논문 13-38A-10-12

한국통신학회논문지(J-KICS) '13-10 Vol.38A No.10
http://dx.doi.org/10.7840/kics.2013.38A.10.908

기기 간 직접통신을 위한 모바일 어플리케이션 및 서비스 디스커버리 프로토콜

최 계 원◆, 이 현°, 장 성 철*,

Mobile Application and Service Discovery Protocol for Device-to-Device Communication

Kae Won Choi◆, Hyun Lee°, Sung Cheol Chang*

요 약

본 논문에서는 기기 간 직접(device-to-device) 통신 시스템에서 근접한 디바이스의 모바일 어플리케이션 및 서비스를 발견하기 위한 디스커버리 프로토콜을 제안한다. 기기 간 직접통신 기술을 기반으로 모바일 소셜네트워크, 모바일 마케팅 등의 근접 기반 어플리케이션을 실현할 수 있다. 이를 위해 우선적으로 주변에 있는 수많은 디바이스에서 원하는 어플리케이션을 찾아내는 디스커버리 프로토콜의 설계가 필수적이다. 기반 시설이 없는 에드혹 망에서 디스커버리 프로토콜을 구현하기 위해서는 디바이스 내부의 어플리케이션 정보를 축약해서 디스커버리 코드를 생성하고 이를 주기적으로 방송하는 방법을 쓸 수 있다. 본 논문에서는 해시함수 및 블룸필터(Bloom filter)를 이용하여 디스커버리 프로토콜을 설계하고 이의 성능을 수학적으로 분석한다.

Key Words : Device-to-Device Communication, Service Discovery, Mobile Application, Hash Function, Bloom Filter

ABSTRACT

In this paper, we propose a discovery protocol for finding nearby mobile applications and services in a device-to-device communications system. The device-to-device communication technology enables proximity-based services such as mobile social networks and mobile marketing. For realizing these proximity-based services, it is essential to design a discovery protocol which pinpoints the devices with mobile applications of interest among hundreds and thousands of devices in proximity. In the infrastructure-less networks such as ad hoc networks, we can design the discovery protocol that periodically broadcasts a short discovery code containing the compressed information of the mobile applications. In this paper, we design the discovery protocol with the discovery code generated by using a hash function and a Bloom filter. We also mathematically analyze the performance of the proposed protocol.

Ⅰ. 서 론

기기 간 직접(device-to-device) 통신은 기지국 등의 인프라를 거치지 않고 여러 디바이스가 정보를 주고

※ 본 연구는 미래부가 지원한 2013년 정보통신·방송(ICT) 연구개발사업의 연구결과로 수행되었습니다.
◆ First Author : 서울과학기술대학교 컴퓨터공학과, kaewon.choi@gmail.com, 종신회원
° Corresponding Author : 한국전자통신연구원 무선액세스시스템연구부, 무선분산접속연구실, hyunlee@etri.re.kr, 정회원
* 한국전자통신연구원 무선액세스시스템연구부, 무선분산접속연구실, scchang@etri.re.kr
논문번호 : KICS2013-09-411, 접수일자 : 2013년 9월 16일, 최종논문접수일자 : 2013년 10월 21일

받을 수 있는 이동통신 기술을 뜻한다[1]. 셀룰러 네트워크 기반의 기기 간 직접통신 기술로써 3GPP에서 LTE(Long-Term Evolution)에 직접통신을 적용하기 위해 최근 표준화를 시작하였다[2]. 또한 비면허 대역에서 기존의 무선랜 표준을 확장한 와이파이 다이렉트 기술이 개발되었고[3], IEEE 802.15.8 대상인식통신 표준화 작업이 진행 중이다.

기기 간 직접통신의 대표적인 장점으로써 i) 통신거리가 짧아지며 ii) 기지국을 거치던 두 단계 전송을 한 단계로 줄일 수 있고 iii) 동시에 여러 디바이스 쌍의 통신을 지원하며 iv) 새로운 종류의 근접 기반 무선 어플리케이션을 도입할 수 있는 점 등을 들 수 있다[4]. 이러한 장점 중에서 근접 기반 어플리케이션의 도입은 무선 통신 생태계에 가장 큰 변화를 가져올 것으로 예상된다[5].

근접 기반 어플리케이션은 두 개 이상의 디바이스가 물리적으로 근접한 위치에 있을 때 각 디바이스 내부의 어플리케이션이 통신을 원하는 상대 어플리케이션을 발견하고 통신 연결을 설정하여 기지국 등의 기반 시설을 거치지 않는 기기 간 직접통신을 통해 제공하는 서비스를 뜻한다. 근접 기반 어플리케이션의 종류에는 대표적으로 모바일 소셜네트워크[6], 모바일 마케팅, 근접 기반 다중 사용자 게임, 미디어 파일 교환 등을 들 수 있다. 모바일 소셜네트워크 어플리케이션을 사용하면 주변에 친구로 등록된 사용자가 있을 경우 이를 사용자에게 알려주고 직접통신 연결을 통해 채팅 등을 할 수 있다. 또한, 취미 등의 프로파일을 입력하면 이에 맞는 주변의 다른 사용자를 찾아주는 기능도 할 수 있을 것이다. 모바일 마케팅 어플리케이션의 경우 식당 등의 상점에서 제공하는 메뉴나 가격 혹은 할인 등의 정보를 주변에 있는 사용자에게 실시간으로 광고할 수 있다. 근접한 사용자가 자신이 원하는 파일을 가지고 있을 때 이를 검색하여 파일을 전송 받는 미디어 파일 교환도 근접 기반 어플리케이션의 한 예이다.

이러한 근접 기반 어플리케이션을 구현하기 위해서는 우선적으로 수많은 주변 사용자로부터 원하는 어플리케이션을 찾아내는 디스커버리 프로토콜을 설계하는 것이 매우 중요하다. 가령 주변 1km 반경 안의 사용자를 찾아내야하는 상황을 가정해보자. 도심지의 경우에는 1km 반경 내에 수천에서 수만 개에 이르는 모바일 기기가 존재할 수 있으며 각각의 모바일 기기마다 수십 가지의 어플리케이션이 설치되어 있을 수 있다[7] (그림 1 참조). 이러한 상황에서 적절한 시간 내에, 가능한 적은 양의 무선 자원을 사용하고, 배터리 소모를 최소화하면서 원하는 어플리케이션을 탐색하는 프로토콜을 설계하는 것은 간단한 일이 아니다.

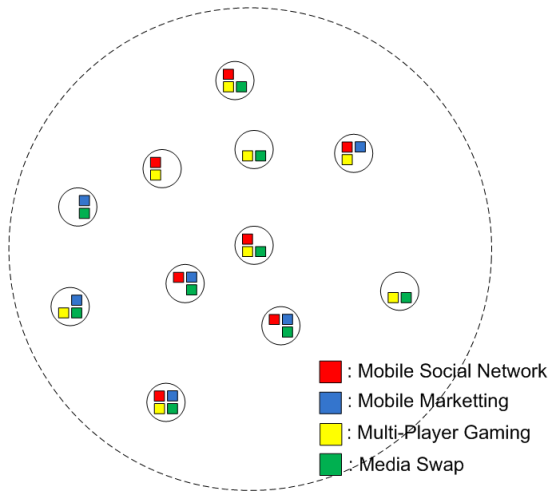

그림 1. 기기 간 직접통신에서의 어플리케이션 발견 시나리오
Fig. 1. Application discovery scenario in device-to-device communication

현재의 통신 시스템에서는 대부분 기기 내부의 어플리케이션 유무와 상관없이 우선적으로 접속 과정을 거친 후 이미 알고 있는 어플리케이션에 연결하거나 상위 계층의 서비스 디스커버리 프로토콜을 이용해 원하는 어플리케이션을 발견한다[8]. 이러한 예로 UPnP[9], 블루투스 SDP(service discovery protocol)[10], 와이파이 다이렉트의 802.11u[11] 등을 들 수 있다. 그러나 이러한 방식으로 수천 개의 기기에서 원하는 어플리케이션을 발견하기 위해서는 각각의 기기에 접속해 상위 계층 프로토콜에 연결하고 찾지 못하였을 경우 접속을 끊는 과정을 반복해야 하고 이러한 과정에서 많은 양의 유니캐스트 정보가 전달되야 하므로 매우 비효율적이라 할 수 있다. 따라서, 본 논문에서는 정보 전달을 최소화하면서 빠른 시간 내에 원하는 어플리케이션을 찾아낼 수 있는 효율적인 디스커버리 프로토콜을 설계한다.

기기 간 직접통신은 기지국 등의 기반 시설이 있는 셀룰러 망이나 기반 시설이 없는 에드혹 형태의 네트워크에서 모두 사용이 가능하며 기반 시설의 유무에 따라 서로 다른 방식으로 디스커버리 프로토콜을 설계할 수 있다. 기반 시설이 있어서 인터넷 등의 망에 접속이 가능한 상황에는 퀄컴이 제안한LTE-Direct[5]와 같이 중앙 집중적인 구조의 네임 서버[12]에 어플리케이션 관련 정보를 저장하여 어플리케이션 발견을

위한 코드를 전달하는 방식을 사용할 수 있다. 그러나 기반 시설이 없는 망에서는 상기 방식의 적용이 불가능하다. 본 논문에서는 기반 시설이 없는 기기 간 통신망에서 해시함수(hash function)이나 블룸필터(Bloom filter)[13,14] 등을 이용하여 어플리케이션 발견을 위한 코드를 생성하고 이를 이용하는 프로토콜에 대해 다룬다. 제안하는 디스커버리 프로토콜은 특정 통신 기술을 가정하고 있지는 않으나 와이파이 다이렉트나 IEEE 802.15.8 대상인식통신 등에 추후 응용될 수 있을 것이다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서는 본 논문에서 제안하는 디스커버리 프로토콜의 구조 및 절차에 대하여 설명한다. Ⅲ장에서는 제안하는 디스커버리 프로토콜에 사용되는 디스커버리 코드를 생성하는 방법을 설명한다. Ⅳ장에서는 디스커버리 프로토콜의 성능을 수학적으로 분석하고 Ⅴ장에서 시뮬레이션과 성능 분석 결과를 제시한다. 마지막으로 Ⅵ장에서 본 논문의 결론을 맺는다.

Ⅱ. 제안하는 디스커버리 프로토콜 구조 및 절차

효율적인 디스커버리 프로토콜을 설계하기 위해서는 어플리케이션 발견 과정이 통신의 하위 계층에서 구현되어야 하며 특정 디바이스에 접속하기 이전에 최대한 빠른 시간에 이루어져야 한다. 또한 어플리케이션이 디스커버리 프로토콜에 접근하여 자신을 등록하고 다른 디바이스의 발견 결과를 얻어낼 수 있는 API(Application Programming Interface)를 제공하여야 한다. 그러므로 디스커버리 프로토콜에는 상위 계층 어플리케이션의 요구 사항을 통신의 하위 계층에서 효율적으로 수행하기 위한 일련의 절차를 통신의 상위부터 하위 계층까지 규정하여야 한다.

한 디바이스 내의 각 어플리케이션은 자신에 관한 정보를 API를 통해 디스커버리 프로토콜 모듈에 등록한다. 어플리케이션에 대한 정보는 기본적으로 어플리케이션의 이름을 나타내는 문자열과 어플리케이션의 특징을 나타내는 속성(attribute)를 들 수 있다. 본 논문에서는 어플리케이션의 이름만을 등록하고 이를 바탕으로 이름이 일치하는 어플리케이션을 다른 기기에서 찾아내는 디스커버리 프로토콜을 다룬다. 본 논문의 프로토콜을 확장하여 이름 외에 속성을 이용하는 보다 일반적인 프로토콜을 설계할 수 있을 것이다.

물리계층 및 MAC(medium access control) 계층 등의 통신 하위 계층에서의 어플리케이션 발견을 효율적으로 수행하기 위해서는 각 디바이스가 자신에 등록된 어플리케이션의 이름을 주기적으로 방송해야 한다. 일반적으로 통신 디바이스는 근접 디바이스의 발견을 위해 자신의 식별자(identifier)가 포함되어 있는 비콘(beacon) 등의 신호를 주기적으로 방송하며 어플리케이션의 발견을 위해서는 추가적으로 어플리케이션 광고도 방송해야 한다. 그러나 어플리케이션 이름을 그대로 방송하면 너무 많은 양의 정보를 전달하게 되어 무선 통신의 자원을 과도하게 소모하게 될 수 있다. 그림 2의 예에서 디바이스 1내에는 Mobile Social Network, Mobile Marketing, Media Swap이라는 이름을 가진 세 개의 어플리케이션이 등록되어 있다. 문자 당 16비트의 크기로 인코딩하는 유니코드(Unicode) 방식을 사용한다고 가정하면 디바이스 1은 최소한 768비트 이상을 어플리케이션 이름의 광고를 위해 주기적으로 전송해야 한다.

어플리케이션 광고에 필요한 비트수를 줄이기 위해 해시함수나 블룸필터를 이용하여 디스커버리 코드를 생성하는 방법을 사용할 수 있다. 디바이스 i 내의 어플리케이션 이름의 집합을 N_i라고 하자. 예를 들어, 그림 2에서 디바이스 1의 어플리케이션 이름의 집합은 N_i={"Mobile Social Network", "Mobile Marketing", "Media Swap"}이 된다. 디바이스 i는 N_i를 표현하기 위한 정보를 모두 방송하는 대신 이로부터 짧은 디스커버리 코드 c_i를 생성하여 주기적으로 방송한다. 디스커버리 코드를 생성하는 함수를 F라 하면

$$c_i = F(N_i) \tag{1}$$

의 관계가 성립한다.

디스커버리 코드에는 어플리케이션의 이름 집합을 축약한 정보만이 담겨 있으므로 디스커버리 코드로부터 어플리케이션 이름을 복구할 수는 없다. 대신 특정 어플리케이션의 이름이 송신자의 어플리케이션 이름의 집합에 포함되었는지의 여부를 디스커버리 코드를 통해 시험할 수 있다. 디바이스 i로부터 수신한 디스커버리 코드 c_i에 어플리케이션 이름 x가 포함되어 있는지 시험하기 위해 함수 R을 사용하여 다음과 같이 결과값 r을 얻을 수 있다.

$$r = R(x, c_i) \tag{2}$$

결과값 r은 어플리케이션 이름 x가 있다고 판단된

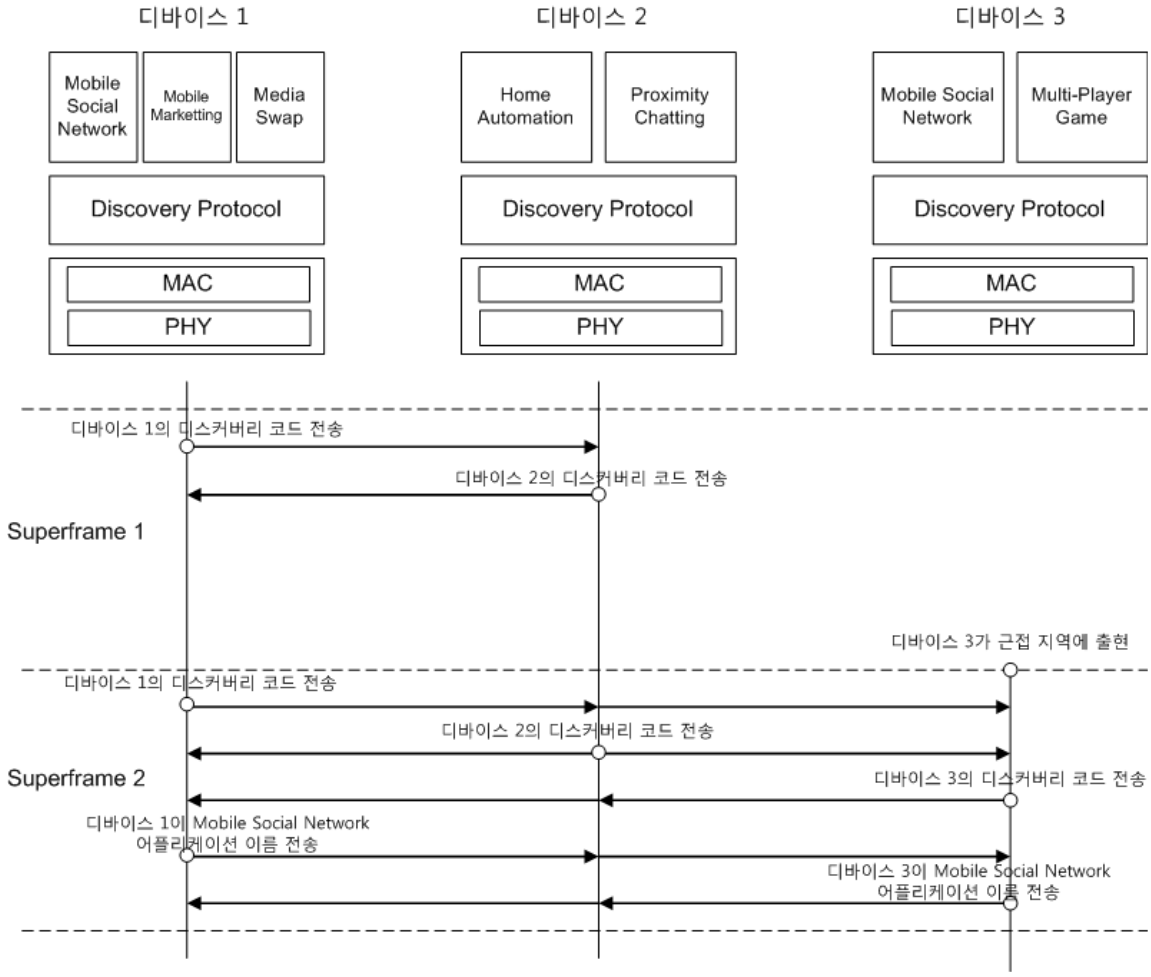

그림 2. 디스커버리 프로토콜 동작 예제
Fig. 2. An example of discovery protocol operation

경우 Y로 주어지며 없다고 판단된 경우 N으로 주어진다.

디스커버리 코드를 이용하여 어플리케이션의 존재 유무를 테스트할 때 정보의 손실로 인해 오류가 발생할 수 있다. 어플리케이션이 존재하지 않지만 존재한다고 판단한 경우는 거짓 양성(false positive) 오류가 발생했다고 할 수 있으며 어플리케이션이 존재하지만 존재하지 않는다고 판단한 경우는 거짓 음성(false negative) 오류로 분류할 수 있다. 이 때, 어플리케이션 발견이 실패하는 경우를 막기 위해서 거짓 음성 오류가 발생하지 않도록 디스커버리 코드를 설계한다. 예를 들어 x =“Mobile Social Network”인 경우 디바이스 1으로부터 받은 디스커버리 코드 c_1을 이용해서 x의 존재 여부를 시험하였을 때 “Mobile Social Network”가 N_1에 포함되어 있으므로 결과값인 r은 항상 긍정인 Y가 나와야 한다. 반면에 거짓 양성 오류는 발생할 수 있으므로 x =“Multi-player game”이라 했을 때 c_1으로 시험한 결과가 N_1에 “Multi-player game”이 없음에도 불구하고 Y로 나올 수 있다.

디바이스는 등록된 각 어플리케이션에 대해 다른 디바이스로부터의 디스커버리 코드에 포함 여부를 시험하여 최소 하나의 디스커버리 코드에 포함되었는지를 판단한다. 디바이스 i의 어플리케이션 중에 상기 조건을 만족하는 어플리케이션의 이름의 집합인 U_i를 다음과 같이 구할 수 있다.

$$U_i = \{x \in N_i | R(x, c_j) = Y, \exists\ j \in W_i\} \quad (3)$$

여기서 W_i는 디바이스 i 주변에 위치한 디바이스의 집합이다. 거짓 양성 오류가 발생할 수 있으므로 디바이스 i는 어플리케이션이 U_i에 속한다고 해서 해당 어플리케이션을 가진 디바이스를 찾았다고 판단할 수는 없다.

어플리케이션의 존재 여부를 명확히 판단하기 위해 디바이스 i는 U_i에 속하는 어플리케이션이 존재할 경우 U_i를 방송한다. 결과적으로 디바이스 i는 W_i에 속하는 모든 주변 디바이스 j로부터 U_j를 수신받게 된다. 디바이스 i는 자신에게 등록된 어플리케이션 x가 U_j에 속하는 경우 어플리케이션 x가 디바이스 j에 속한다고 결론내릴 수 있다.

제안하는 프로토콜을 사용하면 주변 디바이스와의 공통 어플리케이션을 적은 양의 정보 전송만으로 모두 찾을 수 있다. 디바이스 i가 디바이스 j에 근접해 있을 때 디바이스 i는 디바이스 j와의 공통 어플리케이션이 $N_i \cap N_j$인 것을 판단할 수 있어야 한다. 디바이스 i가 c_i를 전송하면 주변에 위치한 디바이스 j는 c_i를 수신한 후 U_j를 전송하는데 $U_j \subset N_j$를 만족하며 거짓 음성 오류가 없으므로 $(N_i \cap N_j) \subset U_j$이다. 그러므로 다음과 같은 조건을 만족한다.

$$N_i \cap N_j = N_i \cap U_j \quad (4)$$

식 (4)로부터 디바이스 i가 U_j를 수신하여 $N_i \cap N_j$를 구할 수 있는 것을 알 수 있다. 또한 디바이스 i는 주기적으로는 c_i만을 전송하며 주변에 새로운 디바이스가 나타나는 경우에만 U_i를 전송하면 되므로 전송하는 정보의 양이 크지 않다.

그림 2에 제안하는 디스커버리 프로토콜의 동작 예제를 제시하였다. 본 예에서는 각 디바이스가 수퍼프레임(superframe) 단위로 디스커버리 코드를 전송한다고 가정하였다. 수퍼프레임 1에서는 디바이스 1과 디바이스 2만 존재하나 공통 어플리케이션은 없는 상황이므로 디스커버리 코드만 전송된다. 수퍼프레임 2에서 디바이스 3이 근접한 영역에 나타나는 상황을 고려하자. 이 때, 디바이스 3은 디바이스 1의 디스커버리 코드 c_1을 수신하여 Mobile Social Network 어플리케이션이 존재한다고 판단한다. 그러므로 U_3 ={"Mobile Social Network"}가 된다. 마찬가지로 디바이스 1에서 디스커버리 코드 c_3를 받고 U_1 ={"Mobile Social Network"}로 판단한다. 그러므로 디바이스 1과 3에서는 U_1과 U_3, 즉 "Mobile Social Network" 어플리케이션의 이름을 각각 전송하며 서로 상대방에 해당 어플리케이션이 공통적으로 존재하는 것을 알게 된다.

Ⅲ. 디스커버리 코드 생성 알고리즘

3.1. 근사 멤버쉽 문제

본 장에서는 제안하는 디스커버리 프로토콜에 사용할 디스커버리 코드 c_i를 생성하는 방법에 대해 설명한다. 디스커버리 코드를 토대로 디바이스 내의 어플리케이션의 존재 여부에 대한 시험을 할 수 있어야 하며 디스커버리 코드는 어플리케이션의 이름을 명시적으로 표현한 정보보다 매우 작은 크기를 가져야 한다. 그러므로 디스커버리 코드 생성은 근사 멤버쉽 문제(approximate membership problem)과 매우 유사한 특징을 가진다고 할 수 있다.

근사 멤버쉽 문제는 주어진 집합 S에 대해 특정 원소 y가 S에 포함되었는지 여부를 묻는 질문인 근사 멤버쉽 쿼리(approximate membership query)에 답할 수 있는 데이터 구조를 만들어내는 것을 목적으로 한다. 이 때, 데이터 구조는 집합 S를 그대로 표현하는 것보다 매우 작은 정보량을 가져야 하며 집합 S에서 직접 검색하는 것보다 빠른 속도를 제공할 수 있어야 한다.

상기 조건에 만족하는 데이터 구조를 생성하기 위해 일정 수준의 오류를 감수해야 한다. 거짓 양성 오류는 $y \notin S$임에도 $y \in S$라고 판단하는 경우를 뜻하며 근사 멤버쉽 문제에서는 거짓 양성 오류를 허용한다. 그러나 $y \in S$일 때 $y \notin S$라고 판단하는 거짓 음성 오류는 허용하지 않는다. 거짓 음성 오류를 허용하지 않으므로 최소한 집합 S에 속하는 원소는 모두 발견할 수 있다. 그러므로 근사 멤버쉽 쿼리는 집합 S에 속하지 않을 것으로 생각되는 원소를 높은 확률로 걸러내는 필터링(filtering) 기능을 제공한다고 할 수 있다.

근사 멤버쉽 쿼리를 구현할 수 있는 방법으로 해시함수와 블룸필터를 들 수 있으며 본 논문에서는 두 방법을 모두 고려한다.

3.2. 해시함수 기반의 디스커버리 코드 생성 알고리즘

해시함수는 주어진 데이터로부터 고정된 길이의 축약된 출력 데이터를 만들어낸다. 임의 길이의 문자열 x를 δ비트의 해시값으로 변환하는 해시함수 H가 있다고 하자. 그림 3에서는 δ가 4비트 일 때 해시함수의 예를 보여준다. 여기서 H("Mobile Social Network")=2, H("Mobile Marketting")=13, H("Media Swap")=13이 된다.

해시함수는 같은 문자열을 넣었을 때 항상 같은 결과를 출력해야 하며 가능한 해시값의 범위에 균등하게 결과가 나오도록 하는 것이 바람직하다. 해당 조건을 만족하는 대표적인 해시함수로는 암호화에 널리 사용되는 SHA-1이나 MD-5를 들 수 있다.

해시함수는 데이터의 길이를 축약하므로 두 개 이상의 다른 문자열이 같은 해시값으로 대응되는 충돌

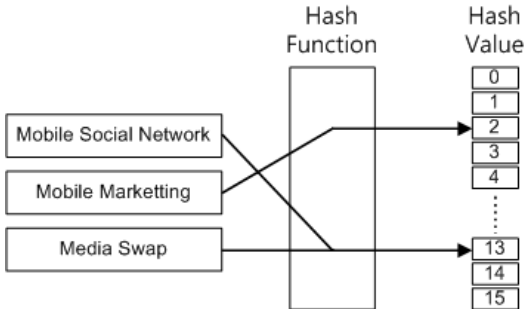

그림 3. 해시함수의 예
Fig. 3. Example hash function

현상이 일어날 수 있다. 그림 3에서 Mobile Marketing과 Media Swap이 동일한 값인 13으로 대응되어 충돌이 발생하는 것을 볼 수 있다.

디스커버리 코드를 생성하기 위해 해시함수를 다음과 같이 사용할 수 있다. 디바이스 i내의 어플리케이션 이름의 집합인 N_i에 포함된 어플리케이션 이름의 개수를 m개라고 하고 각각의 어플리케이션 이름을 $n_1,n_2,...,n_m$이라고 하자. δ비트의 해시값을 출력하는 해시함수 H를 이용하면

$$H(n_1) \| H(n_2) \| ... \| H(n_m) \quad (5)$$

의 형태를 가지는 $m\delta$ 길이의 디스커버리 코드를 생성할 수 있다. 여기서 ' ‖ '는 비트열을 연결하는 연산자이다.

위와 같이 만들어진 디스커버리 코드를 수신하였을 때 어플리케이션 이름 x의 존재 여부를 시험한다고 하자. 이 때, $H(x)=H(n_l)$를 만족하는 $l=1,...,m$이 하나라도 있다면 어플리케이션 x가 존재한다고 판단하고 하나도 없다면 어플리케이션 x가 존재하지 않는다고 판단한다. 이러한 방식을 사용하면 어플리케이션 x가 디바이스 i에 존재하는 경우 $x=n_j$인 j가 있으므로 이에 대해 $H(x)=H(n_j)$가 성립하고 수신측에서는 어플리케이션 x가 존재한다는 결론을 내릴 수 있다. 그러므로 거짓 음성 오류는 없다고 할 수 있다.

반면에 거짓 양성 오류는 발생할 수 있다. 어플리케이션 x가 디바이스 i에 없는 경우 $l=1,...,m$에 대해 $x \neq n_l$이 성립한다. 그러나 $x \neq n_l$인 경우라도 $H(x)=H(n_l)$이 만족할 수 있으며 해시값이 균등하게 분포한다고 가정하였을 때 거짓으로 일치할 확률은 0.5^{δ}가 된다. 그러므로 총 m개의 해시값 중 하나라도 거짓으로 일치할 확률은

$$1-(1-0.5^{\delta})^m \quad (6)$$

으로 주어지며 이를 거짓 양성 확률(false positive probability)라 한다.

디스커버리 코드를 생성할 때 어플리케이션의 수에 비례해서 코드의 길이가 증가하는 가변길이 코드를 사용할 수도 있으며 어플리케이션의 수와 상관없이 일정한 길이의 코드를 사용하는 고정길이 코드를 사용할 수도 있다. δ비트의 해시값을 출력하는 해시함수를 사용했을 때 코드의 길이는 $L=m\delta$가 된다. 가변길이 코드를 사용하였을 경우 δ를 고정시키고 m에 따라 코드 길이 L이 변화하게 된다. 반면 고정길이 코드를 사용하면 코드 길이 L을 고정시키고 해시값의 크기를 $\delta = \lfloor L/m \rfloor$로 변화시킨다.

3.3. 블룸필터 기반 디스커버리 코드 생성 알고리즘

블룸필터[13]는 근사 멤버십 문제를 해결하는 가장 단순하면서 효율적인 방법이라 할 수 있다. 블룸필터의 비트수를 L이라고 하자. L비트의 블룸필터를 생성하기 위해서는 0에서 $(L-1)$사이의 해시값을 출력하는 k개의 서로 다른 해시함수 $H_1,...,H_k$를 사용한다.

디바이스 i의 블룸필터를 다음과 같이 생성할 수 있다. 블룸필터의 q번째 비트를 $B(q)$라 하자. 우선 블룸필터의 모든 비트를 $B(q)=0$으로 초기화 시킨다. 디바이스 i에 존재하는 각 어플리케이션 $x \in N_i$에 대하여 k개의 해시함수의 해시값을 구하고 블룸필터에서 해시값에 해당하는 자리수의 비트를 1로 바꾼다. 즉, $x \in N_i$인 모든 어플리케이션 x마다 해시값 $H_j(x)$을 $j=1,...,k$에 대해 모두 구하여 $B(H_j(x))=1$로 설정한다. 그림 4에 블룸필터 생

성 예를 보였다. 여기서 블룸필터의 길이 $L=16$이며 해시함수의 개수는 $k=3$으로 설정하였다.

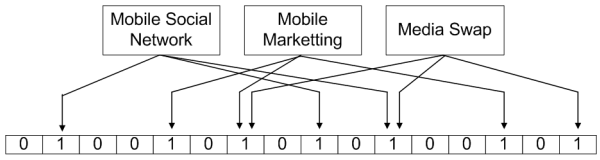

그림 4. 블룸필터의 예
Fig. 4. Example Bloom filter

디바이스 i로부터 블룸필터 기반의 디스커버리 코드를 B를 수신하였을 때 어플리케이션 x가 존재하는 것을 시험하려면 $B(H_j(x))=1$이 $j=1,...,k$에 대해 모두 만족하는 것을 보인다. 만일 $x \in N_i$라면 수신 디바이스는 항상 x가 디바이스 i에 존재하는 것으로 판단하므로 거짓 음성 오류는 발생하지 않는다.

어플리케이션 x가 디바이스 i에 없는 경우에도 있는 경우로 판단하는 거짓 양성 오류가 발생할 수 있다. 거짓 양성 확률은 근사적으로 다음과 같이 구할 수 있다. 디바이스 i에 m개의 어플리케이션이 있을 경우에 k개의 해시함수를 이용하여 길이가 L인 블룸필터를 생성했을 때 하나의 특정 비트가 0일 확률은 다음과 같이 계산할 수 있다.

$$p=(1-1/L)^{km} \simeq \exp(-km/L). \quad (7)$$

수신 디바이스에서 디바이스 i에 존재하지 않는 어플리케이션 x의 존재 유무를 판단하기 위해 k개의 비트를 검사하였을 때 모든 비트가 1일 확률이 거짓 양성 확률이 되며 이는

$$(1-p)^k=(1-\exp(-km/L))^k \quad (8)$$

로 계산된다. 거짓 양성 확률을 최소화하는 k의 값은 $k^*=\ln 2 \cdot (m/L)$이며 이를 k에 대입하면 최소 거짓 양성 확률

$$0.5^{\ln 2 * m/L}=\kappa^{m/L} \quad (9)$$

을 구할 수 있다. 여기서 $\kappa=0.5^{\ln 2} \simeq 0.6185$가 된다.

Ⅵ. 디스커버리 프로토콜 성능 분석

4.1. 성능 분석을 위한 시스템 모델

본 장에서는 제안한 디스커버리 프로토콜의 성능을 분석한다. 간략한 분석을 위해 두 개의 디바이스만이 존재하는 환경을 가정한다. 두 디바이스를 각각 디바이스 1과 디바이스 2라 하자. 각 디바이스에는 다수의 모바일 어플리케이션이 동작하고 있을 수 있다. 디바이스 1과 디바이스 2에 동시에 존재하는 어플리케이션의 수를 A_C라고 하고 디바이스 1에만 있는 어플리케이션의 수를 A_1, 디바이스 2에만 있는 어플리케이션의 수를 A_2로 나타낸다. 이 때, 디바이스 1과 디바이스 2에 있는 어플리케이션의 수는 각각 A_C+A_1과 A_C+A_2가 된다.

어플리케이션의 수 A_C, A_1, A_2는 각각 포아송 분포를 따르는 확률변수이며 서로 독립적이라 가정한다. A_C의 평균은 $\mathrm{E}[A_C]=\alpha_C$ 이며 A_1와 A_2의 평균은 $\mathrm{E}[A_1]=\mathrm{E}[A_2]=\alpha_N$라고 하자.

각 어플리케이션을 지칭하는 이름을 이용하여 동일 어플리케이션을 탐색하며 이름 하나의 평균 길이는 s비트라고 하자.

4.2. 해시함수 기반 프로토콜 성능 분석

상기 모델을 바탕으로 해시함수 기반의 제안 프로토콜의 성능을 분석한다. 분석은 가변길이 코드를 사용했을 때와 고정길이 코드를 사용했을 때로 나누어 진행한다.

가변길이 코드를 사용한 경우 각각의 어플리케이션의 이름은 δ비트의 해시 코드로 인코딩되고 총 코드의 길이는 디바이스 1과 디바이스 2에서 각각 $\delta(A_C+A_1)$과 $\delta(A_C+A_2)$가 된다. 그러므로 디바이스 1, 2에서 전송되는 디스커버리 코드 길이의 평균은 각각 $\delta(\alpha_C+\alpha_N)$이 된다.

디바이스 1이 디바이스 2의 디스커버리 코드를 수신한 후 디바이스 1에만 존재하는 어플리케이션에 대해서 디바이스 2에도 존재한다고 잘못 판단할 확률이 거짓 양성 확률이라 할 수 있으며 이는 디바이스 2가 디바이스 1의 디스커버리 코드를 수신하는 경우에도 마찬가지이다. 디바이스 1의 거짓 양성 확률은 하나의 해시 코드에 대해서 0.5^{δ}이다. 디바이스 2의 디스커버리 코드에 총 A_C+A_2개의 해시 코드가 있는 것을 고려하면 거짓 양성 확률은

$$1-(1-0.5^{\delta})^{(A_C+A_2)} \quad (10)$$

로 계산할 수 있다.

디바이스 1이 전송하는 어플리케이션 이름의 평균 비트수는 다음과 같이 계산된다.

$$\begin{aligned} &s \cdot \mathrm{E}[A_C+A_1(1-(1-0.5^{\delta})^{(A_C+A_2)})] \quad (11)\\ &= s\alpha_C+s\alpha_N(1-\mathrm{E}[(1-0.5^{\delta})^{(A_C+A_2)}]). \end{aligned}$$

여기서, A_C+A_2가 평균이 $\alpha_C+\alpha_N$인 포아송 분포를 따르므로 $\mathrm{E}[(1-0.5^{\delta})^{(A_C+A_2)}]$는 다음과 같이 계산할 수 있다.

$$\begin{aligned} &\mathrm{E}[(1-0.5^{\delta})^{(A_C+A_2)}] \quad (12)\\ &= \exp(-0.5^{\delta}(\alpha_C+\alpha_N)). \end{aligned}$$

그러므로 디바이스 1이 전송하는 어플리케이션 이름의 평균 비트수는

$$s\alpha_C+s\alpha_N(1-\exp(-0.5^{\delta}(\alpha_C+\alpha_N))) \quad (13)$$

이 되며 디바이스 2의 경우도 동일한 값을 가진다.

따라서, 가변길이 해시함수 코드를 사용하는 경우 하나의 디바이스가 전송하는 디스커버리 코드 및 어플리케이션 이름의 평균 크기는 다음과 같이 주어진다.

$$\begin{aligned} Q_{hash,variable} &= \delta(\alpha_C+\alpha_N)+s\alpha_C \quad (14)\\ &+s\alpha_N(1-\exp(-0.5^{\delta}(\alpha_C+\alpha_N))). \end{aligned}$$

지금부터는 고정길이 해시함수 코드를 적용한 경우에 대해 분석한다. 하나의 디바이스에서 디스커버리 코드의 길이를 L로 고정시킨다. 이러한 경우 디바이스 1과 디바이스 2에서 하나의 해시코드 길이는 $\lfloor L/(A_C+A_1)\rfloor$ 과 $\lfloor L/(A_C+A_2)\rfloor$ 가 된다. 이 때, 디바이스 2의 디스커버리 코드에 대한 디바이스 1의 어플리케이션의 거짓 양성 확률은

$$1-(1-0.5^{\lfloor L/(A_C+A_2)\rfloor})^{(A_C+A_2)} \quad (15)$$

가 된다.

디바이스 1이나 디바이스 2가 전송하는 어플리케이션 이름의 평균 비트수는

$$s\alpha_C+s\alpha_N(1-\mathrm{E}[(1-0.5^{\lfloor L/x\rfloor})^x]) \quad (16)$$

이며 여기서 x는 평균이 $\alpha_C+\alpha_N$인 포아송 분포를 따르는 확률 변수이다. 그러므로 고정길이 해시함수 코드를 사용하는 경우 하나의 디바이스가 전송하는 디스커버리 코드 및 어플리케이션 이름의 평균 크기는 다음과 같다.

$$\begin{aligned} Q_{hash,fixed} &= L+s\alpha_C \quad (17)\\ &+s\alpha_N(1-\mathrm{E}[(1-0.5^{\lfloor L/x\rfloor})^x]). \end{aligned}$$

4.3. 블룸필터 기반 프로토콜 성능 분석

블룸필터 기반의 디스커버리 코드 적용 시 디스커버리 프로토콜의 성능을 분석한다. 해시함수 기반의 프로토콜과 마찬가지로 가변길이 코드와 고정길이 코드를 사용한 경우로 나누어 분석을 수행한다.

가변길이 코드의 경우 거짓 양성 확률을 고정시키기 위해 어플리케이션의 수에 비례하여 코드 길이를 정한다. 하나의 어플리케이션 마다 δ비트의 코드를 할당하므로 디바이스 1과 디바이스 2의 코드 길이는 각각 $\delta(A_C+A_1)$과 $\delta(A_C+A_2)$으로 주어지며 평균은 $\delta(\alpha_C+\alpha_N)$가 된다.

가변길이 코드를 사용할 때 최적 해시함수의 개수는 $k=\delta\cdot\ln 2$이며 거짓 양성 확률은

$$0.5^{\delta\cdot\ln 2}=\kappa^{\delta} \quad (18)$$

이 된다. 여기서, $\kappa=0.5^{\ln 2}\simeq 0.6185$이다. 디바이스 1이 디바이스 2로부터 코드를 받았을 경우 디바이스 1과 디바이스 2에 공통으로 속해있는 모든 어플리케이션에 대해 어플리케이션 이름을 전송하고 디바이스 1에만 있는 어플리케이션에 대해서는 거짓 양성이 발생하였을 때만 어플리케이션 이름을 전송한다. 그러므로 디바이스 1에서 전송하는 어플리케이션 이름의 평균 길이는

$$s \cdot \mathrm{E}[A_C+A_1\kappa^{\delta}]=s\alpha_C+s\alpha_N\kappa^{\delta} \quad (19)$$

으로 계산되며 이는 디바이스 2에 대해서도 동일한 값을 가진다.

따라서, 가변길이 블룸필터 디스커버리 코드를 사

용하는 프로토콜에서 하나의 디바이스가 전송하는 평균 비트수는 다음과 같다.

$$Q_{bloom,variable} = \delta(\alpha_C+\alpha_N)+s\alpha_C+s\alpha_N\kappa^{\delta}. \quad (20)$$

다음으로 고정길이 블룸필터 디스커버리 코드를 사용하는 프로토콜을 분석한다. 어플리케이션의 수와 관계없이 길이가 L비트로 고정된 블룸필터를 사용한다고 가정한다. 디바이스 1에서의 최적 해시함수의 개수는 $k=(L/(A_C+A_1))\cdot\ln 2$이며 거짓 양성 확률은

$$0.5^{(L/(A_C+A_1))\cdot\ln 2}=\kappa^{L/(A_C+A_1)} \quad (21)$$

이 된다. 그러므로 디바이스 1이 전송하는 어플리케이션 이름 평균 길이는

$$\begin{aligned} &s\cdot \mathrm{E}[A_C+A_1\kappa^{L/(A_C+A_1)}] \\ &=s\alpha_C+s\alpha_N\mathrm{E}[\kappa^{L/x}] \end{aligned} \quad (22)$$

가 되며 x는 평균이 $\alpha_C+\alpha_N$인 포아송 분포를 따르는 확률 변수이다.

고정길이 블룸필터를 사용할 때 하나의 디바이스가 전송하는 평균 비트수는 다음과 같이 계산된다.

$$Q_{bloom,fixed}=L+s\alpha_C+s\alpha_N\mathrm{E}[\kappa^{L/x}]. \quad (23)$$

V. 성능 평가

본 장에서는 가변길이 해시함수 및 블룸필터 기반 디스커버리 코드를 사용한 디스커버리 프로토콜의 성능 분석 결과와 시뮬레이션 결과를 제시한다. α_C=1로 고정시켰으며 α_N이 10, 20, 30인 경우에 대해 각각 시뮬레이션을 수행하였다. 어플리케이션 이름의 평균 길이는 160 비트이다.

그림 5에 디스커버리 프로토콜에서 한 회의 디스커버리 절차를 위해 전송하는 총 비트수를 δ의 함수로 나타내었다. δ가 낮은 영역에서의 블룸필터의 경우 시뮬레이션과 분석결과 사이에 작은 오차가 있으나 분석결과가 전반적으로 매우 정확함을 알 수 있다. 어느 정도 높은 δ값을 사용하면 거짓 양성 오류로 인한

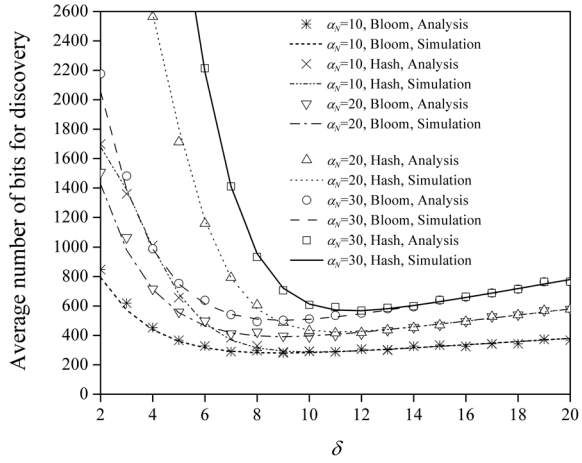

그림 5 해시함수와 블룸필터를 사용하였을 때 디스커버리 프로토콜에서 전송하는 평균 비트수
Fig. 5. Average number of bits for discovery when the hash function and the Bloom filters are used.

어플리케이션 이름의 전송이 거의 사라지는 것을 볼 수 있다. 또한 블룸필터를 사용하는 것이 해시함수보다 좋은 성능을 보임을 알 수 있다.

VI. 결 론

본 논문에서는 기기 간 직접통신 환경에서 주변 디바이스의 어플리케이션을 효율적으로 발견하기 위한 디스커버리 프로토콜을 제안하였다. 각 디바이스는 긴 어플리케이션 이름을 해시함수나 블룸필터를 이용하여 축약한 디스커버리 코드를 주기적으로 전송하여 자신의 어플리케이션을 주변에 알린다. 시뮬레이션과 분석결과를 통해 제안한 프로토콜로 어플리케이션 발견을 위한 정보 전송량을 줄일 수 있음을 확인하였다.

References

[1] G. Fodor, E. Dahlman, G. Mildh, S. Parkvall, N. Reider, G. Mikls, and Z. Turnyi, "Design aspects of network assisted device-to-device communications," *IEEE Commun. Mag.*, vol. 50, no. 3, pp. 170–177, Mar. 2012.

[2] L. Lei, Z. Zhong, C. Lin, and X. Shen, "Operator controlled device-to-device communications in LTE-Advanced networks," *IEEE Wireless Commun. Mag.*, vol. 19, no. 3, pp. 96–104, June 2012.

[3] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications

with Wi-Fi Direct: overview and experimentation," *IEEE Wireless Commun.*, vol. 20, no. 3, pp. 1 - 8, June 2013.

[4] H. Lee, H. H. Choi, S. Jung, S. C. Chang, and D. S. Kwon, "Performance evaluation of device-to-device communications based on system-level simulation in cellular networks," *J. KICS*, vol. 38B, no. 04, pp. 229-239, Apr. 2013.

[5] M. Corson, R. Laroia, J. Li, V. Park, T. Richardson, and G. Tsirtsis, "Toward proximity-aware internetworking," *IEEE Wireless Commun. Mag.*, vol. 17, no. 6, pp. 26 - 33, Dec. 2010.

[6] N. Kayastha, D. Niyato, P. Wang, and E. Hossain, "Applications, architectures, and protocol design issues for mobile social networks: a survey," *Proc. IEEE*, vol. 99, no. 12, pp. 2130 - 2158, Dec. 2011.

[7] T. Petsas, E. P. Markatos, and T. Karagiannis, "Rise of the planet of the apps : a systematic study of the mobile app ecosystem," in *Proc. Internet Measurement Conf. (IMC'13)*, pp. 277 - 290, Barcelona, Spain, Oct. 2013.

[8] W. K. Edwards, "Discovery systems in ubiquitous computing," *IEEE Pervasive Comput.*, vol. 5, no. 2, pp. 70 - 77, Apr. 2006.

[9] UPnP Forum, "UPnP device architecture," Oct. 2008.

[10] *Specification of the Bluetooth system*, 1.C.47/1.0B, Dec. 1999.

[11] IEEE, *Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 9: interworking with external networks*, IEEE Std. 802.11u, 2011.

[12] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," *ACM SIGOPS Operating Syst. Review*, vol. 33, no. 5, pp. 186 - 201, Dec. 1999.

[13] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: a survey," *Internet Math.*, vol. 1, no. 4, pp. 485 - 509, Jan. 2004.

[14] F. Hao, M. Kodialam, and T. V. Lakshman, "Incremental bloom filters," in *Proc. IEEE INFOCOM'08*, pp. 1067 - 1075, Phoenix, U.S.A., Apr. 2008.

최 계 원 (Kae Won Choi)

2001년 2월 서울대학교 지구환경시스템공학부 학사
2007년 8월 서울대학교 컴퓨터공학과 박사
2010년 9월~현재 서울과학기술대학교 컴퓨터공학과 조교수
<관심분야> Device-to-Device Communication, Machine-to-Machine Communication, Cognitive Radio

이 현 (Hyun Lee)
1986년 2월 연세대학교 이과대학 물리학 학사

2012년 2월 충북대학교 정보통신공학과 박사
1994년 3월~현재 한국전자통신연구원 무선분산접속연구실 책임연구원
<관심분야> Device-to-Device Communication, Public Safety and Disaster Relief Communication, Broadband Wireless Access Technology

장 성 철 (Sung Cheol Chang)

1992년 2월 경북대학교 전자공학과 (공학사)
1994년 2월 한국과학기술원 전기 및 전자공학과 (공학석사)
1999년 8월 한국과학기술원 전기 및 전자공학과 (공학박사)
1999년~현재 한국전자통신연구원; 현재 통신인터넷부문 무선분산통신연구실장
<관심분야> 이동통신 무선접속 프로토콜 등