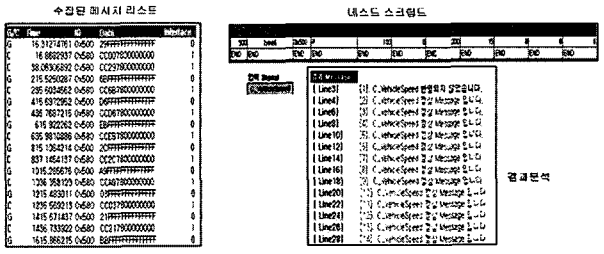
CANTester의 기본기능은 게이트웨이에 송수신된 각각의 패킷을 분석하고 설정된 규칙에 맞게 송수신 되었는지 찾기 위한 것이다. 이 규칙은 CAN디자인 명세서에 의해 송수신된 메시지의 시그널 위치변화와 전송형태를 명시하고 있다. 즉 주어진 조건을 모두 만족하면 메시지가 정상 처리되었음을 기록한다. 그러므로 메시지 분석은 메시지 필드별 parsing을 통해 송수신된 메시지의 각 필드가 조건에 만족하는지 판단하며 전송형태의 구분을 포함한다. (그림 9)은 CANTester에 의해 출력된 결과분석 파일이다.

4. 실험 방법

이 장은 시뮬레이션 결과검증, 게이트웨이 오작동 테스트 위하여 실제 차량네트워크를 구성하여 게이트웨이 장치를



(그림 8) 메시지 생성과정



(그림 9) 결과분석파일

측정하여 보았다.

테스트 환경(그림 10)은 다음3가지 환경으로 구성된다. 첫 번째는 게이트웨이 장치 없이 CAN 메시지가 버스에 전송되는 시스템 환경에서 측정되었으며 두번째는 정상적인 메시지가 게이트웨이에 전송되는 시스템 환경에 측정되었으며 마지막으로 고장(fault) 메시지가 게이트웨이에 전송되는 시스템 환경에서 측정되었다.

여기서 CAN Tester는 Window XP하에 CAN 명세서에 명시된 제약(constraint)에 만족하는 CAN 메시지를 발생시키며 고장(fault) 메시지는 CAN 명세서과 다르게 CAN 프레임 필드에 고장(fault)를 주입하였다. 이 실험에서 CANTester에 의해 생성된 메시지가 제대로 전송되는지 확인하기 위해 게이트웨이장치 없이 버스에 전송되는 메시지가 검사 되어졌으며, 이 실험의 결과는 CAN 프레임 필드에 의해 분류 되어진다. 또한 다음 환경에서 입력메시지 간격을 줄이면서 게이트웨이의 성능이 측정되었다.

4.1 고장(fault) 주입 방법

장비에 오작동으로 인하여 전달되는 메시지에는 다양한 형태의 고장(fault)가 주입될 수 있다. 본 논문에서는 구문 고장(fault)를 적용하여 장비를 테스트 하였다[4]. 구문 고장(fault)는 메시지의 구조나 기술 방법, 값의 형태나 범위상에 고장(fault)가 발생된 것을 가리킨다. 즉 명세서에서 허용되지 않는 값의 사용으로 기인한 고장(fault)를 생성한다.

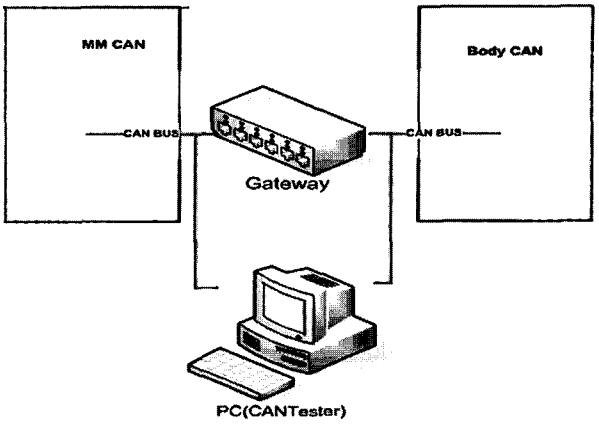
이 실험은 CAN 식별자가 A(E), B(P), C(P), D(P)인 메시지를 입력 메시지로 사용하였으며 <표 1>의 조건으로 고장(fault) 주입 실험을 하였다.

4.2 성능평가 이론

게이트웨이가 처리해야 할 일을 지정하는 메시지가 많을 때 게이트웨이가 자체 생성하여 주기적으로 보내는 메시지를 정확한 시간에 보내는가를 확인하며 또한 브리지 테스트 기능을 제대로 수행하는가를 확인하며 CANTester을 이용하여 100초 동안 게이트웨이가 처리해야 할 입력 메시지를 평균 200, 100, 50, 20, 15ms 간격으로 <표 2> 입력조건으로 게이트웨이에 전송했을 때 브리지 테스트확인과 게이트웨이 생성 메시지에 대해서 시간간격을 체크하였다.

5. 실험결과

이 장은 CANTester 툴에 의해 테스트 되었던 게이트웨이의 반응 결과를 보여준다.



(그림 10) 테스트 수행 환경

〈표 2〉 부하테스트 조건

입력 Message (전송타입)	출력 Message (전송타입)	입력방향
D(P)	D′(P)	MIM CAN → Body CAN
C(P)	C ′(P/E)	MIM CAN → Body CAN
B(P)	B′(E)	MIM CAN → Body CAN
A(E)	A′(E)	Body CAN → MIM CAN
ALL	D′(P), C ′(P/E), B′(E), A′(E)	MIM CAN → Body CAN

〈표 3〉 시뮬레이션 결과

입력 Message	출력 Message	Response	Result
A	A′	CAN Specification에 준하여 출력 Message가 나왔다.	Good
B	B′		
C	C′		

5.1 시뮬레이터 검증결과

이 실험은 명세서에 맞는 CAN 메시지를 게이트웨이에 전송하여 게이트웨이가 명세서에 맞게 CAN 메시지를 처리하는지에 대한 반응을 확인하기 위해서이다. <표 3>와 같이 다음과 같은 입력 메시지를 게이트웨이에 전송한 결과 장비는 CAN 명세서에 준하는 응답을 전송하였다.

〈표 1〉 구문 고장(fault) 형태

항목	Fault Type	정상	Fault
IDE	F1	ID Field의 크기를 0으로 고정	ID Field의 크기를 1(29bit)으로 고정
Signal	F2	Spec에 정의된 범위	Spec에 정의되지 않은 범위
ID	F3	Spec에 정의된 ID들	Spec에 정의 되지 않은 ID들
P.P/E type	F4	200ms간격의 전송	600ms 또는 1s이상 전송 없음
E type	F5	3회 40ms간격의 전송	4회 이상 40ms간격의 전송
전송 time	F6	같은 ID 메시지는 20ms 이상 범위에서 전송	같은 ID 메시지는 20ms이하 범위에서 전송

〈표 4〉 고장(fault) 주입 결과

Fault Type	입력 Message	출력 Message	Response	Result
F1	A	없음	응답없음	Failure
	B	없음		
	C	없음		
F2	A	A′	Invalid로 표시된 Signal 값을 가지 메시지가 발생	
	B	B′		
	C	C′		
F3	A	없음	Fault ID를 가진 메시지를 Drop으로 응답이 없음	
	B	없음		
	C	없음		
F4	A	A′	횟수에 상관없이 A′메시지가 40ms 간격으로 3회 출력	
F5	B	B′	비주기적인 입력 메시지와 상관없이 정상적으로 작동	
	C	C′		
F6	A	A′	입력 메시지에 반응하여 출력되는 메시지가 나오지 않는 경우도 있음	Warning
	B	B′		
	C	C′		

5.2 고장(fault) 주입 반응실험

메시지의 각 필드마다 구문 고장(fault)를 발생시킨 고장(fault) 패킷들로 테스트를 수행하였을 때, 장비의 오작동 여부를 테스트하였다. <표 4>는 게이트웨이 장비에 대하여 구문 고장(fault)를 포함하는 고장(fault) 패킷으로 장비를 테스트한 결과이다. <표 4>에 있는 각 고장(fault) 형태별로 게이트웨이 장비를 테스트한 결과를 정리 한 것이다.

'Warning'이라 표시된 부분은 비록 CAN 프로토콜에 어긋나지만 장비가 지속적으로 정상적인 서비스를 제공할 수 있었다는 것을 가리킨다. 반면 'Failure'는 테스트 장비가 CAN 프로토콜을 사용할 수 없는 심각한 오류를 말한다.

테스트 결과, 고장(fault) 패킷을 사용하여 테스트를 하였음에도 장비가 이를 인지거나 명세서에 준하게 처리하는 등의 동작을 보이는 경우<표 4>의 Result 부분에 표시하지 않았다. 게이트웨이 장비에 대하여 구문 고장(fault) 테스트를 수행한 결과 테스트에 사용된 고장(fault) 패킷에 대하여 테스트 장비는 타당하게 처리하지 못하는 상황이 존재하였다. 고장(fault) 형태 F1의 경우처럼 확장형 포멧 모드로 CAN 네트워크을 설정하여 게이트웨이를 연결하였을 때 게이트웨이는 자체 생성하여 주기적으로 발생시키는 메시지가 나오지 않았으며, 또한 입력 메시지에 대한 브리지 테스트 기능도 상실되었다.

5.3 부하 반응실험

이 장은 각각의 메시지를 게이트웨이에 전송하였을 때 게이트웨이 반응을 확인한 결과이다. <표 5>은 100초 동안 각각의 입력 메시지의 전송간격을 변화시키면서 게이트웨이에 전송하였을 때 게이트웨이가 자체 생성하여 출력하는 메시지들의 손실률이다.

실험결과 게이트웨이에 입력된 메시지에 의해서 반응하는 출력 메시지 중 식별자가 낮은 메시지가 많이 출력될수록 게이트웨이 자체 생성하여 주기적으로 출력하는 메시지 손실률이 높게 나타났으며 출력 메시지 보다 식별자가 낮은 주기적인 메시지는 손실되지 않았다. 실험결과 게이트웨이에서 같은 방향으로 출력되는 메시지 중 우선순위가 높은 메시지가 많으면 이 메시지에 의해 경쟁에 진 메시지들의 손실률은 커졌다. 또한 게이트웨이에 출력되는 메시지의 목적지가 다르면 서로 경쟁을 하지 않음을 보여준다. <표 5>

〈표 5〉 게이트웨이가 자체 생성하여 출력하는 메시지들의 손실률

입력시간	입력 Message	출력 Message	Loss Message Number				
			D′(P)	D′b(P)	C′b(P/E)	C′(P/E)	E′(E)
200ms	D(P)	D′(P)	-	0(0.0%)	0(0.0%)	0(0.0%)	0(0.0%)
	C(P)	C′(P/E)	45(9.0%)	37(7.4%)	0(0.0%)	-	0(0.0%)
	B(P)	B′(P/E)	42(8.4%)	40(8%)	38(7.6%)	49(9.8%)	0(0.0%)
	A(E)	A′(E)	0(0.0%)	0(0.0%)	0(0.0%)	0(0.0%)	159(31.8%)
100ms	D(P)	D′(P)	-	0(0.0%)	0(0.0%)	0(0.0%)	0(0.0%)
	C(P)	C′(P/E)	54(10.8%)	34(6.8%)	0(0.0%)	-	0(0.0%)
	B(P)	B′(P/E)	56(11.25%)	50(10.0%)	48(9.6%)	55(11%)	0(0.0%)
	A(E)	A′(E)	0(0%)	0(0%)	0(0.0%)	0(0.0%)	222(44.4%)
50ms	D(P)	D′(P)	-	0(0%)	0(0.0%)	0(0.0%)	0(0.0%)
	C(P)	C′(P/E)	58(11.6%)	38(7.6%)	0(0.0%)	-	0(0.0%)
	B(P)	B′(P/E)	61(12.2%)	52(10.4%)	71(14.2%)	74(14.8%)	0(0.0%)
	A(E)	A′(E)	0(0%)	0(0%)	0(0.0%)	0(0.0%)	278(75.6%)
20ms	D(P)	D′(P)	-	0(0%)	0(0.0%)	0(0.0%)	0(0.0%)
	C(P)	C′(P/E)	84(16.8%)	83(16.6%)	0(0.0%)	-	0(0.0%)
	B(P)	B′(P/E)	78(15.6%)	70(14.0%)	84(16.8%)	89(17.8%)	0(0.0%)
	A(E)	A′(E)	0(0%)	0(0%)	0(0%)	0(0.0%)	247(49.4%)

에 의하면 A 메시지를 게이트웨이에 입력했을 때 자동차 부속품 부분으로 출력되는 E' 메시지만 손실이 증가되었다.

<표 6>은 각각 입력 메시지를 게이트웨이에 전송하였을 때 게이트웨이에 출력하는 브리지 테스트 메시지의 손실률이다. 여기서 총 메시지 수(C)는 각각의 메시지를 게이트웨이에 전송했을 때 게이트웨이에 의해 브리지 테스트 되어 나온 메시지 개수이다.

입력 메시지의 전송간격을 변화시키면서 게이트웨이에 전송했을 경우 게이트웨이에 브리지 테스트된 메시지 손실들의 손실률은 증가되지 않았다. 그러나 <표 6>에 의하면 이벤트 형태인 A 메시지에 의해 반응한 A' 메시지의 손실률은 증가하였다.

<표 7>은 20ms, 15ms간격으로 메시지를 게이트웨이에 전송하였을 때 게이트웨이에 출력되는 브리지 테스트 메시지의 발생률을 나타낸다. 여기서 총 메시지 수(G)는 각각의 메시지를 게이트웨이에 전송한 총 메시지 개수이다. 아래의

〈표 6〉 브리지 테스트된 메시지 손실들

입력시간	입력 Message	Total Message Number(C)	Loss Message Number			
			D'(P)	C'(P/E)	B'(E)	A'(E)
200ms	D(P)	500	0(0.0%)	-	-	-
	C(P)	1920	-	0(0.0%)	-	-
	B(P)	1500	-	-	0(0.0%)	-
	A(E)	1500	-	-	-	0(0.0%)
	D(P)	500	0(0.0%)	-	-	-
100ms	C(P)	3001	-	2(0.1%)	-	-
	B(P)	2698	-	-	2(0.2%)	-
	A(E)	2879	-	-	-	39(1.6%)
	D(P)	500	0(0.0%)	-	-	-
50ms	C(P)	3708	-	2(0.1%)	-	-
	B(P)	3767	-	-	2(0.1%)	
	A(E)	3954	-	-	-	측정불가

〈표 7〉 브리지 테스트된 메시지 반응률

입력 입력시간	입력 Message ID	Total Message Number(C)	Tranststed Message Number			
			D'(P)	C'(P/E)	B'(E)	A'(E)
200ms	D(P)	5000	500	-	-	-
	C(P)	5000		4896(97.9%)	-	-
	B(P)	5000		-	4975(99.5%)	-
	A(E)	13000		-	-	4862(37.4%)
	INPUT(ALL)	28000	438	4063(91.2%)	4968(99.3%)	4843(37.2%)
15ms	D(P)	6500	500	-	-	-
	C(P)	6500		4783(73.5%)	-	-
	B(P)	6500		-	4798(73.8%)	-
	A(E)	14506		-	-	4858(33.3%)
	INPUT(ALL)	34006	437	3802(58.4%)	3836(74.4%)	4779(32.9%)

조건은 CANTester로 게이트웨이에 출력된 메시지의 손실률 확인할 수 없으므로 반응한 메시지의 수로 게이트웨이의 기능을 확인하였다. 그 이유는 PC에 의해 게이트웨이에 메시지가 전송되면 게이트웨이는 이 메시지의 반응으로 출력 메시지가 40ms간격으로 3회 발생시키지만 게이트웨이에 출력 메시지가 발생하기 전 다른 신호를 가진 입력 메시지가 게이트웨이에 전송되면 오직 뒤에 전송되었던 입력 메시지에 반응하는 출력 메시지만 출력되기 때문이다.

실험결과 20ms간격부터 손실률이 보이기 시작하며 70% 이하의 반응률은 CAN 프로토콜에 어긋난다. 즉 입력된 메시지에 게이트웨이는 반응 메시지를 출력하지 않을 수 있다. 그러나 게이트웨이 장비는 오작동을 하거나 중지되지 않고 지속적으로 정상적인 서비스를 제공하였다.

6. 결 론

본 논문에서는 차량네트워크로 CAN을 사용하는 경우에 대하여 시뮬레이션 알고리즘과 시뮬레이터를 개발하였다. 이는 사용자가 작성한 테스트스크립트에 의해서 테스트케이스가 자동으로 생성되며, 생성된 테스트벡터는 테스트 될 시스템에 전송 되어 진 후 이에 대한 반응 메시지를 수신하여 자동으로 시스템을 분석할 수 있게 구현하였다. 위 알고리즘은 주기적인 이벤트는 물론 간헐적인 이벤트도 처리할 수 있으며, 차량 네트워크의 노드간의 메시지 전송상태를 알 수 있었다. 특히 각 메시지의 시그널에 따라 게이트웨이의 반응을 정확하게 분석하므로 게이트웨이에 반응하여 출력된 메시지의 데이터에 따른 분석이 용이 하였다. 또한 구문 고장(fault)를 테스트함에 있어 장비가 주고 받는 메시지 중 어떤 메시지가 취약점을 지니고, 어떤 필드가 취약점을 지니며, 어떤 형태의 고장(fault)에 취약점을 가지는지 파악할 수 있었으며 차량네트워크 성능테스트에서는 전송지연이 미리 설정된 범위 이내가 되기 위한 차량네트워크 속도와 그때의 네트워크의 부하를 알아 볼 수 있었다. 따라서 차량네트워크를 설계할 때, 시스템 구축에 앞서 손쉽게 네트워크의 구조 변경 테스트 및 최적화가 가능하다.

참 고 문 헌

[1] Wolfhard Lawrenz, "CAN System Engineering," Springer, 1997.

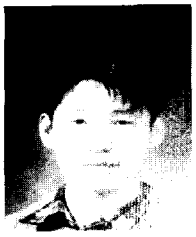
[2] Bosh, "CAN 명세서 Version 2.0," Bosch 1991.

[3] K.Estchber "Controller Area Network Basics,Protocols,Chips and Application IXXAT Automation GmbH, Weingarten, Germany, 2001.

[4] Lijun Shan, Hong Zhu, "Test are generation: Testing software modeling tool using 데이타 mutation," AST, 2006.

[5] 양성모,"자동차 네트웍 시스템 분석을 위한 시뮬레이션 알고리즘 개발에 관한 연구," 1999.

[6] VCI-Virtual CAN Interface VCI-V2 Programmers Manual, Software Version 2.16, IXXAT Automation GmbH.

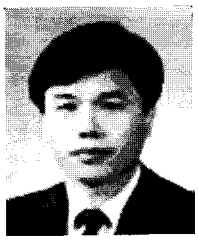


강 호 석

e-mail : inspecter@freechal.com
2004년 아주대학교 공과대학 정보및컴퓨터공학(학사)
2007년 아주대학교 정보통신전문대학원 정보통신공학과(공학 석사)
2007년~현 재 파수닷컴
관심분야 : 실시간 시스템, 소프트웨어 공학

최 경 희

e-mail : khchoi@madang.ajou.ac.kr
1976년 서울대학교 사범대학 수학교육과 (학사)
1979년 프랑스 그랑데꼴 Enseeiht 정보공학과 (공학석사)
1982년 프랑스 Paul Sabatier 정보공학과 (공학박사)
1982년~현 재 아주대학교 정보통신전문대학원 교수
관심분야 : 운영체제, 분산시스템, 실시간 및 멀티미디어 시스템 등



정 기 현

e-mail : khchung@madang.ajou.ac.kr
1984년 서강대학교 공과대학 전자공학 (학사)
1988년 미국Illinois주립대 EECS(공학석사)
1990년 미국Purdue대학 전기전자공학부 (공학박사)
1991년~1992년 현대전자 반도체 연구소
1993년~현 재 아주대학교 전자공학부 교수
관심분야 : 컴퓨터구조, VLSI설계, 멀티미디어 및 실시간 시스템 등