KIPS Tr. Comp. and Comm. Sys.
Vol.3, No.10 pp.365~370 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2014.3.10.365

A Method of Test Coverage Measurement Based on BitTorrent for Internet of Things Environment

Hodong Ryu† · Woo Jin Lee††

ABSTRACT

Although Internet of Things already became a new paradigm on service on network, we should pay more effort for studying about its testing method, since humans, things and environments in IoT are connected to each other without any restrictions. Earlier researches based on emulators showed that such virtual devices on emulators had unavoidable gap between them and real things. Furthermore, growth of connection complexity between the devices and loosing of restrictions make the gap wider. Accordingly, in this paper, we suppose a method of test coverage measurement based on BitTorrent for IoT environment. It has cooperation features among homogeneous devices with avoiding the overlapping on each part of whole test process.

Keywords : Internet of Things, Distributed Environment, BitTorrent, Test Coverage Measurement

사물 인터넷 환경을 위한 BitTorrent 알고리즘 기반의 테스트 커버리지 측정기법

류 호 동† · 이 우 진††

요 약

사물 인터넷(Internet of Things : IoT) 환경이 이미 네트워크기반의 서비스 분야에서 가장 대표적인 패러다임이 되었음에도 불구하고, 인간과 사물 및 환경이 서로 제약 없이 연결되는 특성으로 인하여 해당 환경에 특화된 테스트 기법은 여전히 많은 연구가 필요한 상황이다. 에뮬레이터 기반에서 테스트 대상 장치를 구동하는 대부분의 기존 방식은 IoT와 같이 그 연결이 다양해지고 장치의 구분이 불분명해질수록 실제 환경과의 격차가 커져 결론적으로 부정확한 테스트 결과가 산출될 가능성이 높다. 본 논문에서는 이러한 문제를 개선하고자 각각의 장치들의 오버헤드를 최소화함과 동시에 대상 코드의 특정 부분이 중복적으로 테스트되지 않는 특징을 가진 BitTorrent기반의 테스트 커버리지 측정 기법을 제안한다.

키워드 : Internet of Things, 분산환경, BitTorrent, 테스트 커버리지 측정

1. 서 론

인간과 사물 환경이 네트워크에 연결되어 장치 스스로가 사용자의 명시적인 개입과 시간과 공간적 제약없이 서비스를 제공하는 Internet of Things(IoT) 서비스는 다음 세대의 새로운 패러다임이 되고 있다[1]. 기존의 분산 환경과는 달리 외부의 명시적 개입이 없는 상태에서 장치 각각이 자율적으로 연결되어 서로의 정보를 주고받는 IoT의 대표적인 특성은 다양한 상호작용을 발생시키고 이는 테스트 커버리지 측정을 어렵게 한다. 또한 이는 특정 오류가 발생하는 상황의 추적 또한 어렵게 만드는 원인이 된다. 따라서 이러한 문제해결을 위해 IoT환경에서 테스트 수행 시나리오를 파악하고 효과적으로 커버리지를 측정할 수 있는 새로운 테스팅 기법이 필요성이 대두되고 있다[2].

일반적으로 분산 환경에서의 대부분의 장치들은 임베디드 수준의 많은 하드웨어적 제약을 가지고 있고, 이는 테스트를 위한 추가적인 코드를 수행하는데 걸림돌이 되어 왔다. 이를 해결하기 위해 에뮬레이터를 이용하여 가상의 장치를 구현하고 이를 수행함으로써 테스트를 진행하는 방법이 대

※ 본 연구는 미래창조과학부 및 정보통신산업진흥원의 ICT융합 고급인력과정 지원사업과 (NIPA-2014-H0401-14-1004) 미래창조과학부 및 정보통신기술연구진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [10041145, 자율군집을 지원하는 웰빙형 정보기기 내장 소프트웨어 플랫폼 개발]
† 준 회 원 : 경북대학교 컴퓨터학부 박사과정
†† 정 회 원 : 경북대학교 컴퓨터학부 정교수
Manuscript Received : September 5, 2014
Accepted : October 16, 2014
* Corresponding Author : Woo Jin Lee(woojin@knu.ac.kr)

안으로 활용되어 왔다. 에뮬레이터를 기반으로 하는 기존의 연구에서는 가상의 하드웨어 에뮬레이터를 이용하여 대상 장치의 코드를 수행하고, 여러 개의 에뮬레이터를 두어 대상 장치의 코드를 그 위에서 수행함으로써 테스트를 위한 가상의 분산 환경을 제공한다[3-6]. 하지만, 이러한 방법에서는 가상 환경과 실제 환경의 차이로 인해 문제점이 발생한다. 특히 IoT 환경과 같이 각각의 장치들이 외부의 명시적인 제어 없이 자유롭게 연결되는 상황은 에뮬레이터의 요구사항을 더욱 복잡하게 하는 원인이 된다.

이에 본 논문에서는 실제 하드웨어를 대상으로 하되, 각 하드웨어가 테스트를 수행할 때 소모하는 자원 소모를 최소화하는 방법으로 BitTorrent[8]기반의 테스트 기법을 제안한다. BitTorrent 기법은 하나의 파일을 완성하기 위해 각각의 Peer들이 자신이 가지고 있는 조각들을 서로 교환하는 방식으로 운영되며, 없는 조각은 요청하고 이미 있는 조각은 없는 Peer들에게 전송함으로써 최종적으로 모든 Peer들이 완성된 파일을 가질 수 있게 하는 대표적인 P2P 프로토콜이다. 본 논문에서는 이를 테스트 결과 교환에 사용하여 테스트의 대상이 되는 코드를 테이블 형태로 추상화하고 이를 완성해야 할 대상 파일로 정의하여 각각의 장치들이 테이블의 형태로 구분된 전체 테스트의 일부분을 수행하고 수행된 결과를 서로 공유함으로써 결과적으로 각 장치들이 전체 테스트를 나누어서 수행하는 효과와 동시에 중복된 테스트 수행의 오버헤드를 감소시키는 결과를 얻도록 한다.

공유의 대상이 되는 테이블은 해당 장치에 대한 테스트 시나리오 전체를 의미하고, 각각의 장치들은 본래의 서비스를 제공하는 과정에서 현재 실행 중인 특정 부분에 진입함과 동시에 이미 테스트된 적이 있는지를 자신이 가진 테이블에서 확인한다. 이어 이미 테스트 된함수이면 별다른 추가 행위 없이 그대로 진행하고, 아직 테스트되지 않은 함수라면 코드 분석 과정에서 삽입된 테스트 관련 코드가 실행되어 해당 부분의 실행에 관련된 정보와 테스트 결과를 자신의 테이블에 기록한 후 동종의 다른 장치에 그 데이터를 알린다. 이런 과정이 반복됨으로써 이미 테스트한 결과를 전송받는 과정을 반복한다.

본 기법은 에뮬레이터 환경과 달리 실제와 하드웨어 기반의 환경에서 동종의 테스트 대상 장치들이 테스트를 분업하고 중복적으로 수행되는 테스트를 피하는 과정을 통해 테스트에 소모되는 자원을 최소화하는 특징을 가진다.

본 논문의 구성은 다음과 같다. 2장에서는 분산 환경의 테스팅과 관련된 기존의 연구와 본 논문에서 사용하는 BitTorrent의 특징에 대해 살펴보고, 3장에서는 테스트 대상 장치의 코드를 테이블형태로 추상화시키는 방법과 실제 테스트 데이터를 주고받을 테스트 수행부분, 제시된 방법을 이용한 테스트 과정에 대하여 설명한다. 4장에서는 앞에서 제시한 방법을 더욱 구체화한다. 5장에서는 본 논문에서 제시하는 아이디어의 효용성을 검토하고 제시하는 방법에 대한 평가로 맺는다.

2. 관련 연구

분산 환경은 네트워크 환경과 하드웨어 성능, 특히 임베디드 장치들의 성능 향상에 힘입어 비약적인 발전을 해왔고, 이와 더불어 이러한 환경을 테스트하기 위한 기법의 연구도 함께 이루어져왔다. 현재 IoT 환경에 특화된 테스트에 대한 연구는 시작단계에 있으며, 유비쿼터스 환경에서 에뮬레이터에 기반으로 다수의 장치를 대상으로 가상의 환경을 극복하는 방법으로 진행되어왔다[5,6,10]. 이러한 테스트 방식은 실제상황에서 통제하기 어려운 환경상의 여러 요소들을 테스터가 쉽게 조작할 수 있다는 점이 특징이다. 하지만 그러한 방식은 가상 환경이 실제 환경과 얼마나 비슷한가에 따라 결과가 달라진다는 한계가 있으며, 이에 대한 가장 효과적인 해결방안은 많은 노력을 통해 최대한 실제와 가까운 환경을 구현하는 것이다. 하지만, 대상 환경에서 장치의 수가 많아지거나 IoT와 같이 장치 스스로가 자율적으로 연결과 해지를 반복하는 복잡한 상황 역시 에뮬레이터를 실제와 가깝게 만드는 것을 더욱 어렵게 만드는 요소가 된다.

3. BitTorrent를 이용한 분산 환경 테스팅 기법

다양한 성능과 기능을 가진 여러 장치들이 서로 연결되어 있는 IoT환경에서 같은 하드웨어 상에서 같은 소프트웨어를 수행하는 동종의 장치들이 다수 존재한다고 할 때, 이러한 동종의 장치들은 해당 장치를 대상으로 하는 테스트의 일부분을 담당함으로써 최종적으로는 해당 장치에 대한 하나의 완성된 테스트를 수행할 수 있음을 가정해 볼 수 있다.

본 논문에서는 이러한 가정을 토대로 하여 IoT 환경에서 동일한 코드를 가진 동종의 장치들이 서로 협업을 통하여 테스트에 소모되는 자원을 줄이고, 이미 수행된 일부 테스트 결과를 공유함으로써 테스트를 완성하는 BitTorrent기반의 테스트 커버리지 측정 기법을 제안한다. 그림 1은 제안된 테스트 기법에서 테스트 수행 대상이 동종의 장치로 제한되는 것에 대한 설명으로 일반적인 IoT 환경에서 각각의 장치가 서로 구분 없이 연결되는 것과 동시에 본 논문에서 제시하는 테스트 데이터 교환과정이 동종의 장치들 사이에서만 이루어짐을 보인다.

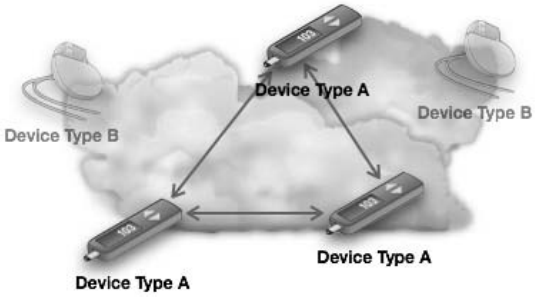


Fig. 1. Connections between homogeneous devices

분산 환경에서 다양한 장치들이 외부의 제어 없이 서로 연결되어 서비스를 제공하는 IoT의 대표적인 특성은 각 장치들이 특정한 규칙 없이 다양하게 연결됨으로써 높은 복잡도의 수많은 상황을 만들게 되고, 이는 수행되는 테스트 시나리오의 파악을 어렵게 만든다. 결과적으로 IoT 환경을 대상으로 하는 테스트는 더욱 어려워진다. 이에 본 논문에서는 이러한 문제의 해결 방안으로 BitTorrent기반의 효율적인 테스트 커버리지 측정 기법을 제안한다.

3.1 BitTorrent 알고리즘의 특징

BitTorrent 기법은 2001년 Bram Cohen이 처음 제안한 방법으로, 전 세계 1억 7천만 명이 사용하고 있는 대표적인 P2P 프로토콜이다. 하나의 완성된 파일을 가진 최초의 Seeder가 파일을 분할하여 배포를 시작한 후 Tracker를 통해 Seeder의 주소를 받은 다른 Leecher들이 조각된 파일을 무작위로 내려 받음과 동시에 Leecher 서로가 자기에게 있는 조각을 다른 Leecher에게 전송하고 자기에게 없는 조각을 전송받는 과정을 통해 최종적으로는 모든 Leecher들이 최초에 배포된 완성된 파일을 가지도록 하는 것이 BitTorrent의 핵심적인 특징이며, 각각의 Leecher들이 하나의 파일을 완성하기 위하여 서로가 가진 파일 조각을 교환하는 독특한 방법은 분산 환경의 장점을 극대화한 대표적인 기법이라 할 수 있다[7].

본 논문에서는 이러한 특징을 기반으로 하여 테스트 결과 파일을 BitTorrent 기법에서 각각의 Leecher들이 완성할 파일로 적용하여 각각의 Leecher들, 즉 테스트 대상 장치들이 각각의 테스트 결과를 해당 부분이 아직 테스트되지 않은 동종의 다른 장치들과 공유하여 최종적으로는 각각의 장치들이 중복된 테스트를 회피하면서 궁극적으로는 주어진 모든 테스트 과정을 완료 하도록한다. 이 장은 테이블을 이용한 대상코드의 블록화 기법과 이를 이용한 상세한 테스트 과정을 설명한다.

3.2 테이블을 이용한 대상 코드의 블록화

앞에서 언급한 바와 같이 각각의 테스트 대상 장치들은 각각이 실행한 테스트 결과를 조합하여 해당 장치에 대한 온전한 테스트 결과를 완성한다. 이를 위해서는 동종의 장치들이 최종적으로 완성하게 될 테스트 결과와 그 과정에서 교환할 테스트 결과의 단위 블록들이 명확하게 정의 되어야 함을 알 수 있다. 이에 본 논문에서는 표 1과 같은 형태로 테스트 블록 테이블을 정의하였다.

Table 1. Test Table for Test data Exchanging

Test Block Table		
Type ID	0x5	
Unique ID	0x2a	
Block Number	Pass/Fail	ID of Executor
1	Pass	0x2a
2	Not Yet	
3	Fail	0x31
4	in Testing	
…….	…….	…….

표 1은 크게 테스트 대상 장치의 ID를 나타내는 영역과 각 영역 즉, 각 함수의 수행 결과를 기록하는 2개의 영역으로 이루어진다. 먼저 장치의 ID 영역은 서로 같은 기능을 가진 동종의 장치들이 동일한 ID를 가짐으로써 Tracker로 하여금 서로 연결시켜야 할 장치들을 구분시켜 준다. Unique ID는 동종의 장치 사이에서 서로 다른 값을 부여하여 서로를 구분하고 아울러 실제 테스트가 수행된 장치의 ID를 기록하는데 사용된다. 테이블의 아랫부분은 실제 각 함수의 수행 결과가 기록 되는 부분이다. 본 논문에서는 대상의 장치가 단위 테스트를 수행하였음을 가정하고 테스트 커버리지의 기준을 함수로 정의하였다. 테스트의 대상이 되는 각 함수는 고유한 번호를 가지며 함수가 수행된 후 테이블에서 자신과 같은 번호를 가진 항목의 결과를 비교한다. 이미 테스트가 수행된 함수는 본래의 구현된 행위만을 수행하고, 아직 테스트가 수행되지 않은 함수는 수행 후 그 결과를 테이블에 관련된 사항과 함께 기록하여 불필요하게 중복된 테스트 행위로 인한 자원의 소모를 최소화하였다. 이때 함께 기록되는 사항은 정해진 규칙이 없으며, 해당 테스트의 목적에 따라 필요한 자료가 정의되어 저장되고, 이러한 자료는 처음으로 이 함수를 수행한 장치의 테이블에만 기록되고 다른 장치에는 해당함수의 테스트 여부만 전달하여 데이터 교환 과정에서 생기는 트래픽 역시 최소화 될 수 있도록 하였다.

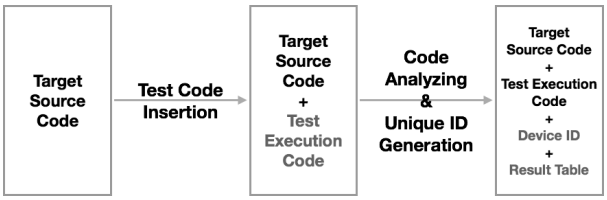


Fig. 2. Modification Processes of Target Code for Testing

3.3 테스트 수행 방법 및 전체 과정

각 테스트 대상 장치에 추가될 테스트 테이블은 테스트 대상이 되는 코드를 분할하여 분할된 각 블록에 고유 번호를 부여하고 해당되는 블록이 테스트 되었는지를 기록하는 방식으로 갱신된다. 대상 장치들은 테스트 테이블의 각 항목들 중 아직 테스트되지 않았던 블록이 새롭게 테스트되어 그 결과가 추가되었을 때 이를 동종의 다른 장치들에게 전송한다. 이를 위하여 Tracker는 테스트 대상 장치들이 초기에 구동되어 등록을 시도할 때 기존에의 등록된 동종의 장치 목록을 전달하고, 해당 장치는 Tracker로부터 주어진 장치들을 대상으로 자신이 테스트한 결과를 공유한다.

Tracker에 의한 장치의 목록 관리와 갱신은 새로운 동종의 장치가 등록될 때마다 반복됨으로써 모든 동종 장치들이 최신의 동종 장치 목록을 가지게 되고, 이와 동시에 새롭게 테스트에 참가한 장치가 빠른 시간에 이미 수행된 테스트

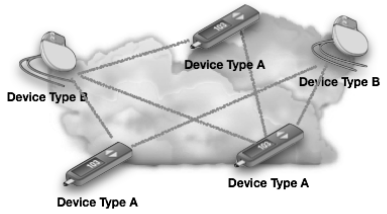


Fig. 3A. General IoT Environment

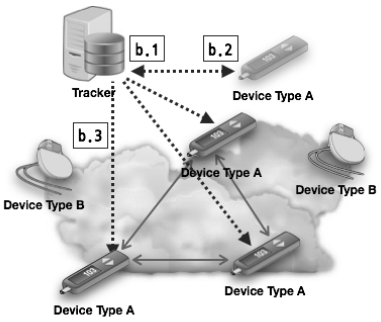


Fig. 3B. Entering in group as newbie

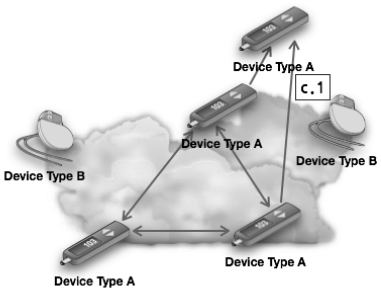


Fig. 3C. Joining in testing

결과를 습득하는 것을 가능하게 한다. 그림 2는 테스트 수행 코드와 장치 ID 및 결과 테이블이 추가되는 과정을 보인다.

앞서 제안한 테스트 과정은 대상 장치들의 테스트 항목이 적용된 테이블과 동종의 장치들간의 테스트 항목 결과를 주고받는 역할을 하는 코드가 동작함으로써 수행된다. 테스트 수행 코드는 크게 다음의 3가지 기능으로 구성된다.

- Tracker에 등록 및 장치 목록 관리 : 최초 구동 시에 주어진 Tracker를 찾아 자신을 등록하고 Tracker가 갱신해 주는 동종의 장치 목록을 관리하는 기능
- 테스트 결과 블록 송신 : 자신이 수행한 블록이 장치들 중 최초로 테스트를 완료한 블록일 때, 이를 목록에 등록된 다른 장치에게 알리는 기능
- 테스트 결과 블록 갱신 : 동종의 다른 장치들로부터 수신된 테스트 결과를 받아 자신의 테스트 테이블을 갱신하는 기능

본 논문에서 제시하는 테스트 방법은 먼저 대상 코드에 원래의 코드를 추상화한 테이블과 BitTorrent 기법으로 동종의 다른 장치와 테스트 결과 데이터를 주고받기 위한 기능을 수행하는 코드를 추가하는 과정에서부터 시작한다. 앞서 언급한 바와 같이 본 논문에서 대상으로 하는 테스트 장치들은 이미 함수 단위의 유닛테스트를 완료한 코드임을 가정하였으므로 테스트 과정을 기록할 테이블에서 각 항목은 그 최대 단위인 각 함수로 정의된다. 이를 위하여 대상 코드 내의 각각의 함수에 중복되지 않는 고유한 일련번호를 부여하고 이를 이용하여 테스팅 테이블을 구성한다.

앞 절에서 설명한 테스트 블록 테이블과 테스트 수행 코드가 추가된 장치들은 실제 구동환경과 동일한 환경에서 본래의 기능과 함께 테스트 기능을 수행한다. 그림 3은 이를 도식화한 것으로, 좌에서부터 차례대로 일반적인 IoT 환경에서의 각 장치들이 서로 데이터를 주고받는 과정이다. 이어서 다음 단계에서는 테스트 대상이 되는 새로운 장치가 네트워크상에 등장했을 때 Tracker에 자신을 알리고(b.1) Tracker로부터 동종의 장치 목록을 받은 후(b.2), Tracker가 동종의 다른 장치들에게 새로운 장치가 네트워크에 추가되었음을 알리는 과정을(b.3) 보이고 있다. 마지막 단계에서는 새롭게 추가된 장치가 이미 일부 테스트를 수행한 블록을 전송받는 모습(c.1)을 보여준다. 새롭게 추가된 장치는 이전에 수행되지 않았던 테스트 중 자신이 처음으로 수행하는 테스트에 대한 결과를 자신의 테이블에 기록함과 동시에 스스로가 수행한 테스트 결과 블록을 전송함으로써 테스트 과정에 참여한다.

4. BitTorrent기반의 테스트 환경 설계

이 장에서는 앞에서 설명한 개괄적인 방법론을 기반으로 대상 코드에 추가적으로 삽입되는 코드와 테스트 모듈의 구조, 마지막으로 테스트 시나리오를 통해 제안하는 테스트 기법을 구체화한다.

```
// 테스트 대상 블록의 번호를 지정
static int blockNumber = 4;
private static class MyBusListener extends BusListener {
    // 블록 번호를 이용하여 테스트 테이블상에서 해당되는 블록을 확인
    // 해당 블록이 아직 수행된적이 없다면
    if(this.testManager.getTestTableManager
        .getBlockState(blockNumber) == BlockState.NotYet)
    {
        // 테스트 모드로 진입
        this.testManager.setState(true);
        // 하나의 장치내에서 여러개의 쓰레드가 수행될 수 있음으로
        // 현재 테스트 중인 블록을 명시
        this.testManager.setTestingBlock(blockNumber);
    }

    // 원래의 코드 수행
    // ...

    // 해당 블록 종료 전 테스트 대상 블록인지를 확인
    if(this.testManager.getState == TestManagerState.inTest)
    {
        // 해당 블록에 테스트 블록 결과 기록
    }
}
```

Fig. 4. Modified Target code

4.1 테스트 대상 코드의 수정 방법

실제 테스트 수행 과정에서 테스트 대상 코드에 임의로 삽입한 테스트 코드가 수행되기 위해서는 현재 수행하고 있는 함수가 이미 테스트되었는지의 여부를 확인해야 한다. 그 결과에 따라 이미 테스트된 함수라면 함수 원래의 코드만을 수행하고, 처음 수행되는 함수라면 테스트 모드로 진입하여 함수 종료직전 함수를 테스트한 결과를 테이블에 저장한다. 그림 4는 IoT 환경의 대표적인 공개 플랫폼인 AllJoyn[9]의 예제코드를 기반으로 하여 테스트를 위해 추가되는 코드와 그 기능에 대하여 설명한다.

4.2 테스트 수행 코드의 구조 및 수행 원리

테스트 대상 코드에는 표 1에서 제시한 테이블과 더불어 실제 테이블을 갱신하고 테이블 상의 블록들을 다른 장치에 전송하며, 자신이 가진 테이블에 기록되지 않은 블록을 다른 장치로부터 전송받는 테스트 데이터 교환 코드가 함께 추가된다. 그림 5는 테스트 테이블 관리와 동종의 다른 장치와 블록을 교환하는 TestManager의 구조를 설명한다. 블록 수신과 관련된 부분은 모델에서 제외한다.

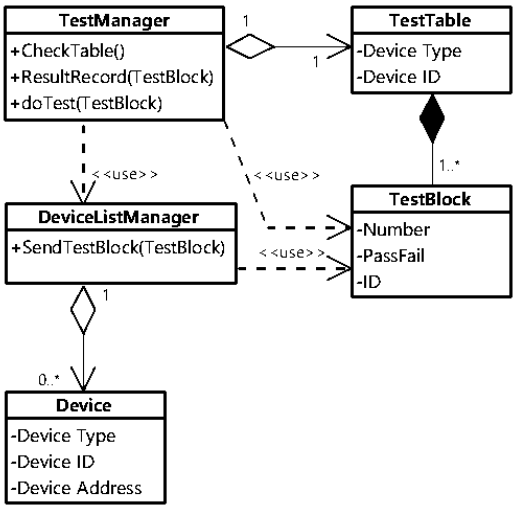


Fig. 5. Class Diagram of Test Module

TestManager는 전체 테스트를 총괄하는 클래스로써 TestTable과 DeviceListManager를 관리하다. 특정 함수가 수행이 되면 TestManager의 CheckTable()이 이미 삽입된 코드로 인하여 수행되고 이는 해당 함수가 이미 수행되었는지를 검사한다. 아직 수행된 적이 없는 함수는 테스트 모드로 상태를 변경하고, 최종 결과가 해당되는 블록에 저장한다. DeviceListManager는 Tracker를 통해 동종의 장치들의 목록을 관리하며, doTest()를 통해 수행된 테스트 결과가 블록에 적용되면 이를 자신이 관리하는 목록에 있는 장치들에게 다시 전송하는 기능을 가진다.

4.3 제안된 기법을 이용한 테스트 시나리오 구현

앞에서 언급하였듯이 본 논문에서 제시하는 테스트환경은 동종의 장치 사이에서 각각이 수행한 테스트 결과를 서로 교환하는 과정을 통해 진행된다. 그림 6은 이를 순차도로 표현한 것으로, 동종의 장치 A그룹에 새로운 장치가 추가된 상황을 가정하고 있다. 기존의 그룹에 새로운 장치가 추가되면 새로운 장치는 자신을 Tracker에 등록하고(2) Tracker는 새로운 장치에 기존의 장치 목록을 전송하고(2.1), 기존의 장치들에게는 새로운 장치의 정보를 전송한다(2.2-3).

다음은 대상 장치가 수행되는 상황으로써 각 함수가 실행되면 테이블을 확인하여 처음 수행되는지를 검사한 다음(4), 그 결과에 따라 테스트 코드를 수행하고 결과를 전송하거나 테스트 코드 수행을 종료한다.

마지막으로 테스트 테이블이 완성된 후 테스트로부터 결과 요청이 왔을 때(6) 해당 장치가 취합된 테스트 테이블을 돌려주는 상황을 보인다(6-1).

5. 평가 및 결론

본 논문에서는 IoT 환경을 대상으로 하는 테스트 과정에서 각 장치들의 테스트 커버리지 측정을 위하여 장치들이 가지는 하드웨어적 제약을 극복하기 위한 방법으로 BitTorrent 기반의 테스팅 기법을 제안하였다. 이를 위하여 테스트 대상이 되는 장치들에 원본 소스코드를 테이블로 추상화하여 삽입하고 BitTorrent 알고리즘을 이용하여 각 코드의 수행 결과를 교환할 테스트 수행 코드를 함께 삽입한 후, 이 장치들을 대상으로 테스트를 수행하는 방법을 사용하였다.

이러한 직접적인 대상 장치의 실행 환경은 기존의 에뮬레이터 상에서 대상 테스트 장치를 구동할 때 발생하는 실제 환경과의 차이와 그로 인하여 발생하는 테스트 결과의 오류를 피하기에 적합한 방법임을 알 수 있었다. 아울러 각각의 장치들 스스로가 수행한 테스트 결과를 동종의 다른 장치에 보내고, 동종의 다른 장치가 이미 수행한 결과를 받음으로써 실제와 같은 환경에서 장치의 테스트 과정을 분산하여 진행함으로써 중복된 테스트의 가능성을 제거해 불필요한 자원의 소모를 막았다.

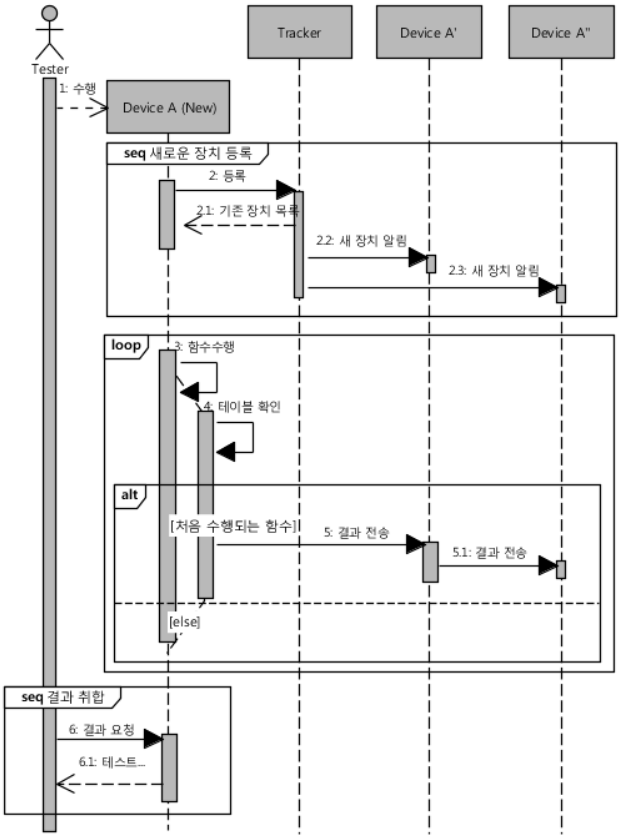


Fig. 6. Test Sequence Diagram

이어지는 연구에서는 앞에서 제시한 설계를 기반으로 하여 IoT 환경에서 다양한 플랫폼과 연동되는 테스트 모듈의 구현이 요구된다.

References

[1] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of Things", *International Journal of Communication Systems*, Vol.25, No.9, pp.1101-1102, Sep., 2012.

[2] R. M. Hierons, "Oracles for Distributed Testing", *IEEE Transactions on Software Engineering*, Vol.38, No.3, pp.629-641, 2012.

[3] T. Hanawa, T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, and M. Sato, "Large-Scale Software Testing Environment Using Cloud Computing Technology for Dependable Parallel and Distributed Systems", 2010 Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW), pp.428-433, 2010.

[4] R. M. Hierons, "Testing in the Distributed Test Architecture: An Extended Abstract", The Eighth International Conference on Quality Software, pp.11-14, 2008.

[5] I. V. McLoughlin, "Virtualized Development and Testing of Embedded Computing Clusters", Second International Conference on Networking and Computing(ICNC), pp.17-26, 2011.

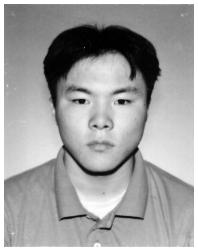
[6] J. Li and K. Moore, "A Runtime and Analysis Framework Support for Unit Component Testing in Distributed Systems", 40th Annual Hawaii International Conference on System Sciences, pp.261-261c, 2007.

[7] B. Cohen, "Incentives build robustness in BitTorrent", Workshop on Economics of Peer-to-Peer systems, 2003.

[8] BitTorrent, http://www.bittorrent.com/about

[9] AllJoyn, https://www.alljoyn.org/

[10] T. Hanawa, T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, and M. Sato, "Large-Scale Software Testing Environment Using Cloud Computing Technology for Dependable Parallel and Distributed Systems", Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW), pp.428-433, 2010.



류 호 동

e-mail : hodong@knu.ac.kr

2007년 경북대학교 전자전기컴퓨터학부 (학사)

2009년 경북대학교 컴퓨터학부(공학석사)

2009년~현 재 경북대학교 컴퓨터학부 박사과정

관심분야 : 모델 기반 테스팅, 분산환경 기반 테스팅

이 우 진

e-mail : woojin@knu.ac.kr

1992년 경북대학교 컴퓨터과학과(학사)

1994년 한국과학기술원 전산학과(공학석사)

1999년 한국과학기술원 전산학과(공학박사)

1999년~2002년 한국전자통신연구원 S/W 공학연구부 선임연구원

2002년~현 재 경북대학교 컴퓨터학부 정교수

관심분야 : 임베디드 소프트웨어 모델링 및 분석, 정형적 방법, 임베디드 소프트웨어 테스팅 등