제33회 한국정보처리학회 춘계학술발표대회 논문집 제17권 제1호 (2010. 4)

무인 비행체를 위한 경량 실시간 프로토콜 기반의 ARINC653 지원

이상헌, 진현욱
건국대학교 컴퓨터공학부
e-mail: {mir1004, jinh}@konkuk.ac.kr

Supporting ARINC653 Using a Lightweight Real-Time Communication Protocol for Unmanned Aerial Vehicles

Sang-Hun Lee, Hyun-Wook Jin
Department of Computer Science & Engineering, Konkuk University

요 약

ARINC 653은 통합형 항공전자 시스템에서 사용되는 응용프로그램 간의 인터페이스와 실시간 운영체제의 표준을 정의한다. ARINC 653에 정의되어 있는 네트워크 통신 인터페이스는 대형 항공기뿐만 아니라 무인 비행체와 같은 작은 항공체에도 적용이 가능하다. 이러한 작은 시스템에서는 TCP/IP와 같이 무거운 프로토콜보다는 경량의 실시간 프로토콜이 적합하다. 본 논문에서는 RTDiP을 이용하여 ARINC 653의 통신 인터페이스 중에서 Queuing-mode를 구현하고 성능 측정을 수행한다.

1. 서론

현대 항공 전자 시스템은 항공전자기술이 발전함에 따라 통합형 항공전자 시스템(Integrated Avionics System)으로 발전해 왔다. 통합형 항공 전자시스템은 각 모듈마다 특정한 임무를 부여하고 표준 데이터 네트워크로 서로 통신 한다. ARINC653[1]은 이런 항공 시스템에서 사용되는 응용프로그램 간의 인터페이스와 실시간 운영체제의 표준을 제시한다. 그리고 항공기 안에 존재하는 각 모듈 간 실시간 커뮤니케이션을 수행 할 수 있도록 통신을 위한 프로그램 인터페이스도 제공한다. 이와 같은 특징으로 인해 ARINC653은 HELISCOPE Project[2]와 같이 몇 개의 임베디드 보드들로 구성된 소형 무인 헬기에도 적용이 가능하다. 그렇지만 대형 항공기와는 달리 소형 비행체에 ARINC653 통신 표준을 실제로 구현하기 위해서 널리 사용되는 통신 프로토콜, TCP/IP 또는 UDP/IP 같은 프로토콜들은 사용하기 적합하지 않다. 소형 비행체의 경우 소형 단일 네트워크로 구성된다. 이런 소형 단일 네트워크 환경에서는 네트워크 혼잡에 의한 패킷의 손실이 일어나지 않으며, 통신을 위해 IP주소가 아닌 MAC 주소로도 충분하다. 이 외에도 TCP/IP와 같은 프로토콜은 소형 단일 네트워크에 필요한 기능보다 너무 많은 기능을 제공하기 때문에 많은 오버헤드를 지니고 있다. 그렇기 때문에 기존의 통신 프로토콜을 소형 무인 비행체에 적용시키기에는 많은 오버헤드가 발생한다. 따라서 소형 무인 비행체에도 적용이 가능한 보다 경량화 되고 실시간성을 제공할 수 있는 프로토콜이 필요하다. RTDiP[3][4][5]은 리눅스 기반의 매우 가벼운 프로토콜로써 소형 무인 비행체 내부와 같은 단일 임베디드 네트워크에 적합한 프로토콜이다. 또한 ARINC653의 통신 인터페이스를 구현하기 적합한 구조를 지니고 있다.

본 논문에서는 리눅스 시스템 상에 ARINC653의 내용 중 통신을 위한 인터페이스인 Queuing-mode를 경량화 프로토콜인 RTDiP을 이용해 구현하고 그 성능을 측정한다.

본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT 연구센터 지원 사업(NIPA-2010-C1090-1031-0003)과 교육과학기술부의 세계수준의 연구중심대학 육성사업(#R33-2008-000-10068-0)의 연구결과로 수행되었음.

2. 배경지식

본 장에서는 경량 실시간 프로토콜인 RTDiP과 항공시스템에서 사용되는 ARINC653 중 통신 인터페이스에 관하여 살펴본다.

2.1 RTDiP

RTDiP은 Real-Time Direct Protocol의 약자로써 리눅스 기반의 실시간 통신을 위한 경량 프로토콜이다. RTDiP은 통신을 위해 MAC 주소만을 사용하며 하나의 통신 연결 안에 존재하는 네트워크 패킷마다 우선순위를 설정할 수 있다. 그리고 송신측에서 사용자 프로세스에서 네트워크 컨트롤러로 데이터를 바로 전송하기 때문에 커

널버퍼로 데이터를 복사하는 오버헤드를 가지고 있지 않다. RTDiP의 큰 특징 중 하나는 Bottom half를 사용하지 않는다는 것이다. 이를 통해 RTDiP을 사용하는 사용자 프로세스는 프로세스 우선순위에 따라 네트워크 데이터를 전달 받을 수 있으며 다른 우선순위가 낮은 네트워크 패킷들로 인한 지연시간을 줄일 수 있다. RTDiP은 커널 모듈과 사용자 API로 구성되어 있으며 별도의 커널 수정 없이 사용 가능하다. 위와 같은 RTDiP의 특징으로 인해 RTDiP은 소형 단일 네트워크에 적합한 좋은 성능을 보여준다.

2.2 ARINC653

ARINC653은 항공전자시스템을 위한 표준으로써 실시간 운영체제와 그 위에서 동작하는 프로그램간의 인터페이스를 규정한다. ARINC653의 모든 프로그램 인터페이스들은 Partitioning OS상에서 동작하도록 정의되어 있으며 각 파티션 간의 통신을 위해서 Queuing-mode와 Sampling-mode를 지원한다.

Sampling-mode는 송신측에서 전송한 모든 메세지가 수신측에 저장되지 않는다. 오직 하나의 메시지만을 저장한다. 만약 하나의 메시지가 수신측에 도착하여 사용자 프로세스가 데이터수신을 요청하기 전에 또 다른 메시지가 도착한다면, 이전 메시지 위에 새로운 메시지가 덮어 써진다. 따라서 수신측 버퍼에는 항상 마지막 메시지만 존재하게 된다.

Queuing-mode는 송신측에서 보낸 메시지를 수신측에서 모두 간직하고 있는 통신 인터페이스이다. 사용자 프로세스가 수신요청을 하면 수신측에 도착한 메시지들을 도착한 순서대로 사용자 프로세스에 전달한다.

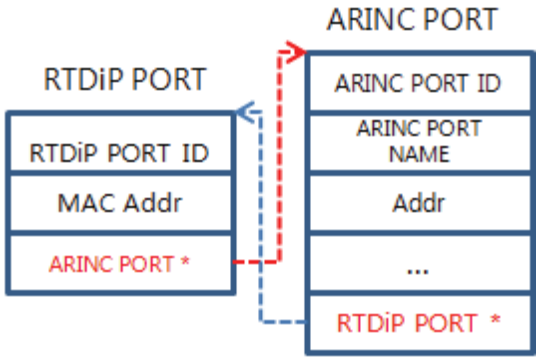
Queuing-mode는 포트와 채널을 통해 서비스 된다. 포트와 채널의 설정내용은 시스템 설정 때 결정 된다. 그리고 통신이 수행중일 때 포트와 채널의 환경 설정 값은 변경이 불가능 하다. Queuing-mode는 포트마다 전송방향이 존재한다. 송신포트로 지정된 Queuing-mode 포트는 오직 데이터를 송신만 가능하고, 반대로 수신포트는 데이터를 송신할 수 없다. 본 논문은 2개의 ARINC653 통신 인터페이스 중 Queuing-mode에 중점을 두어 연구한다.

3. RTDiP 기반의 ARINC653 구현

RTDiP을 이용하여 ARINC653의 Queuing-mode를 구현할 때 다음과 같은 점을 유의하여 구현을 수행하였다. 우선 RTDiP이 지원하는 통신상의 장점을 최대한 유지 할 수 있는 구조로 설계한다. 그리고 ARINC653으로 통신을 수행 하였을 때 RTDiP의 통신 오버헤드 보다 많은 오버헤드가 발생하지 않도록 구현하는 것을 목표로 설정한다. 하지만 ARINC653과 RTDiP 사이에 다른 점이 존재하기 때문에 몇 가지 문제점이 발생한다. 본 장에서는 ARINC653을 구현할 때 발생하는 문제점에 대해서 알아보고 그 해결방안에 대하여 서술한다.

3.1 목적지 프로세스 식별자 매핑

리눅스 시스템에서 RTDiP을 이용하여 ARINC653의 Queuing-mode를 지원하기 위해서 ARINC653의 고유 개념인 포트와 채널의 설정이 필요하다. 일반적인 네트워크 포트는 응용프로그램이 실행할 때 사용자가 정의하는 값으로 설정할 수 있다. RTDiP의 네트워크 포트도 이와 같이 응용프로그램에서 설정이 가능하다. 하지만 ARINC653의 포트설정은 시스템 설정 때 포트이름, 포트식별자, 목적지 주소 등 통신에 필요한 값이 이미 모두 정의되며, 이 포트 내용들은 2장에서 밝힌 바와 같이 응용프로그램의 수행시간에 변경할 수 없다. 이로 인해 RTDiP 포트를 ARINC653에서 그대로 사용할 수 없다. 또한 ARINC653의 통신 응용프로그램은 별도의 네트워크 식별자를 가지고 있지 않다. 오직 포트 식별자를 통해 포트를 확인하고 데이터를 전송한다. 이 문제점을 해결하기 위해 RTDiP의 통신 연결 설정을 수행할 때 (그림 1)과 같이 RTDiP과 Queuing-mode 포트를 각각 따로 생성한 후 서로를 가리킬 수 있는 자료구조를 추가하여 서로를 연결해 주었다. RTDiP 포트에는 통신을 수행하기 위한 네트워크 호출자가 존재한다. 그래서 서로 포트를 연결해주면 Queuing-mode 포트 식별자를 통해 RTDiP의 네트워크 통신 식별자를 바로 접근하여 직접적인 통신에 사용할 수 있다. 또한 RTDiP이 Queuing-mode포트의 포트데이터를 참조할 때 활성화 되어있는 모든 Queuing-mode포트들을 검색하여 자신과 맵핑되어 있는 포트를 찾는 오버헤드를 줄일 수 있다.



(그림 1) 포트 자료구조

3.2 수신 대기 시간 구현

데이터를 수신할 때 수신측에 데이터가 도착하기 전에 사용자 프로세스가 데이터 수신을 요청하면 요청한 데이터가 도착할 때 까지 사용자 프로세스는 잠들게 된다. 이 때 수신 요청한 데이터가 도착하지 않으면 사용자 프로세스는 계속해서 잠들어 있는 문제가 발생한다. 이런 문제점

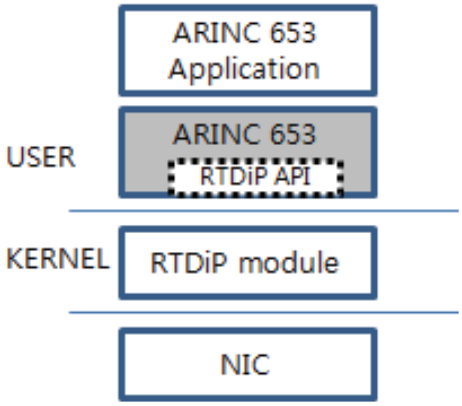
을 해결하기 위해 ARINC653 Queuing-mode의 데이터 수신 인터페이스는 사용자 프로세스가 수신요청을 한 후 수신을 대기하는 시간을 설정할 수 있다. 사용자 프로세스가 수신 요청을 했을 때 데이터가 없는 경우 사용자 프로세스는 잠들게 되고 Queuing-mode에서 설정한 대기시간 동안 데이터를 기다린 후 데이터가 도착하지 않으면 사용자 프로세스는 에러 값을 리턴한 후 깨어나게 된다. 이와 같은 기능은 RTDiP의 시간제한 API[5]를 통해 구현 가능하다. Queuing-mode의 수신포트에서 시간 데이터를 전달받아 RTDiP의 Set_ExpireTime함수를 호출하여 수신대기 시간을 설정하면 된다. 단 Queuing-mode의 시간 데이터는 long long형 64비트이고 RTDiP의 시간 데이터는 32비트이다. 따라서 시간 데이터를 전달할 때 64비트 데이터를 32비트 데이터 형으로 변환하는 과정을 추가하였다.

3.3 네트워크 연결 해제

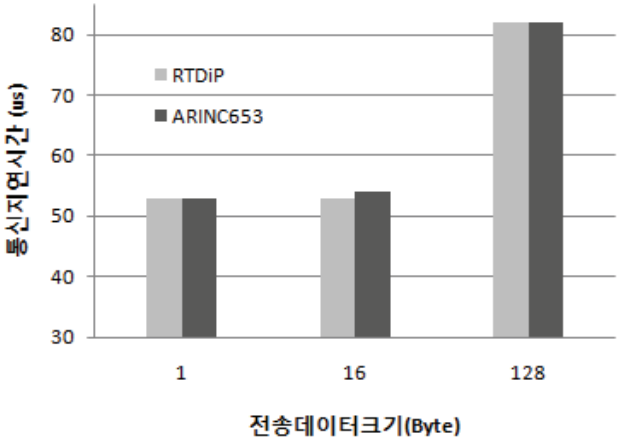
ARINC653에는 네트워크 커넥션을 닫아주는 인터페이스가 존재하지 않는다. RTDiP은 통신을 수행한 후 통신이 종료될 시 네트워크 커넥션을 해제 해 주어야한다. 하지만 ARINC653에 통신 종료 인터페이스가 존재하지 않기 때문에 어느 시점에 네트워크 RTDiP 네트워크 커넥션을 해제할 것인지 결정하고 명시적으로 연결을 해제할 수 있는 방안이 필요하다. 이 문제점을 해결하기 위해 한 가지 제약을 설정 하였다. ARINC653에서 Queuing-mode로 통신을 할 경우에는 통신 응용프로그램이 종료될 때 까지 네트워크 커넥션을 해제하지 않는 것이다. 그리고 프로세스가 종료될 때 호출되는 함수에 통신열결 해제 루틴을 추가하였다. ARINC653의 프로그램 인터페이스 중 프로세스의 수행이 종료될 때 호출되는 STOP_SELP함수가 존재한다. 이 안에 RTDiP의 네트워크 커넥션을 해제하는 함수를 넣어 응용프로그램이 종료될 때 RTDiP의 네트워크 커넥션도 같이 해제될 수 있도록 하였다.

3.4 노드 내 통신 지원

ARINC653은 노드 내 통신, 즉 하나의 시스템 내에서 서로 다른 응용프로그램과 통신을 지원한다. RTDiP의 경우 서로 다른 노드 사이에 통신은 MAC 주소를 이용하여 지원하지만 같은 시스템 안의 프로세스 간 통신은 지원하지 못한다. Queuing-mode가 노드 내 통신을 지원해 주기 위해 RTDiP Kernel 모듈을 확장하였다. 메시지를 전송하기 위해 send명령을 호출하였을 경우, 목적지 주소를 보고 현재 시스템과 같은 MAC 주소를 지녔다면 NIC에 데이터를 전송하지 않고 바로 목적지 포트의 데이터 수신 데이터 큐에 데이터를 넣는다. 그리고 send명령을 종료한다.



(그림 2) ARINC653 구현 구조도



(그림 3) RTDiP과 ARINC653의 통신지연시간

(그림 2)는 RTDiP을 이용해 ARINC653 Queuing-mode를 구현한 구조이다. RTDiP의 커널 모듈은 거의 수정하지 않고 사용자 API를 호출하여 ARINC653 Queuing-mode를 구현하였기 때문에 RTDiP의 여러 가지 장점을 그대로 유지 할 수 있게 하였다.

4. 성능 측정

성능측정을 위해 2개의 동일한 Huins acumen 270 임베디드 보드를 허브나 스위치 없이 연결하여 단방향 지연시간을 측정하였다. 실험을 위해 사용한 운영체제는 Linux kernel 2.6.12 버전이며 이더넷 컨트롤러는 LAN91c111을 사용하였으며 네트워크 디바이스 드라이버는 smc91x를 사용하였다. 실험 군은 다음과 같이 설정하였다. 하나는 순수하게 RTDiP 만을 사용한 단방향 지연시간을 측정하였고 다른 하나는 RTDiP을 사용하여 구현한 ARINC653 Queuing-mode의 단방향 지연시간을 측정하였다.

(그림 3)는 이 두 개의 실험군의 실험측정 결과를 보여준다. 전송 데이터가 1바이트일 경우 RTDiP과 ARINC653은 거의 같은 값을 나타낸다. 16바이트일 경우에는 ARINC653이 약간 더 높은 지연시간을 보인다. 여기서 1

바이트와 16바이트 사이에 단 방향 지연시간이 별로 차이가 없는 이유는 실험에 사용한 smc91x 디바이스 드라이버에서 실재 패킷을 전송할 때 보낼 패킷의 길이가 60바이트 보다 작을 경우 60바이트로 패킷을 전송하기 때문이다. 그래서 1바이트부터 60바이트 까지 단방향 지연시간은 거의 비슷하다. (그림 3)에 나타나듯이 RTDiP과 ARINC653의 통신지연시간이 거의 차이나지 않는다. 이를 통해 RTDiP을 이용하여 ARINC653을 구현할 시 추가적인 오버헤드가 많이 추가되지 않음을 알 수 있다.

5. 결론 및 향후계획

본 논문에서 항공 전자시스템 표준인 ARINC653에 대해 알아보고 경량화 프로토콜인 RTDiP을 이용하여 ARINC653 통신 인터페이스인 Queuing-mode를 구현하였다. 그리고 단방향 지연시간 실험을 통해 추가적인 오버헤드 없이 RTDiP을 이용하여 ARINC653 Queuing-mode이 구현됨을 보였다.

향후 계획으로 ARINC653에서 지원하는 또 다른 통신 방법인 Sampling-mode도 지원 하고 두 개의 통신 모드를 ARINC653 규격의 스케줄러 위에 지원하는 것이다.

참고문헌

[1] Airlines Electronic Engineering Committee, "ARINC SPECIFICATION 653 1-2", December 2005.

[2] Doo-Hyun Kim, Kodirov Nodir, Chun-Hyon Chang, and Jung-Guk Kim , "HELISCOPE Project: Research Goal and Survey on Related Technologies", In Proc. of IEEE ISORC 2009, pp. 112-118 Tokyo, Japan, March 2009.

[3] Sang-Hun Lee and Hyun-Wook Jin, "Communication Primitives for Real-Time Distributed Synchronization over Small Area Networks," In Proc. of IEEE ISORC 2009, Tokyo, Japan, March 2009.

[4] Sang-Hun Lee and Hyun-Wook Jin "Real-Time Communication Support for Embedded Linux over Ethernet", Proceedings of International Conference on Embedded Systems and Applications (ESA 2008), pp. 239-245, Las Vegas, NV, USA, July 2008.

[5] 이상헌, 진현욱, "실시간 분산 동기화 프로토콜을 위한 통신 API 확장," 2009 한국컴퓨터종합학술대회 (KCC 2009) 논문집, 2009년 7월.