센서네트워크 어플리케이션을 위한 네트워크 프레임워크와 통합시뮬레이터 간의 인터페이스 구현 및 설계

이정주* · 곽동은** · 서민석*** · 박현주***

Design and Implementation of Interface Module between Network Framework for Sensor Network Application and Co-Simulator

Lee jeong-joo* · Koak Dong-eun** · Seo Min-Suk*** · Park Hyun-Ju***

이 논문은 2단계 BK21사업과 한국연구재단(과제번호 : 2012-0090)의 지원을 받아 수행된 연구임

요 약

신뢰성 있는 소프트웨어 개발을 위해서 가장 중요한 단계 중의 하나가 소프트웨어 테스트이다. 최근에 점진적이고 반복적인 소프트웨어 개발 방법론이 각광을 받으면서, 소프트웨어의 작은 변경에 따른 회귀 테스트의 중요성이 점점 커지고 있다. 또한 센서네트워크와 같은 다수의 노드 환경에서 동작하는 소프트웨어를 검증하기 위한 시뮬레이터 환경이 필요하다. 본 논문에서는 네트워크 프레임워크와 통합시뮬레이터 간의 인터페이스 모듈을 구현하여, 네트워크 프레임워크로 구현한 센서네트워크 어플리케이션을 다양한 가상의 환경에서 단위테스트하기 위한 환경을 제공한다.

ABSTRACT

For the development of reliable software, Software testing is the most important. Recently small changes of the software according to the importance of regression testing is growing. To verify Application of a large number of nodes, Network simulator environment is required. This paper proposed interface module between network framework for sensor network application and co-simulator to unit test sensor network application. To conclude, developer can focus on sensor network application implementation only, so the improved integrated simulator contributes to increase development productivity.

키워드

TCP/IP, LR-WPAN, IOMux, Network simulator

Key word

TCP/IP, LR-WPAN, IOMux, Network simulator

* 준회원 : 한밭대학교 (doublej@hanbat.ac.kr)
** 준회원 : 한밭대학교
*** 정회원 : 한밭대학교

접수일자 : 2012. 10. 10
심사완료일자 : 2012. 11. 19

Open Access http://dx.doi.org/10.6109/jkiice.2013.17.2.515

Ⅰ. 서 론

신뢰성 있는 소프트웨어 개발을 위해서 가장 중요한 단계 중의 하나가 소프트웨어 테스트이다. 최근에 점진적이고 반복적인 소프트웨어 개발 방법론이 각광을 받으면서, 소프트웨어의 잦은 변경에 따른 회귀 테스트의 중요성이 점점 커지고 있다.

또한 XP(Extreme Programming)과 테스트 주도 개발(Test-Driven Development)가 각광 받으면서 테스트의 중요성이 강조되고 있다.[1]

이러한 측면에서 네트워크 디바이스 계층의 소프트웨어 개발 환경을 기반으로 개발자는 단일 노드에 대한 시스템-온-칩의 검증뿐만 아니라 다수의 노드가 존재하는 유무선 네트워크를 가상으로 구축하여 검증할 수 있는 통합시뮬레이터가 연구되었다.[2] 연구된 통합시뮬레이터를 이용하여 가상의 네트워크 환경을 구축한 후 네트워크 프레임워크로[3] 구현한 센서네트워크 어플리케이션을 이용한 테스트를 진행 할 때 통합 시뮬레이터의 TCP/IP 통신에서 어플리케이션 계층이 데이터를 송/수신하기 위해 소켓을 사용한다. 그러나 통합 시뮬레이터의 LR-WPAN의 경우, TCP/IP 통신의 소켓 역할을 하는 모듈이 존재 하지 않는다.

이에 본 논문에서는 네트워크 프레임워크와 통합 시뮬레이터 간의 센서네트워크 어플리케이션을 위한 통신모듈장치인 인터페이스 모듈을 제안하여, 다양한 환경에서 센서네트워크 어플리케이션을 단위테스트하기 위한 시뮬레이션 환경을 제공한다.

Ⅱ. 관련 연구

2.1. 통합 시뮬레이터

SystemC와 NS-3[4]에서 시스템/네트워크 측면의 운영체제 기능의 제공 가능한 기능을 이용하였다. SystemC는 프로세스 및 IPC와 같은 시스템 측면의 기능은 제공할 수 있지만 통신 프로토콜은 제공하기 어렵다. 반면, NS-3는 실제 운영체제와 유사한 소켓 인터페이스를 제공하지만, 시뮬레이터의 근본적인 한계로 인해 동시성 실행 모델을 제공할 수 없다.

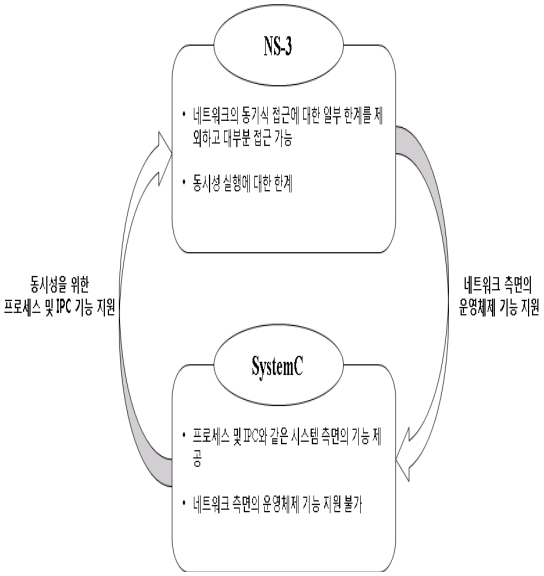

그림 1. 시뮬레이터 간의 상호 보완성
Fig. 1 Simulator complementarity between

그림 1과 같이 NS-3와 SystemC를 통합하면 시스템/네트워크 측면의 운영체제 기능을 제공하면서 네트워크의 동기식 접근 방식을 지원할 수 있다. 더 나아가 입출력 다중화 기능을 SystemC에서 제공하는 기능을 이용하여 보완하면 네트워크 프레임워크[5]를 위한 가상의 환경을 도출할 수 있다.

위에서 설명한 통합 시뮬레이터[2]는 NS-3에서 지원하는 통신 모델에 의존 한다. 현재 NS-3는 TCP/IP를 기반으로 하는 통신 모델을 주로 지원하고 WSN(Wireless Sensor Network)을 위한 LR-WPAN(LowRate-Wireless Personal Area Networks) 프로토콜은 NS-2에서 NS-3로 변환 작업을 진행 중이다.

이에 본 논문에서는, 네트워크 프레임워크와 통합 시뮬레이터 간의 인터페이스 모듈을 제안한다.

Ⅲ. 본 론

본 장에서는 네트워크 프레임워크와 통합 시뮬레이터 간의 인터페이스 모듈을 제안한다.

3.1. LrWpanMsgBroker의 설계 및 구현

본 절에서는 LrWpanMsgBroker를 설명한다. LrWpan MsgBroker는 네트워크 프레임워크가 NS-3을 통해 LR-WPAN 패킷을 송수신하기 위한 목적으로 사용한다. [그림 2]는 LR-WPAN을 위한 통합 시뮬레이터와 네트워크 프레임워크 간의 구조를 나타낸다.

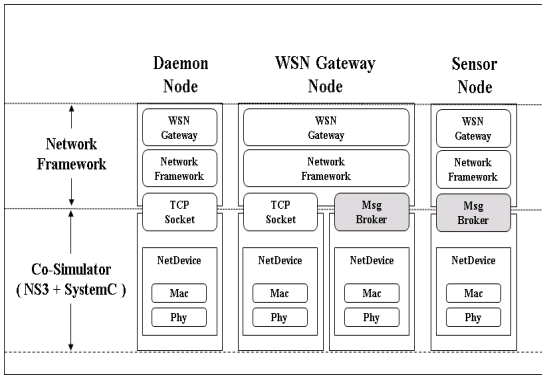

그림 2. 센서 네트워크를 위한 통합 시뮬레이터와 네트워크 프레임워크 간의 구조
Fig. 2 Integrated simulator for sensor networks and between network framework structure

NS-3는 TCP/IP 통신을 위한 TCP Socket 모듈을 제공한다. 하지만 NS-3의 LR-WPAN은 어플리케이션과 네트워크 디바이스를 연결하는 기능을 제공하지 않는다.

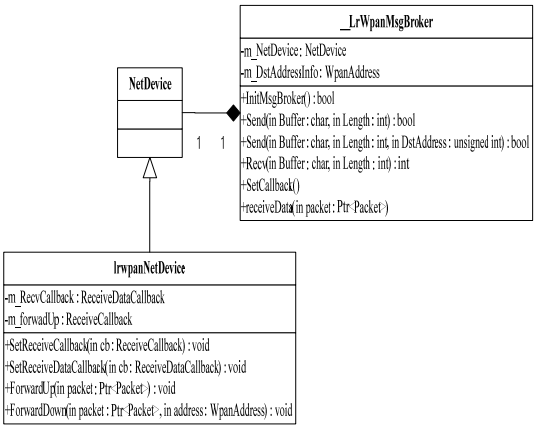

그림 3. LrWpanMsgBroker의 클래스 다이어그램
Fig. 3 Class diagram of LrWpanMsgBroker

이에 본 논문에서는 네트워크 프레임워크의 어플리케이션과 LR-WPAN의 연결을 위한 LrWpanMsgBroker를 제안한다.

[그림 3]은 LrWpanMsgBroker의 클래스 다이어그램을 나타낸다. MsgBroker는 NS-3의 LrWpanNetDevice를 소유한다. 네트워크 프레임워크의 MsgBroker는 이를 이용하여 LR-WPAN에 패킷을 송/수신 한다.

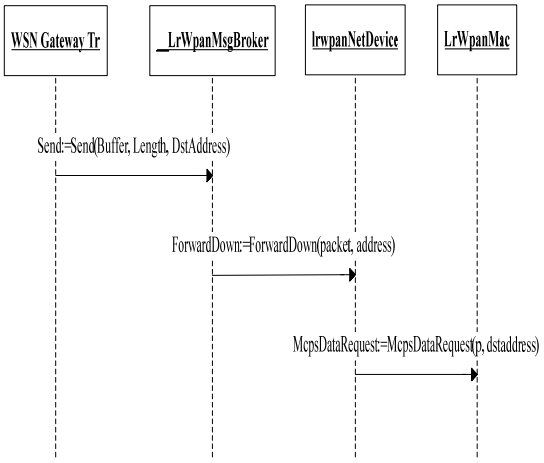

그림 4. LR-WPAN 패킷 송신을 위한 시퀀스 다이어그램
Fig. 4 Sequence diagram for the LR-WPAN packet transmission

[그림 4]는 네트워크 프레임워크가 LR-WPAN에 패킷을 송신하는 흐름을 나타낸다. 네트워크 프레임워크 어플리케이션은 LrWpanMsgBroker를 통해 LR-WPAN의 NetDevice를 제어 할 수 있으며, NetDevice는 NS-3의 LR-WPAN 프로토콜을 수행한다.

[그림 5]는 네트워크 프레임워크가 LR-WPAN의 패킷을 수신하는 흐름을 나타낸다. [그림 5]는 아래와 같은 동작 흐름을 가진다.

① 네트워크 프레임워크에서, LR-WPAN의 패킷 수신 이벤트를 감지하기 위한 콜백 함수를 등록한다. [InitMsgBroker() 수행]
② NS-3에서 LR-WPAN 패킷을 송신하면, LrWpanMac에서 패킷을 수신한다. 수신된 패킷을 lrwpanNet Device에게 포워딩 해준다.

③ lrwpanNetDevice는 미리 등록된 콜백 함수[receive Data(packet)]를 수행하여 네트워크 프레임워크에게 패킷 정보를 전달한다. 전달 받은 패킷 정보는 LrWpanMsgBroker가 리스트 형태로 관리한다.

④ 네트워크 프레임워크의 어플리케이션에서 Recv() 함수를 호출 하면 LrWpanMsgBroker는 리스트 형태로 관리하는 패킷데이터를 어플리케이션에게 전달한다.

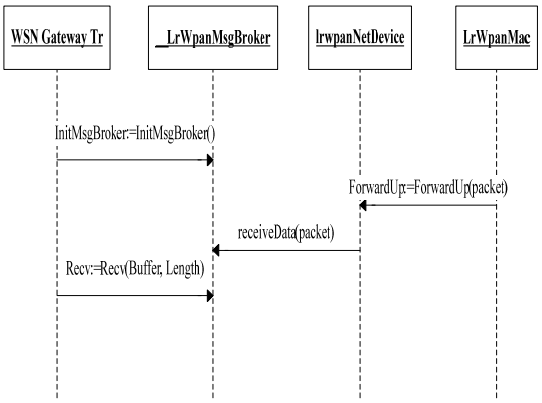

그림 5. LR-WPAN 패킷 수신을 위한 시퀀스 다이어그램
Fig. 5 Sequence diagram for the LR-WPAN packets received

'본 논문에서 제안하는 LrWpanMsgBroker의 패킷 수신을 위해서는 Recv 동작을 폴링(Polling) 방식으로 수행해야 하는 문제가 있다. 이런 문제를 해결하기 위해, 본 논문에서는 LrWpanMsgBroker의 이벤트를 감지하기 위한, 입출력 다중화 모듈(LrIOMux)을 제안한다.

3.2. LrIOMux의 설계 및 구현

본 절에서는 LrWpanMsgBroker의 수신 이벤트를 감지하기 위한 입출력 다중화 모듈(LrIOMux)을 설명한다.

[그림 6]은 본 논문에서 제안하는 LrIOMux의 클래스 다이어그램이다. LrIOMux의 주요 기능은 이벤트 등록과 이벤트 검출이다.

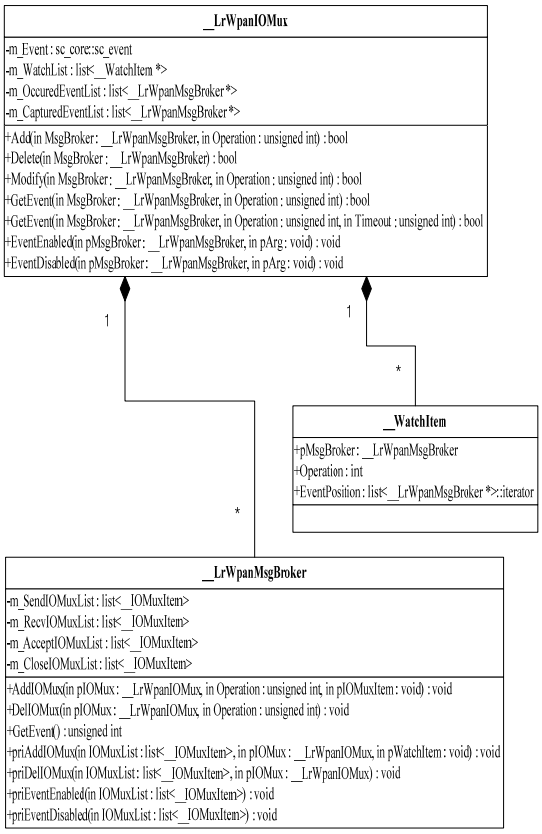

그림 6. LrIOMux의 클래스 다이어그램
Fig. 6 LrIOMux class diagram

LrIOMux는 Add 함수를 이용하여 LrWpanMsgBroker의 이벤트를 등록한다. 그리고 GetEvent 함수를 이용하여, 등록된 LrWpanMsgBroker의 수신 이벤트가 발생할 때까지 프로세스를 대기한다.

[그림 7]는 LrIOMux에 LrWpanMsgBroker의 수신 이벤트를 등록하는 흐름을 나타낸다. 어플리케이션에서 LrWpanIOMux에 이벤트를 등록하면, LrWpanIOMux는 LrWpanMsgBroker에 WatchItem을 등록한다. WatchItem은 LrWpanMsgBroker와 이벤트 종류, 그리고 이벤트를 검출하고 싶은 LrWpanMsgBroker의 리스트를 포함하는 자료 형이다.

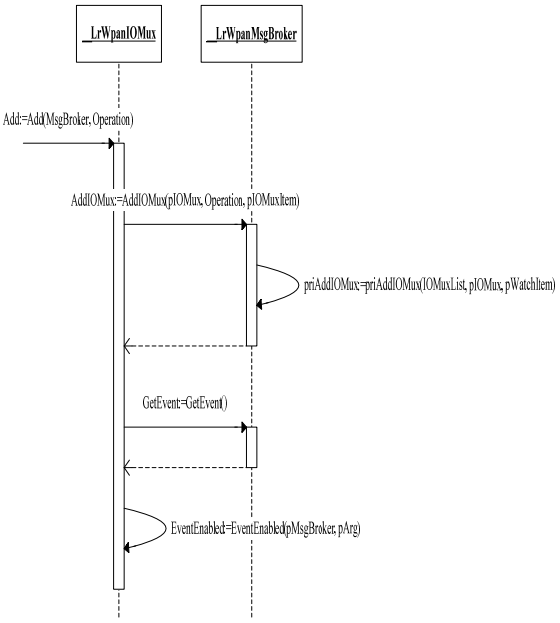

그림 7. 이벤트 등록을 위한 동작 흐름
Fig. 7 Flow behavior for event registration

[그림 8]은 LrWpanIOMux에 이벤트를 등록하기 위한 Add 함수의 구현을 나타낸다. Add 함수는 등록하려고 하는 MsgBroker가 WatchList에 이미 등록 되어 있는지 검사 한다. 이벤트를 검출하고 싶은 MsgBroker 객체가 LrWpanIOMux의 WatchList에 존재 하지 않을 경우, 새로운 WatchItem 객체를 생성하여 리스트로 보관하고 MsgBroker 객체에게 이벤트 검출을 위한 정보(WatchItem)을 전달한다. 마지막으로, LrWpanIOMux의 이벤트를 초기화함으로써 Add 함수의 동작을 마친다.

[그림 9]는 LrIOMux의 GetEvent 동작 흐름을 나타낸다. 어플리케이션에서 LrWpanIOMux의 GetEvent를 실행하면 LrWpanIOMux는 등록된 LrWpanMsgbroker의 이벤트가 발생할 때까지 프로세스를 대기한다. NS-3에서 LR-WPAN의 패킷 이벤트가 발생하면, lrwpanNet Device는 콜백 함수(receiveData)를 호출하여 LrWpan MsgBroker에게 패킷을 전달하고 LrWpanMsgBroker는 LrWpanIOMux에게 이벤트를 통보한다.

```
bool
__LrWpanIOMux::Add(__LrWpanMsgBroker * pMsgBroker, unsigned int Operation)
{
  list<__WatchItem *>::iterator    Iter;
  __WatchItem *                    pWatchItem;

  for( Iter = m_WatchList.begin(); Iter != m_WatchList.end(); Iter++ )
  {
     pWatchItem = *Iter;
     if( pWatchItem->pMsgBroker == pMsgBroker)
     {
        return false;
     }
  }

  /* Create/Add WatchItem */
  pWatchItem = new __WatchItem;

  pWatchItem->pMsgBroker        = pMsgBroker;
  pWatchItem->Operation         = Operation;
  pWatchItem->EventPosition = m_OccuredEventList.end();

  m_WatchList.push_back( pWatchItem );


  /* Add IOMux into Socket */
  pMsgBroker->AddIOMux( this, Operation, (void *)pWatchItem );


  /* Initialize Events */
  if( pMsgBroker->GetEvent() & Operation )
  {
     EventEnabled( pMsgBroker, pWatchItem );
  }
}
```

그림 8. LrWpanIOMux에 이벤트 등록을 위한 Add 함수의 구현
Fig. 8 Implementation of the add function for Event registration to LrWpanIOMux

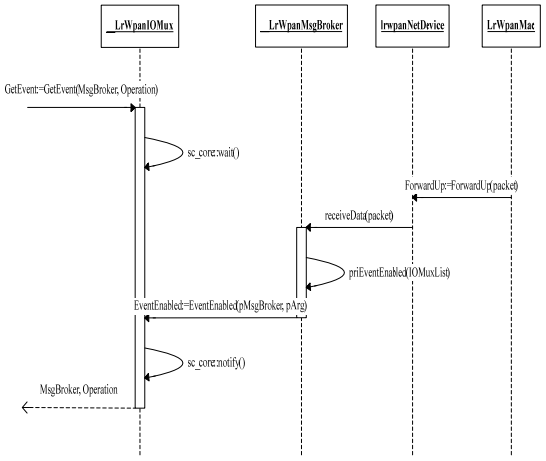

그림 9. LrIOMux의 GetEvent 동작 흐름
Fig. 9 GetEvent Operation flow of LrIOMux

LrWpanIOMux가 이벤트를 감지하면 GetEvent로 인해 대기하고 있는 프로세스를 깨우고 어플리케이션에게 MsgBroker 객체와 이벤트 종류를 반환한다.

[그림 10]은 LrIOMux의 GetEvent 함수의 구현을 나타낸다. GetEvent 함수는 LrWpanIOMux가 관리하는 이벤트가 존재 하지 않으면, sc_core::wait() 함수를 사용하여 프로세스를 대기 상태로 만든다.

그런 후, NS-3에서 LR-WPAN의 패킷 수신 이벤트가 발생하면 LrWpan-MsgBroker는 priEventEnabled 함수를 호출하여 LrWpanIOMux의 EventEnabled 함수를 수행한다. LrWpanIOMux의 EventEnabled 함수는 GetEvent 함수에 의해 대기 상태에 있는 프로세스를 깨우고, 해당 어플리케이션에게 LrWpanMsgBroker 객체와 이벤트의 종류를 반환한다.

[그림 11]은 LrWpanMsgBroker의 priEventEnabled 함수의 구현을 나타내고, [그림 12]는 LrWpanIOMux의 EventEnabled 함수의 구현을 나타낸다.

```
bool
_LrWpanIOMux::GetEvent(__LrWpanMsgBroker **pMsgBroker, unsigned int *pOperation, unsigned int Timeout)
{
   Timeout = MAX_INT_VALUE & Timeout;

   while( m_CapturedEventList.empty() == true )
   {
      if( m_OccuredEventList.empty() == true )
      {
         sc_core::wait( (int)Timeout, SC_SEC,s m_Event );

         if( timed_out() == true )
         {
            *pOperation = __LrWpanIOMux::TIMEOUT;
            return false;
         }
      }

      m_CapturedEventList = m_OccuredEventList;
   }

   *pMsgBroker = m_CapturedEventList.front();
   *pOperation = (*pMsgBroker)->GetEvent();

   m_CapturedEventList.pop_front();

   return true;
}
```

그림 10. LrIOMux에서 GetEvent 함수 구현
Fig. 10 GetEvent function implementation from LrIOMux

```
void
__LrWpanMsgBroker::priEventEnabled( list<__IOMuxItem> & IOMuxList )
{
   list<__IOMuxItem>::iterator      Iter;

   for( Iter = IOMuxList.begin(); Iter != IOMuxList.end(); Iter++ )
   {
      Iter->pIOMux->EventEnabled( this, Iter->pArg );
   }
}
```

그림 11. LrIOMux에 priEventEnabled 함수의 구현
Fig. 11 priEventEnabled function implementation from LrIOMux

```
void
__LrWpanIOMux::EventEnabled(__LrWpanMsgBroker * pMsgBroker, void * pArg)
{
   __WatchItem * pWatchItem = (__WatchItem *) pArg;

   if( pWatchItem->EventPosition != m_OccuredEventList.end() )
   {
      return;
   }

   m_OccuredEventList.push_back( pWatchItem->pMsgBroker );
   pWatchItem->EventPosition--;

   sc_core::notify( m_Event );
}
```

그림 12. GetEvent 함수에 의해 대기 중인 프로세스를 깨우기 위한 EventEnabled 함수의 구현
Fig. 12 Function for EventEnabled by GetEvent function to wake up the waiting process of implementation

Ⅳ. 실 험

본 장에서는, 제안하는 통합 시뮬레이터의 효용성을 검증한다. 다양한 가상 네트워크 환경을 통합 시뮬레이터 기반에서 구축한 후, 개발자가 구현한 네트워크 프레임워크 기반의 소프트웨어를 구동한다. 이에 소프트웨어가 구동하면서 송/수신 하는 패킷의 흐름을 패킷 분석 도구를 통하여 보인다.

4.1. LrWpanIOMux 검증을 위한 단위 테스트

본 절에서는 개발자가 구현한 네트워크 어플리케이션을 통합 시뮬레이터 기반위에서 검증하는 예제를 보인다. [그림 13]은 검증을 위한 예제구조를 나타낸다. 이 예제에서는, 이더넷 채널에 3개의 노드가 있고 LR-WPAN 채널에 2개의 노드가 있다. 그리고 각 노드에는 네트워크 프레임워크 기반에서 개발자가 구현한 어플리케이션을 가정한다. 노드의 이름은 PDA 노드, Daemon 노드, WSN Gateway 노드, Sensor Device 노드로 부여한다.

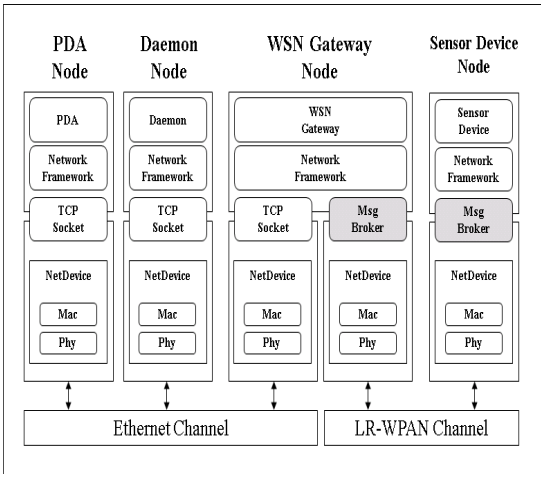

그림 13. 검증을 위한 네트워크 토폴로지 구조
Fig. 13 The network topology structure for verification

[그림 14]는 검증 시나리오이고, 이 시나리오에 대한 설명은 다음과 같다.

PDA 노드는, Daemon 노드에게 데이터를 송신한다. (이더넷 채널 사용) Daemon 노드는 PDA 노드로부터 받은 데이터를 가공한 후, 이 정보를 데이터베이스에 저장하고 WSN Gateway 노드에게 데이터를 송신한다. (이더넷 채널 사용) WSN Gateway 노드는 Daemon 노드로부터 받은 데이터를 가공한 후, 이 정보를 SensorDevice 노드에게 데이터를 송신한다. (이더넷 채널, LR-WPAN 채널 사용)

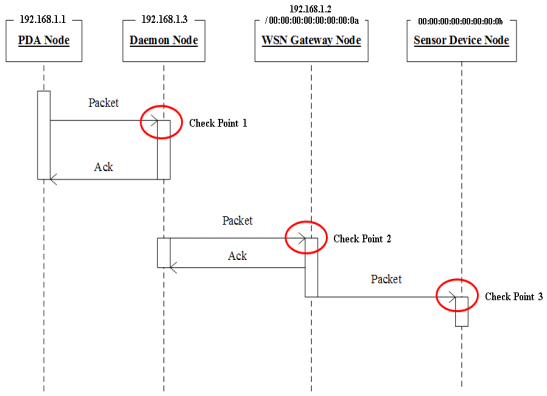

그림 14. 검증을 위한 시나리오
Fig. 14 Scenarios for verification

이러한 동작 시나리오에서, [그림 14]에서 제시한 3개의 시점에서 수신한 패킷을 검증한다.

```
/* Step 1 */
NodeContainer  nodes;
nodes.Create (3);

CsmaHelper  Csma;
Csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
Csma.SetChannelAttribute ("Delay", StringValue ("1ms"));

NetDeviceContainer devices;
devices = Csma.Install (nodes);

/* Step 2 */
InternetStackHelper  stack;
stack.Install (nodes);

Ipv4AddressHelper address;
address.SetBase ("192.168.1.0",  "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign (devices);


/* Step 3*/
NodeContainer lrNodes;
lrNodes.Create(2);

LrWpanHelper lrWpanHelper;
LrWpanPhyHelper  phy;
LrWpanMacHelper  mac;

NetDeviceContainer lrDevices;
lrDevices = lrWpanHelper.Install( phy, mac, lrNodes );
```

그림 15. 네트워크 토폴로지를 구성하는 코드
Fig. 15 Code to configure the network topology

[그림 15]부터 [그림 16]은 위의 시나리오에 대한 예제 코드를 나타낸다. [그림 15]는 네트워크 환경 구성에 대한 코드를 나타낸다. 각 단계 별 설명은 다음과 같다.

1단계 : 3개의 노드를 생성한 후, 각 노드의 네트워크 디바이스는 이더넷 디바이스로 설치한다. 그리고 CSMA(Carrier Sense Multiple Access) 채널을 생성한 후, 이더넷 디바이스를 장착한 3개의 노드를 이 채널에 연결한다. 따라서 이러한 코드는 이더넷 디바이스를 장착한 3개의 노드가 하나의 공유 채널에 연결한 형태를 구성한 것이다.

2단계 : 위의 1단계에서 생성한 각 노드에 인터넷 프로토콜을 설치한다. NS-3의 라이브러리에는, 네트워크 계층 및 트랜스포트 계층에 대한 다양한 라이브러리를 지원한다. 본 예제서는 이러한 프로토콜 중에서 TCP/IP를 지원하는 인터넷 프로토콜의 설치 예제를 보인다.

3단계 : 2개의 노드를 생성한 후, 각 노드의 네트워크 디바이스는 LR-WPAN 디바이스로 설치한다.

```
/* Applications : Daemon */
__NwkFrwHelper<__TraceDaemonFactory>  DaemonHelper;
DaemonHelper.GetFactory().SetDbUrl("127.0.0.1");

ApplicationContainer DaemonApps = DaemonHelper.Install (nodes.Get (2));
DaemonApps.Start (Seconds (1.0));
DaemonApps.Stop (Seconds (60.0));


/* Applications : WSN Gateway */
__NwkFrwHelper<__TraceGwayFactory>  GwayHelper;
GwayHelper.GetFactory().SetDaemonIp( "192.168.1.3" );
GwayHelper.GetFactory().SetDaemonPort( 30303 );
GwayHelper.GetFactory().SetGwayId( "01" );
GwayHelper.GetFactory().SetNetDevice( lrDevices.Get(0) );

ApplicationContainer GwayApps = GwayHelper.Install (nodes.Get (1));
GwayApps.Start (Seconds (2.0));
GwayApps.Stop (Seconds (60.0));

/* Applications : Pda */
__NwkFrwHelper<__TracePdaFactory>  PdaHelper;
PdaHelper.GetFactory().SetDaemonIp( "192.168.1.3" );
PdaHelper.GetFactory().SetDaemonPort( 30303 );
PdaHelper.GetFactory().SetId( "PDA01" );
PdaHelper.GetFactory().SetPassword( "semco" );

ApplicationContainer PdaApps = PdaHelper.Install (nodes.Get (0));
PdaApps.Start (Seconds (3.0));
PdaApps.Stop (Seconds (60.0));

/* Applications : Sensor Device */
__NwkFrwHelper<__LrWpanEndDeviceFactory>  EndDeviceHelper;
EndDeviceHelper.GetFactory().SetNetDevice( lrDevices.Get(1) );

ApplicationContainer EndDeviceApp =
EndDeviceHelper.Install( lrNodes.Get(1) );
EndDeviceApp.Start ( Seconds (4.0) );
EndDeviceApp.Stop  ( Seconds (60.0) );
```

그림 16. 어플리케이션들을 설치하는 코드
Fig. 16 Code that installs the application

[그림 16]은 각 노드에 어플리케이션을 설치하는 코드를 나타낸다. 각 어플리케이션은 네트워크 프레임워크를 기반으로 구현한 소프트웨어이다. 각 어플리케이션은 인스턴스를 생성하기 위한 자신만의 팩토리 오브젝트가 있다. 이러한 팩토리 오브젝트에게 어플리케이션 생성을 위한 파라미터를 전달한다. PDA 노드와 WSN Gateway 노드는, Daemon 노드에게 접속해야 하므로 Daemon 노드에 대한 주소 및 포트 번호를 팩토리 오브젝트에게 전달한다. Daemon 노드는 소켓 접속을 받는 서버이므로 접속을 허락할 포트 번호를 설정한다. WSN Gateway와 Sensor Device의 LR-WPAN 프로토콜 지원을 위한 네트워크 디바이스를 설정한다. WireShark를 이용하여 LrWpanIOMux 검증을 위해서 CSMA 노드들의 패킷 흐름을 분석한 후, [그림 14]의 Check1 Point와 Check2 Point로 필터링하면 [그림 17]과 같은 디바이스의 패킷 흐름 볼 수 있다. 이것은 TCP/IP 프로토콜을 이용한 노드들에 대한 패킷 수신에 대한 검증을 하는 부분으로 패킷 수신의 확인을 통해서 검증 할 수 있었다. 그리고 [그림 14]의 Check Point 3을 WireShark로 분석하면 [그림 18]과 같은 디바이스의 패킷 흐름 볼 수 있다. 이것은 LR-WAN 프로토콜을 이용하는 노드에 대한 패킷 수신을 검증한 것이다. 패킷의 수신된 내용을 통해서 검증 할 수 있다.

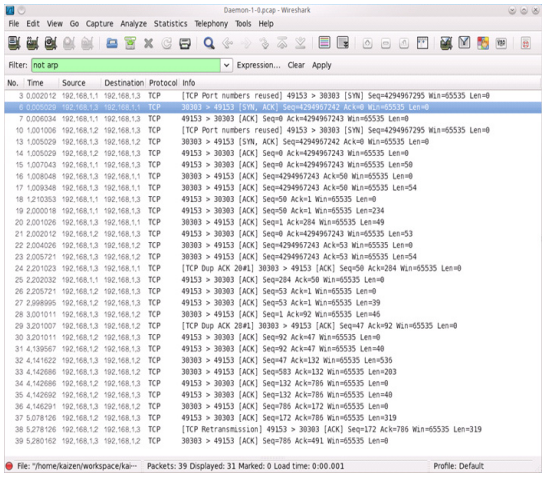

그림 17. CSMA 디바이스의 패킷 흐름 분석 화면
Fig. 17 CSMA device screen, the packet flow analysis

결론적으로, 본장에서 제안한 통합시뮬레이터는, LR-WPAN 프로토콜과 TCP/IP 프로토콜을 사용하는 다수의 노드에 대한 검증 작업을 자동화 할 수 있을 뿐만 아니라 실행 흐름에 대한 분석 작업에 있어 SystemC와 NS-3가 제공하는 기능을 모두 활용 할 수 있다.

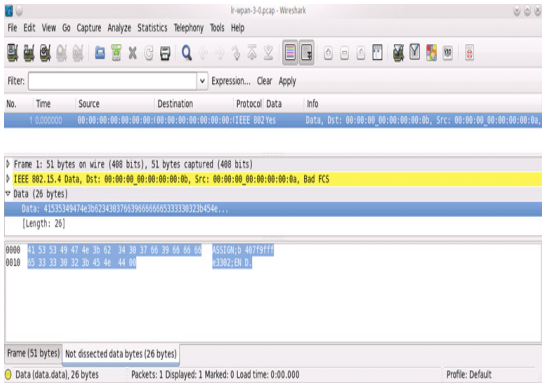

그림 18. LR-WPAN 디바이스의 패킷 수신 분석 화면
Fig. 18 Receive packets for analysis of LR-WPAN devices screen

V. 결 론

본 논문에서는 네트워크 프레임워크를 이용하여 구현한 센서네트워크 어플리케이션을 가상의 환경에서 시뮬레이션 할 수 있도록 네트워크 프레임워크와 통합 시뮬레이터 간의 인터페이스 모듈을 제안하였다.

본 논문에서 제안한 인터페이스 모듈을 이용하면, 소프트웨어 구현 시 네트워크프레임워크를 이용하여 개발 생산성을 높일 수 있고 이를 가상의 환경에서 시뮬레이션 할 수 있는 환경을 제공할 수 있다.

특히 센서네트워크 어플리케이션의 경우, 네트워크를 구성하는 노드의 개수가 많아 실제 테스트 환경을 구성하는데 어려움이 많다. 하지만 본 논문에서 제안한 인터페이스 모듈을 이용하면 이와 같은 어려움을 해결할 수 있다. 또한 NS-3와 SystemC를 결합함으로써, NS-3와 SystemC의 Trace 기능을 사용하여 소프트웨어를 더욱 세부적으로 검증 할 수 있으며, SystemC의 기능을 통해 네트워크 디바이스의 하드웨어적인 검증 환경을 제공할 수 있다.

본 논문의 향후 연구 과제로는 NS-3에서 지원하는 미완성 LR-WPAN 프로토콜 모델의 보완이 필요하다. 현재 NS-3의 미완성 LR-WPAN의 Mac 계층은 데이터를 송/수신 하는 기능만 지원한다. 이에 BEACON-NOTIFY, ASSOCIATE, DISASSOCIATE 등 LR-WPAN의 네트워크를 구성 및 관리하기 위한 기능을 추가하는 연구가 필요하다.

참고문헌

[1] R. Hightower & N. Lesiecki, “Java Tools for Extreme Programming”, 2002, Jone Wiley & Sons.

[2] Hoeung Lee, “A proposal of development and verification environment of network applications based on a Co-Simulator”, IEICE vol.8

[3] 이호응, “무선 센서 네트워크 환경에서 게이트웨이 어플리케이션의 개발 환경을 위한 패킷 처리 관점의 네트워크 프레임워크 설계 및 구현”, 한국해양정보통신학회, 2011, 4,

[4] NS-3, http://www.nsnam.org/

[5] Stephen D. Huston James CE Johnson Umar Syyid, “ACE Programmer’s Guide, The:Practical Design Patterns for Network and Systems Programming, 1st Edition, Pearson Education, Inc.

저자소개

이정주(Jeongjoo Lee)

2011년 한밭대학교 전파공학과 졸업(학사)
2011～ 현재 한밭대학교 정보통신전문대학원 전파공학과 재학(석사과정)

※관심분야 : 무선 통신 소프트웨어, 실내 위치기반 서비스, etc.

곽동은(Dongeun Koak)

2010년 한밭대학교 정보통신 공학과 졸업(학사)
2012년 한밭대학교 정보통신전문 대학원 전파공학과 석사

※관심분야 : 무선 통신 소프트웨어, 임베디드 소프트웨어

서민석(Minsuk seo)

2010년 한밭대학교 정보통신 공학과 졸업 (학사)
2012년 한밭대학교 정보통신전문 대학원 전파공학과 졸업 (석사과정)

※관심분야 : 무선 통신 소프트웨어, 임베디드 소프트웨어

박현주(Hyunju Park)

1990년 서울 시립대학교 전산통계학과 학사
1992년 서울대학교 대학원 전산과학과 석사
1997년 서울대학교 대학원 전산과학과 박사
1997년~현재 한밭대학교 전파공학과 교수

※관심분야 : 데이터베이스, 무선 통신 소프트웨어