논문 13-38B-05-04

한국통신학회논문지 '13-05 Vol.38B No.05
http://dx.doi.org/10.7840/kics.2013.38B.5.344

모바일 통합 SNS 게이트웨이의 상위 구조 및 MQTT 기반의 푸시 알림 프로토콜 설계

이 신 호◆, 김 현 우*, 주 홍 택°

Design of The High-Level Architecture of Mobile Integration SNS Gateway and The MQTT Based Push Notification Protocol

Shinho Lee◆, Hyeonwoo Kim*, Hongtaek Ju°

요 약

본 논문에서는 스마트폰에 실행되는 여러 종류의 모바일 트래픽 중, 급격히 증가한 SNS 트래픽 해결 방안으로 통합 SNS 게이트웨이 서버에 대한 설계를 제안한다. 현재 모바일 SNS 애플리케이션들은 해당하는 SNS 서버에 각각 대응하여 개별적으로 접근하여 정보를 갱신한다. 우리가 제안하는 통합 SNS 게이트웨이는 이러한 SNS들을 하나로 통합하고, 단말과 SNS 서버 사이에 게이트웨이 서버를 두어 자주 반복되는 데이터 요청에 대하여 트래픽을 줄이고, 모바일 통신환경을 개선할 수 있는 방법을 제안한다. 또한, 단말과 통합 SNS 게이트웨이 서버 간의 푸시 알림 서비스를 위하여 MQTT 프로토콜의 QoS레벨과 페이로드 크기를 다양하게 전달하여 캡처하여 분석하고, 메시지에 대한 종단 간 지연 및 패킷 손실에 대한 분석 결과를 활용하여 MQTT 기반의 통합 SNS 게이트웨이 푸시 알림 서비스를 위한 프로토콜 설계 결과를 제시한다.

Key Words : MQTT, SNS, Gateway, Squid Cache Server, Push Notification Service

ABSTRACT

In this paper, In order to solve the traffic of the SNS to be executed on smartphone, we propose the design of the SNS integrated gateway server. Also, we capture the payload size in a variety of methods to send the MQTT protocol and QoS level of MQTT protocol for communication between the client and the Gateway. And we present the results and analysis of packet loss and end-to-end delay. By using analysis proposed, we present the results of the integrated SNS Gateway push notification protocol design.

Ⅰ. 서 론

오늘날 우리는 스마트 시대에 살고 있다. 다양한 종류의 스마트 기기가 출시되고, 스마트 기기 사용이 급증함에 따라, 모바일 트래픽도 기하급수적으로 증가하고 있다. 특히 모바일 트래픽 중 SNS(Social Networking Service) 트래픽이 급격히 증가하고 있는데[1,2], 이에 대응하기 위한 방법으로 푸시 알림 방식이 활용되고 있다. 모바일 SNS 애플리케이션은 주기적으로 SNS 서버에서 새로운 정보를 가져오는 폴링 방식보다 SNS 서버에서 모바일 SNS 애플리케이션으로 변화된 정보를 보내주는 푸시 방식이

※ 본 연구는 교육과학기술부와 한국연구재단의 지역혁신인력양성사업으로 수행된 연구결과임(2012H1B8A2025942)
◆ 주저자 : 계명대학교 컴퓨터공학과 컴퓨터네트워크 연구실, leeshinho@kmu.ac.kr, 준회원
° 교신저자 : 계명대학교 컴퓨터공학과 컴퓨터네트워크 연구실, juht@kmu.ac.kr, 종신회원
* 계명대학교 컴퓨터공학과 컴퓨터네트워크 연구실, hwkim84@kmu.ac.kr, 준회원
논문번호 : KICS2013-03-122, 접수일자 : 2013년 3월 6일, 최종논문접수일자 : 2013년 5월 9일

효과적이다. 현재 많은 모바일 SNS 애플리케이션들은 이러한 푸시 방식으로 동작하고 있으며 Y. J. Lee 외 3명이 제안한 추상적 푸시 프레임워크[3]와 같이 푸시 방식을 개선하는 다양한 방법들이 제시되고 있다. 푸시 방식에 기초한 SNS 애플리케이션들은 각각의 SNS마다 제공하는 형식[4]이 매우 큰 차이가 있어, 사용자들은 각각의 SNS가 제공하는 여러 개의 SNS 애플리케이션을 사용하고 있다. 그리고 각 SNS 애플리케이션은 SNS 서버에 개별적으로 접근하여 1:1 통신 방식으로 최신 정보를 갱신하여 사용자에게 제공하고 있다. 예를 들어, 푸시 방식에 근거한 Facebook, Twitter를 사용자가 동시에 사용하고 있다고 가정하면, 이 애플리케이션들은 각 SNS 서버에 개별적으로 접근하여 정보를 갱신한다.

이처럼 1:1 통신 방식은 초기 웹이나 FTP처럼 사용자가 각 웹 서버나 FTP 서버에 접속하여 원하는 정보를 가져왔으나, 이후 웹의 사용이 증가함에 따라, 웹의 응답 속도 및 트래픽에 대한 문제가 대두되면서, 웹의 응답 속도를 높이고 트래픽을 줄이는 방법이 제시되었다. J. Y. Kim 외 2명은 웹 캐싱을 이용하여 네트워크 병목현상을 제거하는 방법으로 캐시를 활용하는 방법을 제안[5]하였는데, 사용자 트래픽이 모이는 지점에 웹 캐싱 장치를 설치하고 웹 서버로부터 이미 가져온 정보를 빠르게 사용자에게 제공하는 방법으로 응답시간과 트래픽을 줄일 수 있었다. SNS도 이처럼 중간에 어떤 장치를 두고 통신성능을 개선할 수 있다. 하지만 현재 SNS 애플리케이션의 동작방식은 초기 웹의 방식과 유사하지만, 웹 캐싱과 같이 통신성능을 높이기 위한 방법을 적용하기가 어렵다. 이와 같은 통신성능개선을 가장 어렵게 만드는 요인은 각 SNS의 통신 프로토콜이 표준에 의한 통신 프로토콜이 아니라서 SNS별로 다른 자가 통신 프로토콜을 사용하고 있기 때문이다. 하지만 현재 SNS의 통신 방식에서는 웹 캐싱과 같은 방법으로는 프로토콜 차원에서 통신성능을 개선하기는 불가능하다. 그러나 현재 모바일 SNS 트래픽이 급격히 증가함에 따라, 웹 캐싱과 같이 중간에 통신성능을 개선하기 위한 방법은 필요하다.

본 논문에서는 SNS의 통신성능을 개선하기 위하여 애플리케이션 게이트웨이 방식을 제안한다. 즉, 현재 SNS 통신방식에서 중간에 통신성능을 개선하기 위한 장치가 도입된다면 애플리케이션 게이트웨이인 통합 SNS 게이트웨이가 적합한 방법이다. 통합 SNS 게이트웨이는 통합 SNS 애플리케이션과 각 SNS 서버의 중간에 위치하여 통신성능을 개선하기 위하여 캐싱과 동기화 그리고 통합인증기능을 제공한다. 기존의 SNS는 자가 통신 방식을 사용하므로 프로토콜 변환을 할 수 없지만, 통합 SNS 게이트웨이의 기능을 분산하여 성능을 개선하는 방식이다.

제시하는 통합 SNS 게이트웨이 방식은 2개의 단계를 거쳐서 통신한다. 게이트웨이와 각 SNS 서버 사이에는 기존의 SNS 자가 프로토콜을 사용하며, 게이트웨이와 통합 SNS 애플리케이션 사이에는 새롭게 고안된 통합 SNS 프로토콜을 사용한다. 이 과정에서 게이트웨이는 통신성능을 개선하기 위한 기능을 수행한다. 통합 SNS 애플리케이션은 기존의 SNS 애플리케이션들에서 제공하는 기능을 하나의 애플리케이션으로 통합하여 제공한다. 통합 SNS 애플리케이션을 별도로 설치하여야 하는 단점과 통합 SNS에서 제공하는 기능이 각 SNS에서 제공하는 기능과 다를 수 있다는 단점이 있지만, 통신성능을 개선하는 긍정적인 효과가 있다. 본 논문에서는 이와 같은 통합 SNS 게이트웨이에 대한 구조 설계와 통합 SNS 게이트웨이와 통합 SNS 애플리케이션 간의 푸시 알림 서비스를 위한 프로토콜 설계 결과를 제시한다.

통합 SNS 게이트웨이는 캐싱, 동기화 기능과 통합인증 기능, 그리고 각 SNS 서버와 통신하기 위한 어댑터가 핵심 요소이며 이들을 유기적으로 연결하기 위한 방법이 제시된다. 통합 SNS 프로토콜은 푸시 알림 서비스 방식의 통신방식에 기초하고 있으며 IBM에서 발표한 오픈 프로토콜인 MQTT(Message Queuing Telemetry Transport)[6] 프로토콜을 사용하였다. MQTT는 초창기 제한된 환경에서의 낮은 대역폭 또는 높은 지연이나, 신뢰할 수 없는 네트워크를 위하여 설계된 프로토콜로 모바일 푸시 알림 서비스에 사용하기 적합한 프로토콜이다.

본 논문의 구성은 다음과 같다. 2장에서는 통합 SNS 게이트웨이 설계를 위한 기존 연구에 대하여 설명한다. 3장에서는 통합 SNS 게이트웨이에 대한 개념과 설계에 대하여 설명한다. 4장에서는 단말과 통합 SNS 게이트웨이 간의 푸시 알림 서비스를 위하여 활용할 MQTT에 대한 성능 분석과 결과를 제시한다. 또한, MQTT의 성능 분석을 한 결과를 바탕으로 MQTT 기반의 통합 SNS 게이트웨이 푸시 알림 프로토콜 설계 결과를 제시한다. 마지막으로,

5장에서는 결론 및 향후 연구에 대하여 설명하고 본 논문을 마무리한다.

Ⅱ. 기존 연구

컴퓨터 네트워크에서 게이트웨이는 서로 다른 프로토콜 사이의 변환을 담당하는 장치이다. 예를 들어, 비슷한 기능을 제공하지만 새롭게 개선된 프로토콜이 나오면 기존의 프로토콜을 수용하기 위하여 프로토콜 변환을 담당하는 게이트웨이를 도입한다. 이처럼 프로토콜 수준에서 동작하는 게이트웨이는 서로 다른 프로토콜이나 영역을 접목해 주는 역할을 담당한다. 애플리케이션 게이트웨이[7]는 인증이나 통신 기능을 통합하여 제공한다. 다시 말해서, 웹 프록시 서버나, 여러 개의 단말에서 요구되는 인증을 하나로 모아서 높은 수준의 인증을 담당하는 것이 애플리케이션 게이트웨이이다. 애플리케이션 게이트웨이라는 용어는 NAT(Network Address Translation)에서도 사용되나 그 의미는 프로토콜 게이트웨이가 적합하다. 애플리케이션 게이트웨이는 주로 통신 성능을 향상하기 위해서 사용된다. 본 논문에서는 기능적인 통합을 통하여 통신성능을 향상하는 애플리케이션 게이트웨이로 SNS에 적용하였다.

SNS 통신 방식은 크게 푸시 알림 방식과 콘텐츠 로드 방식으로 나뉜다.

푸시 알림 방식은 일정 주기마다 통신이 끊어지지 않게 Keep Alive 메시지를 단말에 전송하여 서비스를 이용할 수 있는지 확인하다가, 서버에서 이벤트가 발생하면 해당 단말에 메시지를 전송하는 통신 방법이다. 아래의 그림 1은 푸시 알림 방식을 유지하기 위하여 주기적인 푸시 알림 Flag의 송/수신에 대하여 설명하는 그림이다.

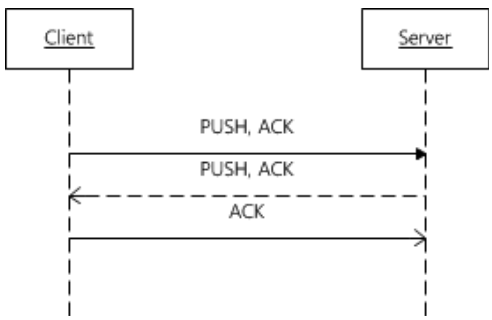

그림 1. 푸시 알림 방식 데이터 처리 과정
Fig. 1. Push Notification Service Packet Process

콘텐츠 로드 방식은 단말에서 필요한 콘텐츠를 서버로 요청하게 되면, 해당 콘텐츠를 단말로 전송하여 응답하는 방식이다. 우리가 일반적으로 단말에서 서버에 해당 콘텐츠를 요청할 때, 서버 측에서 단말에게 요청한 콘텐츠를 보내는 것과 동일한 통신방식이다. 그림 2는 콘텐츠 로드 방식에 대한 통신을 설명하는 그림이다.

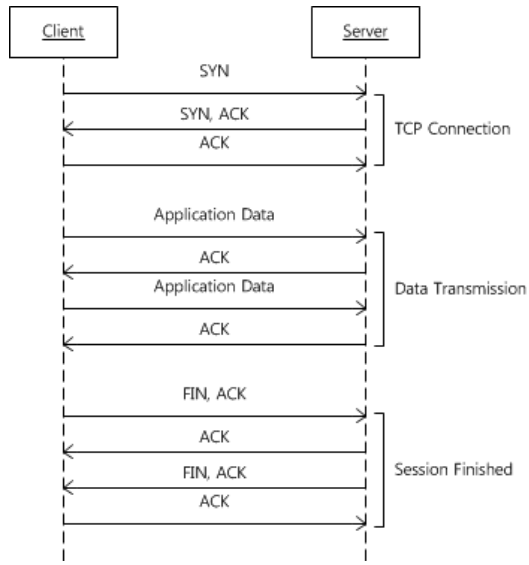

그림 2. 콘텐츠 로드 방식 데이터 처리 과정
Fig. 2. Contents Request Packet Process

MQTT는 서로 다른 OS(Operating System) 환경이나 SNS의 프로토콜에서 통합 SNS 게이트웨이를 구현하는데 적합한 프로토콜이다. MQTT는 푸시 알림 서비스를 구현함에 있어 벤더 서비스 업체의 의존성과 관계없이 개인적인 푸시 알림 서비스를 구축할 수 있는 장점과 메시징에 대한 제약사항이 없다는 장점을 지니고 있다. 또한, MQTT는 기존의 HTTP 프로토콜보다 배터리를 효율적으로 사용할 수 있다는 것이 검증되었다[8-9]. 현재 MQTT는 대표적으로 Facebook Messenger에서 채택하여 활용되고 있으며[10], MQTT를 이용하여 많은 오픈 소스 프로젝트들이 진행되고 있고, 다양한 실험을 통하여 그 유용성이 검증되었다[11-12]. Perez, Julio[13]는 OMNeT++ 시뮬레이션 환경에서 MQTT 프로토콜에 대한 시뮬레이션 환경에서의 메시지의 개수와 여러 클라이언트 간의 성능을 분석하였다. OMNeT++[14]는 오픈 소스 프로젝트로써 Network Simulation을 위한 도구이다.

MQTT는 기본적으로 1개의 Broker 서버와 2개의 Publish, Subscribe 클라이언트[15]로 구성된다.

Broker 서버는 2개의 Publish, Subscribe 클라이언트 사이에서 주어지는 Topic에 대한 메시지 전달의 중개자 역할을 한다. Publish 클라이언트가 Topic을 발행하고 메시지를 Broker 서버로 전달하면, Subscribe 클라이언트는 관심 있는 Topic을 구독하게 된다.

MQTT 는 메시지 처리에 대한 신뢰성 보장을 위하여 3단계의 QoS(Quality of Service)[16]를 지원한다. 그림 3은 QoS 레벨에 따른 패킷 교환 방법을 설명한다.

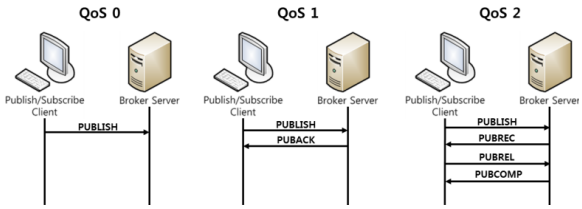

그림 3. QoS 레벨에 따른 패킷 교환 방법
Fig. 3. Packet transmission method about QoS Level

QoS 0은 메시지를 한 번만 전달하고 전달 여부는 확인하지 않는다. 따라서 큰 페이로드를 가지는 메시지일 경우, 패킷 손실이 발생하면 메시지가 전달되지 않고 유실될 가능성이 높다. QoS 1은 메시지를 최소 1번 이상 전달하고 전달 여부를 확인한다. 하지만 PUBACK 패킷이 유실된다면 메시지가 불필요하게 중복으로 전달될 가능성이 있다. QoS 2는 4-way handshake을 통해 정확하게 한 번만 전달한다. QoS 2를 사용한다면 메시지가 유실될 가능성은 없지만 4-way handshake 과정에 의하여 종단 간 지연이 늘어나게 된다. QoS 레벨이 높을수록, 패킷을 교환하는 횟수가 늘어나기 때문에 상대적으로 종단 간 지연은 늘어나게 된다.

Ⅲ. 통합 SNS 게이트웨이 설계

이 장에서 우리는 모바일 SNS를 위한 통합 SNS 게이트웨이의 개념과 통합 SNS 게이트웨이 설계에 대하여 설명한다.

3.1. 모바일 통합 SNS 게이트웨이

모바일 통합 SNS 게이트웨이는 사용자 단말에 설치되는 모바일 통합 SNS 애플리케이션과 기존의 SNS를 제공하는 각 SNS 서버 사이에 위치한다. 그림 4는 현재의 일반 SNS 애플리케이션에서 SNS별로 통신하는 방식과 통합 SNS 게이트웨이를 사용한 방식을 비교한 그림이다.

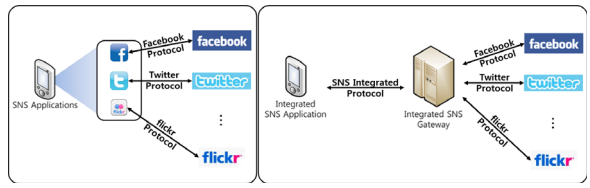

그림 4. 현재의 방식과 게이트웨이를 사용한 방식 비교
Fig. 4. Contents Request Packet Process

본 논문에서 제안하는 게이트웨이는 사용자 정보를 저장하고, 통합 SNS 애플리케이션의 요청에 의해 데이터를 저장하거나 제공하며, SNS 서버에 직접 접근하여 사용자의 SNS 정보를 갱신하고 최신 정보를 가져와서 통합 SNS 애플리케이션에 제공한다. 이 게이트웨이는 다수 사용자 접속에 따른 계정 관리 및 보안처리 기능도 수행한다. 사용자 계정 정보를 이용하여 주기적으로 SNS에 접근하여 정보를 갱신하고 최근 정보를 입수한다. 이 과정에서 게이트웨이는 통합 SNS 애플리케이션의 부하를 최소화하고 통신 성능을 향상하도록 기능을 수행한다.

사용자 계정 정보를 이용하여 주기적으로 SNS에 대한 데이터를 임시 저장하는 캐시 기능과 변경된 데이터만을 전송하는 동기화를 기능을 통하여 통신 성능을 향상시킨다. 본 논문에서는 이후로 통신성능 향상에 관해서만 다루기 때문에 캐시 기능과 동기화 기능에 대하여 주로 언급하고 기술적인 내용 전달에 지장을 초래하지 않는 범위에서 계정관리나 보안기능에 대하여는 최소로 언급한다.

통합 SNS 게이트웨이의 캐싱 기능은 응답속도를 향상시키고 네트워크 트래픽을 감소시킨다. 게이트웨이는 통합 SNS 애플리케이션이 요청한 최근의 전송한 내용을 복사하여 저장하고 있다가, 이후에 SNS 애플리케이션에서 동일한 내용을 요청하면 SNS 서버에서 데이터를 받아 오지 않고 저장된 데이터를 곧바로 보낸다. 이 방식은 전통적인 인터넷 캐싱과 동일하며 SNS 서버의 부하 및 트래픽, 응답시간을 줄여준다. 이 방법이 SNS 게이트웨이에서 가능하기 위해서는 SNS로 제공되는 객체에 대하여 동일한 데이터인지를 식별할 수 있는 방법이 도입되어야 한다. 웹의 경우 웹 객체별로 URL이 바로 이러한 역할을 한다. 또한, 각 객체에 대하여 객체의 일정 시간 동안의 유효성을 계산할 수 있어야 한다. 웹의 경우 HTTP 프로토콜에서 객체의 수명이 전달되며 이를 사용하고 있다.

우리는 다른 연구에서 이미 Facebook을 중심으로 SNS 프로토콜을 분석하는 연구를 실시하였다[17]. 이 연구에서는 공개되지 않는 SNS 프로토콜을 분석하는 방법을 제시하고 SNS 프로토콜을 분석한 결과를 제시하였다. 연구결과 SNS 프로토콜은 HTTP 프로토콜을 기반으로 만들어져 있다. 따라서 기본적으로 기존의 웹 캐싱 장치를 사용하여도 통신성능의 향상이 있다. 우리가 제안하는 통합 SNS 게이트웨이에서는 기존의 웹 캐싱 기능을 활용한다. 다만 통합 SNS 애플리케이션에 제공하기 위해서는 기존의 웹 캐싱 결과를 통합 프로토콜로 변환하여 제공한다. 이미 많은 SNS 데이터에 사진, 동영상 등 멀티미디어 데이터가 포함되어 있기 때문에 이 웹 캐싱 기능은 효과가 적지 않다.

또한, 기존 웹 캐싱 기능 활용과 함께 SNS 프로토콜 분석결과를 기반으로 캐싱 가능한 객체에 대하여 캐싱이 가능하다. 예를 들어, 기존 SNS는 사진을 자신의 서버에 올리면 결과를 다시 전송받아서 자신의 페이지를 완성한다. 이러한 방식은 사진을 올리고 다시 내려받는 과정을 통해 불필요한 트래픽을 유발하지만, 게이트웨이는 이 불필요한 트래픽을 절감한다. 본 논문에서는 이와 같은 기존 SNS 프로토콜의 분석 결과를 활용한 캐싱 방법에 대해서는 자세히 다루지 않으며 현재 연구가 진행 중이고 다른 논문으로 발표할 예정이다.

캐싱 기능과 함께 통합 SNS 게이트웨이에서 통신성능을 개선하기 위한 방법으로 동기화 기능의 활용이 있다. 동기화 기능은 서로 다른 위치에 있는 장치 또는 프로그램 간에 동일한 데이터를 유지하기 위한 방법으로 서로 변경된 부분만을 주고받는다. 동기화 기능은 개인 단말기의 연락처 및 일정정보 등과 같은 정보들을 컴퓨터나 서버의 정보들과 일치시키기 위해서 주로 사용되었다. 예를 들어, Facebook의 경우 서버에 접속하면 친구리스트를 모두 다시 갱신한다. 하지만 동기화 기능을 활용하면 기존의 친구리스트에서 변화된 부분만을 전송하여 일치시키므로, 따라서 응답속도를 높이고 트래픽을 줄일 수 있다. 그러나 기존의 SNS는 동기화 기능을 활용하지 않고 있다. 동기화 기능은 동일한 데이터를 유지하기 위해서 차이점을 고려하지 않고 모든 데이터를 갱신하는 방법보다 개선된 방법으로 트래픽과 응답속도를 향상시킬 수 있다. 우리가 제안하는 통합 SNS 게이트웨이에서는 이 기능을 활용하여 게이트웨이와 각 SNS 서버 사이에서 기존의 방식대로 통신하여 최근 정보를 유지하며, 최근 정보는 기존의 정보와 비교하여 변경된 정보를 계산하고, 변경된 정보만을 통합 SNS 애플리케이션에 전달한다. 또한, 우리는 각 SNS 서버별로 동기화 기능으로 성능을 향상시킬 수 있는 정보를 정의하고 동기화 객체를 객체 트리로 구성하여 동기화 영역을 쉽게 계산할 수 있게 하였다. 동기화를 위한 객체정보 구성방법은 본 논문에서 자세히 다루지 않는다.

3.2. 모바일 통합 SNS 게이트웨이 설계

이 절에서는 앞 절에서 기술한 통합 SNS 게이트웨이에 대한 설계결과를 제시한다. 그림 5는 통합 SNS 게이트웨이의 구성도를 나타낸다.

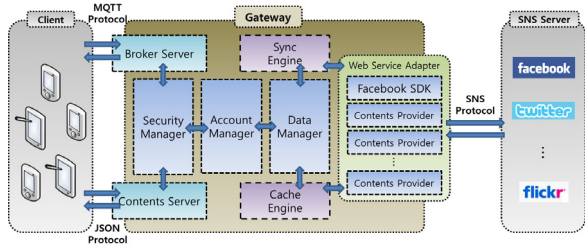

그림 5. 통합 SNS 게이트웨이 서버 구성도
Fig. 5. SNS Integration Gateway Server Architecture

통합 SNS 게이트웨이는 각 SNS 서비스별로 사용하는 독특한 프로토콜을 사용하여 각 SNS 서버에 접속하여야 한다. 이러한 기능을 제공하는 것이 Web Service Adapter이다. Web Service Adapter는 SNS별로 개발되어야 하며 각 SNS에서 제공하는 개발 플랫폼을 사용하여 개발한다. 즉, Facebook Web Service Adapter는 Facebook에서 제공하는 Facebook SDK를 사용하여 개발한다.

또한, 통합 SNS 게이트웨이는 Security Manager, Account Manager 그리고 Data Manager는 보안기능, 계정관리기능, 데이터관리 기능을 제공한다.

통합 SNS 게이트웨이의 캐싱은 오픈 소스 프로젝트로 개발된 Squid 캐시 서버[18]를 이용한다. Squid 서버는 캐시 및 프록시를 위한 서버이다. 앞서 설명한 바와 같이 기존의 SNS 프로토콜이 HTTP 기반으로 만들어져 있으므로 웹 캐싱 기능을 기본으로 활용할 수 있으며, 따라서 우리는 Squid 서버의 기능을 활용한다. 또한, Squid 캐시 서버에 캐싱 여부를 판단하는 부분을 개선하여 통합 SNS 게이트웨이만의 효과적인 캐싱이 가능하도록 개선한다.

동기화 기능은 오픈소스 프로젝트로 개발되는 Funambol 동기화 엔진[19]을 활용한다. Funambol 동

기화 엔진은 동기화 표준 프로토콜인 SyncML 프로토콜에 기반하여 개발되었다. 이 엔진은 동기화를 위한 객체에 따라서 별도로 어댑터를 개발할 수 있는 방법을 제공하고 있다. 통합 SNS 게이트웨이는 SNS 객체에 대하여 객체 트리를 구성하고 이 트리를 기반으로 변화된 정보를 계산할 수 있도록 개선하였다.

통합 SNS 애플리케이션과 게이트웨이 간의 푸시 기반 통신은 오픈 소스 프로젝트로 개발된 MQTT Broker 서버인 Mosquitto[20]를 활용한다. MQTT는 푸시 기반의 전송방식만을 제공하므로 이를 기반으로 통합 SNS 푸시 알림 서비스 프로토콜을 설계하여 이용한다. MQTT 기반의 통합 SNS 프로토콜 설계 결과는 다음 절에서 제시한다.

통합 SNS 애플리케이션에서 푸시 알림을 받게 되면 사용자는 단말을 통하여 해당 푸시 알림을 받은 SNS 정보를 게이트웨이에 요청하게 된다. 우리는 이러한 콘텐츠 로드 방식 구현하기 위하여 JSON(JavaScript Object Notation)[21]을 활용한다. JSON은 경량의 데이터 교환 포맷으로 데이터 타입이 자유로우며 읽고 쓰기가 용이하고, XML보다 데이터 형식이 간단하게 표현 가능하다[22].

각 기능은 게이트웨이로서 동작하기 위해서 자체적으로 정의된 세부 기능에 따라 개발된 구성요소로서 지면의 한계로 자세히 다루지 않는다.

아래의 그림 6은 하나의 페이지를 모바일 애플리케이션에서 갱신하는 동안 통합 SNS 게이트웨이 서버의 각 구성요소 간의 동작 과정을 나타낸다.

단말에서 게이트웨이 서버에 페이지에 대해 요청을 하게 되면 게이트웨이 서버에서는 요청한 콘텐츠에 대하여 캐시와 동기화 여부를 확인한다.

만약 요청 콘텐츠가 동기화 콘텐츠이면 해당 콘텐츠의 새롭게 동기화 된 정보를 확인한 후, 이를 전송하게 된다.

만약 요청 콘텐츠가 캐시 콘텐츠이면 해당 콘텐츠에 대하여 캐시가 되어 있는지 확인한다. 해당 콘텐츠가 캐시 되어 있으면, 게이트웨이 서버에서 해당 단말에 대한 응답을 한다. 만약 해당 콘텐츠가 캐시 되어 있지 않으면, Web Service Adapter에서 해당 실제 SNS 서버에 콘텐츠를 요청하여 캐싱한다. 단말에서 자주 요청되는 콘텐츠를 캐싱한다면, 실제 SNS 서버에 도달하지 않고 게이트웨이 서버에서 단말에게 콘텐츠를 제공하게 되므로 트래픽과 응답 시간을 줄일 수 있다.

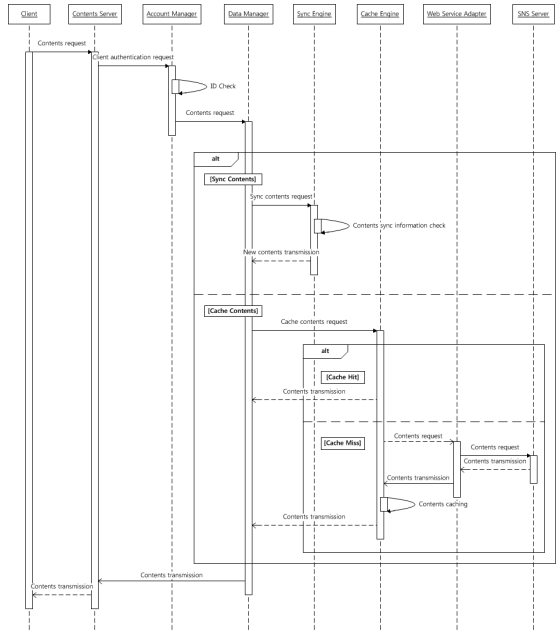

그림 6. 통합 SNS 게이트웨이 순차 다이어그램
Fig. 6. Integration SNS Gateway Sequence Diagram

Ⅳ. 통합 SNS 푸시 알림 서비스 프로토콜 설계

통합 SNS 푸시 알림 서비스를 위한 MQTT 기반의 프로토콜을 설계한 결과를 본 장에서 제시한다. 푸시 알림 서비스를 위하여 프로토콜 설계 결과는 다음과 같이 3가지 사항을 규정해야 한다. 첫째는 전달할 메시지의 Topic Tree, 둘째는 QoS 수준, 셋째는 메시지의 길이이다. 첫째 Topic Tree는 SNS별로 구성되어야 하므로 이 논문에서는 다루지 않는다. 다음 첫째 절에서는 QoS 수준과 메시지 길이를 결정하기 위한 실험 환경과 측정방법을 설명하고 두 번째 절에서는 실험결과와 이를 바탕으로 도출된 결정사항을 제시한다. 재전송시간은 전달해야 할 메시지의 내용을 전송 프로토콜로서 전달하고자 하는 Publish 클라이언트에서 메시지가 정확하게 Subscribe 클라이언트까지 전달하기 위해서는 높은 QoS 레벨을 사용하는 것이 좋은 방법이다. 하지만 이 경우 그만큼 종단 간 지연이 늘어나게 된다. 만약 페이로드 크기에 대한 패킷 손실과 종단 간 지연에 대한 결과를 분석하여 페이로드 크기에 따른 적절한 QoS 레벨을 도출할 수 있다면, MQTT를 사용함에 있어서 더욱 효과적인 푸시 알림 서비스 네트워크 환경을 구성할 수 있을 것이다. 마지막으로, 셋째 절에서는 분석된 결과를 바탕으로 MQTT 기반의 푸시 알림 프로토콜을 설계한 결과를 제시한다.

4.1. 게이트웨이 및 단말 간의 통신 실험 환경

이 절에서 우리는 실제 네트워크 환경에서 단말과 게이트웨이 서버 사이에서 MQTT의 QoS 레벨과 페이로드 크기에 따른 종단 간 지연 및 패킷 손실의 성능 분석을 위한 네트워크 환경 및 패킷 측정 방법에 대하여 설명한다. 실험을 위한 서버의 OS는 Linux 환경의 CentOS 6.3 Final이며, Broker 서버 소프트웨어는 오픈 소스 프로젝트인 Mosquitto를 활용하였다.

MQTT는 Publish 클라이언트와 Subscribe 클라이언트 사이의 Broker 서버를 통하여 메시지를 전달한다. 그림 7은 각각의 클라이언트들이 Broker 서버를 통하여 해당하는 Topic에 메시지를 전달하는 과정을 보여준다.

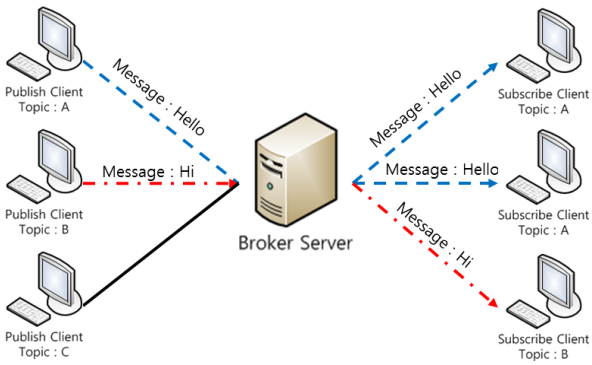

그림 7. MQTT 메시지 전달 과정
Fig. 7. MQTT Message Transmission Process

Publish 클라이언트는 발행하고자 하는 Topic을 Broker 서버 측에게 알린다. Broker 서버는 발행하는 Publish 클라이언트의 Topic과 구독하는 Subscribe 클라이언트의 Topic이 동일한 Topic일 경우, 해당 Topic을 기준으로 클라이언트 간 메시지 전달의 중개자 역할을 하게 된다. Subscribe 클라이언트는 Broker 서버에게 구독 요청을 한 Topic에 한하여 메시지를 전달받는다. 만약 Publish 클라이언트가 'A'라는 Topic을 Broker 서버에게 전달하게 되고, 다수의 Subscribe 클라이언트가 'A'라는 Topic으로 구독을 요청하게 된다면, 'A' Topic으로 구독을 요청한 해당 Subscribe 클라이언트들에 한하여, Publish 클라이언트의 메시지를 전달받게 된다.

실제 네트워크와 유사한 환경에서의 실험을 위하여 클라이언트와 게이트웨이 간의 통신환경을 아래 그림 8과 같이 구현하였다. 우리는 실제 네트워크와 유사한 환경에서 실험을 진행하기 위하여, 모바일 기기에서 3G망을 거쳐 Broker 서버로 통신한다. 3G망을 거쳐서 실험 환경을 구축함으로써, 실제 무선 네트워크 환경에서의 실험과 동일하다고 볼 수 있다. 실험은 Android 2.3 환경에서 IA92 라이브러리[23]를 이용한 애플리케이션 클라이언트에서 실험을 진행하였다. 실험 대상 기기의 MTU(Maximum Transmission Unit)[24]는 1,500으로 설정되어 있다.

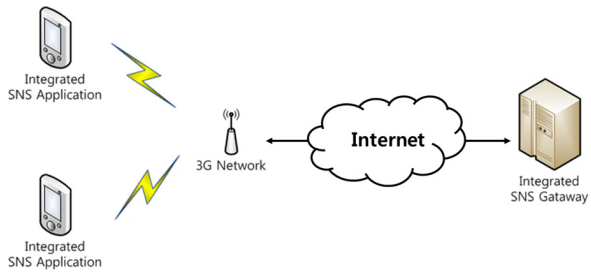

그림 8. 실제 네트워크 실험 구성
Fig. 8. Actual Network Configuration

우리는 패킷을 수집하기 위하여 Shark 애플리케이션을 이용하였다. 모바일 환경은 패킷을 캡처하기 위해서 제한된 환경이므로 Shark 애플리케이션을 통하여 패킷을 캡처하고 저장되는 Pcap 파일을 Wireshark로 패킷 분석을 한다. 5분의 측정 시간 동안 서버와 클라이언트 사이의 패킷을 수집하여 종단 간 지연과 패킷 손실을 분석한다. 종단 간 지연 측정은 Publish 클라이언트에서 게이트웨이 서버를 지나서 Subscribe 클라이언트까지의 타임스탬프를 이용하여 측정한다. 패킷 손실 측정 방법은 5분 동안 측정한 패킷들의 재송신 요청 패킷들을 카운트하여 측정한다.

4.2. 실험 결과와 통신 파라미터 설정

이 절에서 우리는 위의 실험을 토대로 패킷에 대한 종단 간 지연과 패킷 손실에 대해 분석을 하고, QoS 레벨과 페이로드에 따른 종단 간 지연과 패킷 손실에 대하여 결과를 제시한다. 또한, 이 결과를 바탕으로 통합 SNS 프로토콜로 사용하기 위한 설정 값을 제시한다.

아래의 그림 9는 게이트웨이 서버와 단말 간의 QoS 레벨과 페이로드에 대한 종단 간 지연 결과를 비교하는 그래프이다. 전반적으로 QoS 레벨이 증가함에 따라 평균 종단 간 지연이 증가한 것을 볼 수 있다. 그 이유는 QoS에 따른 패킷의 송/수신 개수의 차이 때문이다. 평균적으로 큰 종단 간 지연 현상을 보여주는데, 이는 무선 네트워크 3G망의 전송 속도가 지연 현상의 원인이 된다. 페이로드의 크기가 4,000에서 증가 폭이 상당히 급격해진다. QoS 0

의 경우는 4,000에서 0.65에서 0.61로 오히려 감소하였고, QoS 1은 0.57로 그대로 유지하고 있으며, QoS는 0.47로 페이로드 길이가 1,000일 경우와 같은 값이다. 그러나 페이로드 길이 4,000을 기점으로 모든 QoS 수준에서 전달지연이 최소 0.15 이상씩 증가하였다. 한 가지 더 주목할 점은 QoS 별로 전달지연이 크지 않다는 점이다. 결론적으로 전달지연에 관해서는 QoS 수준은 고려하지 않아도 되며 페이로드 크기는 4,000 이하로 하는 것이 좋다는 결론이다.

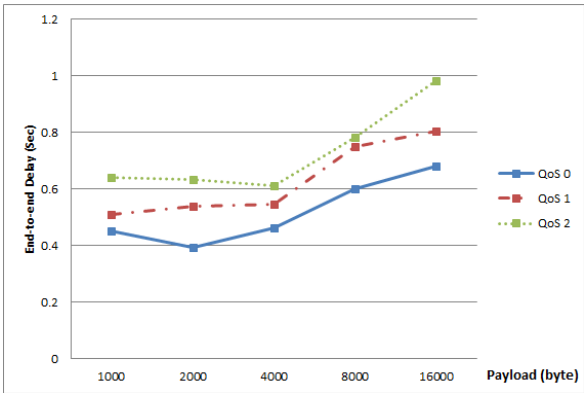

그림 9. 평균 종단 간 지연 분석결과
Fig. 9. Mean End-to-End Delay analysis result

아래 그림 10은 게이트웨이 서버와 단말 간의 패킷 손실률을 분석한 결과이다. 실험 결과 전체적으로 QoS 0, QoS 1과 비교하면 QoS 2의 패킷 손실이 낮은 것을 확인할 수 있다. 전체적으로 패킷 손실률은 안정적이지만, QoS 0이나 QoS 1에 비교해 상대적으로 QoS 2의 패킷 손실이 낮다. 그 이유는 4-way handshake를 이용하여 메시지를 전달하므로 손실률이 줄어들었기 때문이다.

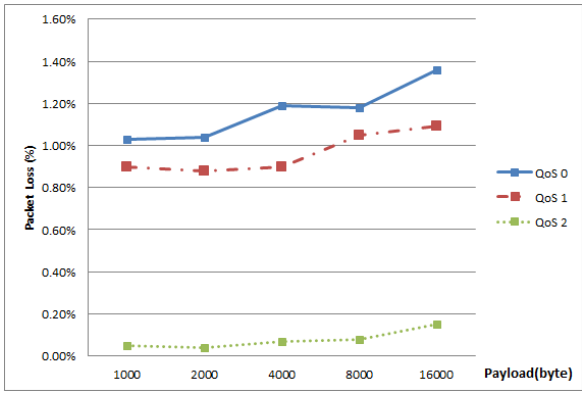

그림 10. 패킷 손실 분석결과
Fig. 10. Packet loss analysis result

4.3. 푸시 알림 서비스 프로토콜 설계

우리는 MQTT의 페이로드 부분을 활용하여 각각의 SNS의 기능별 푸시 알림 서비스를 구분할 수 있도록 새로운 통합 SNS 게이트웨이 프로토콜을 설계하였다. 그림 11은 통합 SNS 게이트웨이의 푸시 알림 서비스를 위하여 프로토콜을 설계한 결과이다.

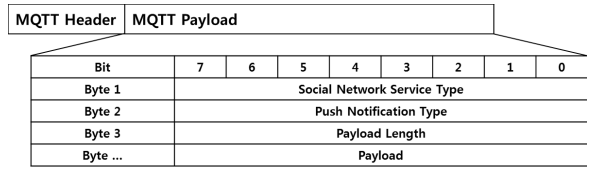

그림 11. 푸시 알림 서비스 프로토콜 설계
Fig. 11. Push Notification Service protocol design

통합 SNS 게이트웨이의 푸시 알림을 해주기 위해서는 SNS에 대한 정보와 기능별 푸시 알림 기능이 필요하다. 우리는 그림 11과 같이 필드를 구분하고, MQTT의 페이로드 부분을 활용하여, 프로토콜을 설계하였다. 앞서 제시한 실험 결과를 토대로 푸시 알림의 메시지 크기는 4,000 이하로 제한한다. 그림 12는 통합 SNS 애플리케이션과 게이트웨이 사이에 설계된 프로토콜을 활용하여 푸시 알림 서비스에 대한 과정을 나타내는 시퀀스 다이어그램이다.

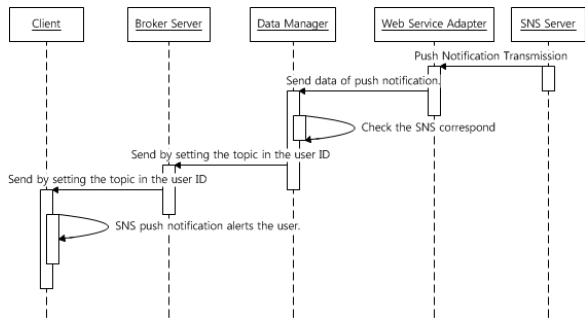

그림 12. 단말-게이트웨이 푸시 알림 전송 다이어그램
Fig. 12. Push Notification Process

SNS 서버에서 해당 사용자의 푸시 알림을 통합 SNS 게이트웨이에 송신하여 게이트웨이가 수신 받게 되면, 게이트웨이의 Web Service Adapter 모듈은 해당 푸시 알림을 Data Manager 모듈에 푸시 알림 데이터를 보내게 된다. Data Manager는 받은 푸시 알림 데이터와 해당 사용자의 ID를 조합하여 Topic을 생성하고, 생성된 Topic과 우리가 설계한 프로토콜을 Broker 서버에게 전송한다. Broker 서버는 Subscribe로 등록된 단말을 확인하고 Data Manager 모듈에서

수신받은 통합 SNS 푸시 알림을 Topic에 해당하는 단말에 송신하게 된다. 단말이 게이트웨이의 푸시 알림을 수신받게 되면, 해당 단말은 푸시 알림 데이터의 프로토콜을 파싱하여 SNS 정보와 푸시 알림의 메시지를 사용자에게 알리게 된다.

Ⅴ. 결론 및 향후 연구

본 논문에서 우리는 폭발적으로 증가하는 SNS 트래픽에 대응하기 위하여, 통합 SNS 게이트웨이를 활용한 통신성능 개선 방안을 제시하였다.

제시한 방안을 실현하기 위한 통합 SNS 게이트웨이의 상위 설계 결과를 제시하였으며, 게이트웨이 서버와 단말의 푸시 알림 서비스를 위하여 MQTT 기반의 프로토콜 설계 결과도 제시하였다. 통합 SNS 게이트웨이는 캐싱 기능과 동기화 기능을 활용하여 통신성능을 개선할 방법과 각 기능을 구현하기 위해 기존의 구현 결과를 최대한 활용하는 방법으로 설계되었다. 통합 SNS 푸시 알림을 위한 프로토콜은 MQTT를 기반으로 설계되었으며, 실제 네트워크 환경에서 성능을 측정하고 분석한 결과를 설계에 반영하여 적용하였다.

우리가 제안한 통합 SNS 게이트웨이의 장단점을 종합하여 표 1로 정리하였다.

표 1. 통합 SNS 게이트웨이의 장점 및 단점
Table 1. Advantages and disadvantages of integration SNS gateway

Advantage	· To use the synchronization function between the cache, reducing traffic and increase the content request / response speed. · Use the Web Service Adapter, it can respond flexibly to new SNS. · Provides services to integrate existing SNS.
Weakness	· Be installed separately integrated SNS application. · The functionality to the integrate SNS application provides, it may be different from the functionality provided by each SNS.

현재 제시된 설계 결과를 바탕으로 구현을 진행중이며, 구현 완료 후에 통합 SNS 게이트웨이의 트래픽에 대한 통신 성능 개선 결과 확인을 위한 실험을 수행할 계획이다. 실험결과로 게이트웨이 서버와 일반 SNS 실제 서버와의 응답시간 및 트래픽 사용량을 측정하고 분석하여 통합 SNS 게이트웨이에 대한 성능개선 효과를 평가할 것이다.

References

[1] Y. R. Choi, J. Y. Chung, B. C. Park, and J. W. K. Hong, "A study on system architecture for application-level mobile traffic monitoring and analysis," *KNOM Review*, vol. 14, no. 2, pp. 10-21, Dec. 2011.

[2] H. Min and M. S. Kim, "Towards smart phone traffic classification," in *Proc. 14th Asia-Pacific Network Operations and Manage. Symp. (APNOMS),* pp. 1-4, Seoul, Korea, Sep. 2012.

[3] Y. J. Lee, J. S. Oh, and B. G. Lee, "Logical push framework for real-time SNS processing," in *Proc. 4th Int. Conf. Computational Aspects of Social Networks (CASoN)*, pp. 47-51, São Carlos, Brazil, Nov. 2012.

[4] P. Y. Kim, S. S. Moon, and H. Y. Youn, "A study on social network service characteristic according to communication type," in *Proc. KICS Int. Conf. Commun. (KICS ICC 2010)*, pp. 639-640, Jeju Island, Korea, Jun. 2010.

[5] J. Y. Kim, K. W. Cho, and K. Koh, "A proxy server structure and its cache consistency mechanism at the network bottleneck," in *Proc. 23rd Annu. Int. Computer Software Applicat. Conf. (COMPSAC '99).* pp. 278-283, Phoenix, U.S.A., Oct. 1999.

[6] IBM, *The MQTT protocol*, Retrieved Aug., 20, 2012, from http://www.mqtt.org.

[7] IPv6.com, *ALG - Application Level Gateway*, Retrieved July, 15, 2012, from http://ipv6.com/articles/gateways/Application-Level-Gateway.htm.

[8] M. Prihodko, "Energy consumption in location sharing protocols for Android applications," M.S. Thesis, Dept. Comput. Inform. Sci., Software, Syst., Linköpings University, Oct. 2012.

[9] S. Nicholas, *Power Profiling: HTTPS Long*

Polling vs. MQTT with SSL, on Android(2012), Retrieved Oct., 15, 2012, from http://stephendnicholas.com/archives/1217.

[10] L. Zhang, *Building Facebook Messenger(2011)*, Retrieved Aug., 13, 2012, from http://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920.

[11] M. Collina, G. E. Corazza, and A. Vanelli-Coralli, "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST," in *Proc. IEEE 23rd Int. Symp. Personal Indoor and Mobile Radio Commun. (PIMRC 2012),* pp. 36-41, Sydney, Australia, Sep. 2012.

[12] M. Ma, Y. Huang, C. H. Chu, and P. Wang, "User-driven cloud transportation system for smart driving," in *Proc. IEEE 4th Int. Conf. Cloud Comput. Technol. Sci. (CloudCom),* pp 658-665, Taipei, Taiwan, Dec. 2012.

[13] P. Julio, "MQTT Performance Analysis with OMNeT++," M.S. thesis, IBM Zurich Research Laboratory, Institut Eurécom, Sep. 2005.

[14] A. Varga, *OMNeT++*, Retrieved Aug., 13, 2012, from http://www.omnetpp.org.

[15] P. Th. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec. "The many faces of publish/subscribe," *J. ACM Comput. Surveys (CSUR)*, vol. 35, no. 2, pp. 114-131, June 2003.

[16] S. Behnel, L. Fiege, and G. Muehl, "On quality-of-service and publish-subscribe," in Proc. *26th IEEE Int. Conf. Distributed Comput. Syst. Workshops (ICDCS 2006),* pp. 20, Lisbon, Portugal, July 2006.

[17] I. S. Jung, H. W. Kim, D. K. Hong, and H. T. Ju, "Protocol reverse engineering to Facebook messages," in *Proc. Int. Conf. Intell. Syst., Modelling and Simulation (ISMS)*, Jan. 2013.

[18] D. Wessels, H. Nordström, A. Jeffries, A. Rousskov, F. Chemolli, R. Collins, and G. Serassio, *Squid.* Retrieved Dec., 12, 2013, from http://www.squid-cache.org.

[19] Funambol, *SyncML*, Retrieved Jan., 4, 2013, from http://www.funambol.com.

[20] Mosquitto, *Mosquitto*, Retrieved Aug., 15, 2012, from http://www.mosquitto.org.

[21] D. Crockford, *Introducing Json(2006)*, Retrieved Aug., 4, 2012, from http://www.json.org.

[22] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta "Comparison of JSON and XML data interchange formats: A case study," in *Proc. ISCA 22nd Int. Conf. Computer Applicat. Ind. Eng. (CAINE 2009)*, pp 157-162, San Francisco, California, USA, Nov. 2009.

[23] Eclipse, *Paho Project*, Retrieved Aug., 15, 2012, from http://www.eclipse.org/paho.

[24] IETF, *Path MTU Discovery*, Retrieved Aug., 20, 2012, from http://www.ietf.org/rfc/rfc1191.txt.

이 신 호 (Shinho Lee)

2012년 2월 계명대학교 컴퓨터공학과 학사

2012년 3월~현재 계명대학교 컴퓨터공학과 석사과정

<관심분야> SNS, Mobile 네트워크 관리

김 현 우 (Hyeonwoo Kim)

2010년 8월 계명대학교 컴퓨터공학과 학사

2010~2012년 계명대학교 컴퓨터공학과 석사

2012년 9월~현재 계명대학교 컴퓨터공학과 박사과정

<관심분야> Firewall, 네트워크 관리

주 홍 택 (Hongtaek Ju)

1989년 한국과학기술원 전자계산학과 학사
1989~1991년 포항공과대학교 컴퓨터공학과 석사
1991~1997년 대우통신 종합연구소 선임연구원
1997~2002년 포항공과대학교 컴퓨터공학과 박사
2002년~현재 계명대학교 컴퓨터공학과 부교수
2007년~현재 대구경북과학기술원 겸임연구원
<관심분야> 네트워크 및 시스템 관리, Embedded 웹 서버를 이용한 네트워크 요소관리, SyncML, 인터넷 침입 예측, 네트워크 보안