https://doi.org/10.7840/kics.2024.49.12.1801

사물인터넷 환경에서 CoAP 응용을 위한 QUIC 프록시 개발 및 성능 평가

최 수 영◆, 추 영 열°

Development of a QUIC Proxy for CoAP Applications in IoT Environment and its Performance Evaluation

Su-young Choi◆, Young-yeol Choo°

요 약

CoAP 기반 디바이스로부터 수집된 IoT 데이터를 다양한 사용자나 시스템에 제공하기 위해 인터넷에 연결할 수 있는 인터페이스 시스템이 필수적이다. 이는 CoAP 메시지를 인터넷 응용 서비스 메시지로 변환하거나 반대로 변환하는 역할을 수행할 수 있는 프록시 시스템을 통해 달성할 수 있다. 이 논문에서는 CoAP 기반 무선 센서 네트워크를 인터넷에 연결하기 위한 QUIC에서 CoAP 프록시 서버의 개발과 QUIC 프록시 및 HTTP 프록시를 사용하는 CoAP 응용 프로그램의 전송 효율을 비교하기 위한 실험을 설명한다. 또한, Cooja 시뮬레이터를 사용하여 5%, 10%, 15%의 패킷 손실 조건에서 CoAP Basic, CoCoA 및 Adaptive RTO의 혼잡 제어 알고리즘 성능을 테스트 한다. 위에서 언급한 패킷 손실 조건에서 QUIC 및 HTTP 프록시와 결합된 CoAP 응용 프로그램의 통신 지연을 측정하고 개발된 QUIC 프록시 서비스의 동작 테스트를 수행했다. 실험 결과, QUIC 프록시와 결합된 CoAP 응용 프로그램이 HTTP 프록시와 결합된 응용 프로그램보다 8% 이상 뛰어났음을 보여주었다.

키워드 : 코앱, 코코아, 퀵, 프록시, 혼잡제어
Key Words : CoAP, CoCoA, QUIC, Proxy, Congestion Control

ABSTRACT

An interface system capable of connecting to the internet is essential for providing IoT data collected from CoAP-based devices to various users or systems. This can be achieved through a proxy system that converts CoAP messages into internet application service messages or vice versa. This paper describes the development of a CoAP proxy server over QUIC to connect CoAP-based wireless sensor networks to the internet and the experiments conducted to compare the transmission efficiency of CoAP applications using QUIC and HTTP proxies. Additionally, the performance of the congestion control algorithms CoAP Basic, CoCoA, and Adaptive RTO was tested under packet loss conditions of 5%, 10%, and 15% using the Cooja simulator. Under these packet loss conditions, the communication delay of CoAP applications combined with QUIC and HTTP proxies was measured. The experimental results showed that the CoAP application combined with the QUIC proxy outperformed the application by more than 8%.

◆ First Author : Tongmyung University Department of Computer Engineering, suyoung@tu.ac.kr, 정회원
° Corresponding Author : Tongmyung University Department of Computer Engineering, yychoo@tu.ac.kr, 정회원
논문번호 : S202407-134-B-RE, Received July 4, 2024; Revised August 8, 2024; Accepted August 23, 2024

I. 서 론

무선 센서 네트워크(WSN, Wireless Sensor Network)는 현대 정보통신 기술의 주요 분야 중 하나로, 센서 노드들이 무선 통신을 통해 데이터를 수집, 처리, 전송하는 네트워크 시스템을 의미한다. 이러한 네트워크들은 작고 저렴한 센서 노드들을 사용하여 환경의 물리적 또는 환경적 조건을 모니터링하고 데이터를 중앙 처리 위치로 전송한다. 이 기술은 다양한 응용 분야에서 활용되며, 농업, 환경 모니터링, 산업 자동화, 보안 등 다양한 산업 분야에서 중요한 역할을 하고 있다. 인터넷의 보급과 스마트 시티, 사물인터넷(IoT) 등의 새로운 개념의 발전으로 인해 WSN 기술에 대한 수요가 더욱 증가하고 있다. 이러한 추세는 WSN 기술의 시장을 더욱 성장시키고 있으며, 이에 따라 기업들은 새로운 기술과 솔루션을 개발하여 시장 경쟁력을 확보하기 위해 노력하고 있다. 국내의 제조업도 글로벌 환경 규제에 선도적으로 대응하기 위해, 다른 한편으로는 고부가가치화를 위해 IoT 기술과의 융합에 박차를 가하고 있다[1].

IoT 장비들은 계산 능력이나 메모리 용량, 전원의 공급 측면에서 성능이 제한 되어있다. 이러한 장비들이 인터넷과 웹 응용에 접속하기 위해서는 많은 제약사항이 있어 이를 위해 IETF(Internet Engineering Task Force)에서 UDP(User Datagram Protocol) 기반의 웹 전송 프로토콜인 CoAP(Constrained Application Protocol) 표준이 제안되었다[2]. 또한, 인터넷에 접속되는 IoT 기기들의 증가에 따라 사물인터넷에서의 혼잡제어에 관한 많은 연구들이 수행되었다[3,5]. 기본 CoAP 표준은 BEB(binary exponential backoff)와 RTO(Retransmission Time Out)를 기반으로 하는 혼잡 제어 메커니즘을 제공한다[2]. 하지만, 다양한 연구에 따르면 혼잡 수준이 높을 때, CoAP의 성능이 저하되는 것으로 나타났다[3]. 이에 RTT(Round Trip Time) 측정, 동적 RTO 백오프 계산 및 RTO 에이징 방법을 사용하여 기본 CoAP 혼잡제어 메커니즘 개선한 CoCoA(CoAP Simple Congestion Control/Advanced)가 IETF CoRE 작업 그룹에서 표준화되고 있다. 하지만 (I. Florea, R. Rughinis, 2017)에 따르면 기본 CoAP 혼잡제어와, CoCoA의 초당 요청 수(throughput) 측면의 성능 평가에서는 CoCoA가 언제나 기본 CoAP 혼잡제어 보다 우수한 것은 아니라는 것을 보여주며, CoCoA의 혼잡제어는 TCP의 SRTT(Smoothed Round Trip Time)의 변형 알고리즘으로 매 송/수신시 RTO를 재설정하기 위한 연산이 이루어지며 이는 곧 센서 노드의 에너지 소모량을 가중시키게 되며 매 전송마다 측정된 RTT값의 기반으로 예측 RTT를 계산하기 때문에 측정 RTT값의 변동폭이 큰 경우에는 효율적으로 대응할 수 없는 문제점을 가지고 있다[5,6]. 이러한 단점을 보완하여 일정 기간 또는 횟수의 통신 왕복 시간(RTT)과 재전송 횟수를 측정하여 통계적인 기법을 적용, 네트워크 환경의 손실율을 파악하고 파악된 손실율에 따라 적용 가중치(α)를 달리하는 방법으로 예측 RTT값을 계산하는 적응형 RTO(Adaptive RTO)가 제안되었다[6].

CoAP은 저전력의 패킷 손실이 있는 네트워크에서 제한된 자원을 가진 장치들이 웹에 접속될 수 있도록 제정된 UDP 기반의 경량 프로토콜로, HTTP 기능의 제한된 부분 집합만을 지원한다[2]. 따라서 CoAP 장비와 웹 응용과의 접속을 위해서는 CoAP 메시지를 HTTP로 변환하거나 그 반대의 역할을 수행하는 Proxy System이 필요하다. CoAP의 RESTful 디자인, 메서드, 응답 코드, 그리고 URI 스키마의 유사성은 HTTP와의 기술적 호환성을 용이하게 한다. HTTP는 전송계층 프로토콜로 TCP를 사용하여 흐름 제어, 혼잡 제어, 오류 제어 등의 기능을 제공하지만, Head-of-Line (HoL) 차단 및 느린 연결 설정 시간 등과 같은 문제점이 있어 HTTP/2에서는 다수의 병렬 TCP 연결을 통해 이를 해결코자 하였다[7]. UDP를 사용함으로써 발생하는 오류 제어 문제는 CoAP에서 CON 메시지 형식을 사용하여 피기백 응답 (Piggybacked Response)으로 일부 보완이 가능하나 TCP만큼의 신뢰성을 제공하지는 못한다.

Google에서는 이러한 HTTP + TCP 조합의 문제점 해결을 위해 QUIC을 개발하였고 이를 YouTube 등의 서비스에 적용하여 성능에서 향상된 실험 결과를 발표하였다[7]. 이 결과는 보완을 거쳐 IETF에서 표준으로 제정되었고 현재도 일부 보완이 진행중이다[8,9]. QUIC은 UDP 기반의 통신사양으로 HTTP의 문제점인 3방향 핸드쉐이크로 인한 지연시간 단축, 네트워크 성능의 향상, 보안 기능의 강화를 주 목적으로하여 이미 구글 탐색, YouTube 외에도 여러 인터넷 응용에서 채택되어 사용되고 있다. UDP 기반의 웹 응용을 지원한다는 점에서 CoAP의 단점 보완 및 정합성이 더 높을 것으로 기대된다. 이 논문에서는 IoT 환경에서의 CoAP 응용을 위한 QUIC Proxy Server 시스템을 구현하여 WSN 장비의 데이터가QUIC 기반의 웹 응용에서 서비스 될 때의 성능과 기존 TCP를 사용하는 HTTP 응용에서 서비스 될 때의 전송 효율을 비교 분석한다. 첫 번째로 네트워크 시뮬레이터인 Cooja 시뮬레이터를 활용하여 5%, 10%, 15%의 무선 손실 환경에서 CoAP Basic, CoCoA, Adaptive RTO 혼잡제어 알고리즘의 성능 비

교 결과를 기술한다. 두 번째로 HTTP/TCP와 QUIC/UDP의 통신 속도를 비교 분석 한다. 이를 위해 HTTP Client와 QUIC Client를 각각 C#과 GO 프로그래밍 언어로 개발하였으며, 각 프로토콜의 요청 메시지를 CoAP 메시지로 변환 또는 그 반대의 역할을 수행하기 위한 HTTP to CoAP Proxy Server와 QUIC to CoAP Proxy Server를 각각 Node js와 GO 언어로 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서 이론적 배경으로 CoAP 혼잡제어 메커니즘과 QUIC 프로토콜을 설명하고 3장에서 HTTP와 QUIC 성능 측정을 위한 시스템 개발 내용을 기술한다. 4장에서 실험 및 방법론을 설명하고 5장에서 CoAP Basic, CoCoA, Adaptive RTO, 그리고 HTTP/TCP와 QUIC/UDP의 성능을 측정하고 그 결과를 분석한다. 끝으로 6장에서 결론을 맺는다.

II. 관련연구 및 이론적 배경

이 장에서는 제한된 자원을 갖는 장치와 함께 사용하기 위한 웹 전송 프로토콜인 CoAP의 Basic 혼잡제어와, CoCoA, Adaptive RTO 혼잡제어에 대해서 설명하고 Google Inc.에 의해 개발된 전송 프로토콜인 QUIC에 대해 간략히 기술한다.

2.1 기본 CoAP 혼잡제어

CoAP는 확인가능(CON), 재설정(RST), 확인불가능(NON) 및 승인(ACK) 메시지의 네 가지 유형의 메시지를 지정한다[2,6]. 확인형(CON) 메시지로 통신을 할 경우 수신 측에서 응답(ACK)을 보내도록 규정하고 있다. 확인형 메시지는 전송 후 재전송 타임아웃(Retransmission Time Out, RTO)내에 응답이 없으면 메시지를 재전송하고 RTO 값은 아래 식(1)과 같이 계산된다.

$$RTOstandard = ACK\_TIMEOUT \times ACK\_RF \quad (1)$$

여기서 표준안에서는 ACK_TIMEOUT 값은 2초, ACK_RF는 1~1.5 사이의 random 값을 갖는 것으로 정의되어있다. 결과적으로 RTO는 2~3초 사이의 임의의 값으로 결정된다. CoAP 요청 메시지 전송 후 RTO 내에 응답을 받지 못하면 RTO를 2배 증가시키고 메시지를 재전송한다. 재전송을 최대 재전송 횟수인 4회 수행하였음에도 응답을 받지 못하면 리셋 메시지를 전송하고 통신을 종료한다[2].

2.2 CoCoA 혼잡제어

기본 CoAP 혼잡 제어의 주요 단점은 네트워크 환경 변화에 둔감하다는 것이다. 이 문제를 해결하기 위해 CoCoA는 적응형 RTO 계산, 가변 백오프 팩터(VBF), 그리고 RTO 에이징 기법을 활용한다. CoCoA의 적응형 RTO 계산은 SRTT와 유사하게 지수 가중 이동 평균(EWMA)을 사용하여 혼잡을 감지하고 RTO 값을 조정하기 위해 RTT와 RTT 변동 추정치의 가중 평균을 계산한다. 그러나 SRTT와 달리, CoCoA는 각 원격 수신자에 대해 강한 RTO 추정기(RTTstrong)와 약한 RTO 추정기(RTTweak) 등 두 가지 형태의 RTO 추정 방식을 사용한다. RTT와 RTO 값은 아래 식(2)와 식(3)에 따라 계산된다.

$$\begin{aligned} RTTx &= (1-\alpha) \times RTTx + RTTxnew \\ RTTVARx &= (1-\beta) \times RTTVARx + \\ &\quad |RTTx - RTTxnew| \end{aligned} \quad (2)$$

$$RTOx = RTTx + Kx \times RTTVARx \quad (3)$$

여기서 α와 β 값은 각각 0.125와 0.25이고, x는 강한 추정기(strong) 또는 약한 추정기(weak)를 의미한다. 식(3)에서 K_x는 x가 strong일 때는 4의 값을, weak일 때는 1의 값을 갖는다[6]. 마지막으로, 수신자에 대해 유지되는 총 RTO 값을 나타내는 RTOoverall은 식(4)와 같이 계산된다.

$$RTOoverall = \gamma x \times RTTx + (1-\gamma x) \times RTOoverall \quad (4)$$

여기서 γstrong은 0.5이고 γweak은 0.25이다. 그런 다음 RTOoverall은 초기 RTO (RTOinit)를 결정하는데 사용된다.

2.3 Adaptive RTO 혼잡제어

적응형 RTO 알고리즘은 적응형 RTO 계산을 사용하여 혼잡 상황을 감지하고 RTO 값을 조정한다. SRTT와 유사하게 EWMA를 사용하여 RTT와 RTT 변동 추정치의 가중 평균을 계산한다. 네트워크 상태를 효율적으로 평가하기 위해 일정 기간 또는 통신 횟수 동안의 왕복 시간(RTT)과 재전송 횟수를 활용하여 네트워크의 손실율을 파악한다. 파악된 손실율은 식(2)에서 RTT 예측값을 계산하기 위한 가중치(α)로 사용된다.

또한, RTO 계산의 부하를 줄이고 측정된 RTT 값의 급격한 변화에 대응하기 위해 일정 기간 또는 통신 횟수 동안 측정된 RTT 값의 평균값을 사용한다. 이 기간이

나 통신 횟수는 네트워크 상태에 따라 조정될 수 있다. 이러한 접근 방식은 네트워크 성능을 향상시키고 무선 센서 네트워크의 수명을 연장하는 데 기여할 수 있다[6].

2.4 QUIC

QUIC은 원래 Google Inc.에 의해 개발된 전송 프로토콜로, 최근에 IETF에 의해 표준화된 현대적인 전송 계층 프로토콜이다[8]. QUIC은 TCP의 몇 가지 제한을 극복하고자 만들어졌으며, 특히 웹 트래픽을 처리할 때 더 나은 성능을 제공하기 위해 설계되었다. 그림 1은 HTTP 트래픽을 전송할 때 사용되는 전통적인 TCP/TLS 접근 방식과 비교하여 QUIC 프로토콜 스택을 보여준다[10,11].

QUIC은 전송 계층으로 UDP를 사용하여 TCP 3방향 핸드셰이크에서 하나의 1-RTT 지연을 줄인다. 이는 QUIC과 TLS 핸드셰이크를 결합함으로써 달성된다. 통신을 이전에 이미 수립한 끝점이 있는 경우, 최신 버전의 TLS 프로토콜 채택인 TLS 1.3은 클라이언트가 TLS 핸드셰이크가 완료되기 전에 애플리케이션 데이터를 보낼 수 있도록 하여 1-RTT 및 0-RTT를 모두 지원한다[12,13]. QUIC 프로토콜을 사용하면 첫 번째 핸드셰이크 협상에 1-RTT가 걸리지만 이전에 연결된 클라이언트는 캐시된 정보를 사용하여 0-1 RTT만으로 TLS 연결을 복원할 수 있다[14].

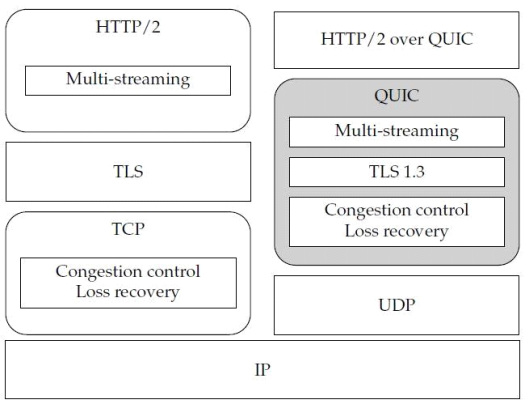

그림 1. 전통적인 전송 계층의 TCP/TLS 아키텍처와 QUIC 프로토콜 스택 비교
Fig. 1. Traditional architecture of the transport layer TCP/TLS vs. the QUIC protocol stack

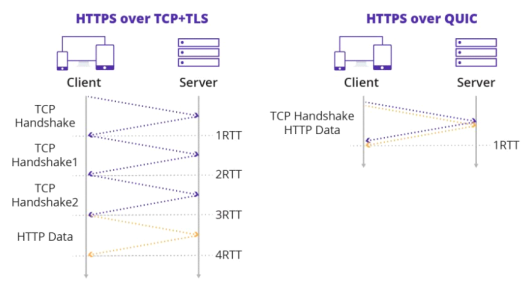

그림 2. QUIC의 빠른 핸드셰이크 및 연결 설정
Fig. 2. QUIC's fast handshake and connection establishment

Ⅲ. IoT 서비스 전송을 위한 Proxy 및 Client 시스템 구현

3.1 QUIC Proxy Server 및 QUIC Client

IoT를 위한 전송 프로토콜로서 QUIC의 분석을 수행하기 위해 GO 언어로 개발된 quic-go 라이브러리를 활용하여 서버 및 클라이언트를 구현하였다. 이 두 프로그램은 동일한 GO 1.22.2 버전을 기반으로 하며, 오픈소스이기 때문에 QUIC를 프로젝트 기능에 맞게 쉽게 접근하여 수정할 수 있다. 그림 3은 QUIC Client와 QUIC Proxy Server의 구현에 대한 사항을 준다. QUIC Client의 QUIC 클라이언트의 'quic.DialAddrEarly()' 함수는 클라이언트가 서버에 초기 데이터를 보내면서 0-RTT 연결을 시작할 수 있게 한다. 0-RTT는 연결을 시작하면서 처음부터 데이터를 전송할 수 있도록 하여, 연결 설정 시간을 단축시킨다. 이 기능은 특히 연결을 자주 새로 시작해야 하거나 지연 시간이 중요한 응용 프로그램에서 유용하다.

QUIC Proxy 서버는 라즈베리파이3B+(Rasbian GNU/Linux 9) 내부에서 실행되며, 'quic.ListenAddr()' 함수는 주어진 주소와 TLS 설정을 사용하여 QUIC 서버를 시작한다. 이 함수는 네트워크 주소를 리스닝하고, 클라이언트로부터의 연결을 수신 대기한다. quic.ListenAddr()는 quic.Listener 인터페이스를 구현하는 객체를 반환하며, 이 객체를 통해 서버는 들어오는 연결을 수락한다. QUIC 클라이언트로부터 요청을 대기하다가 요청이 수신되면 요청 메시지를 CoAP-URI로 변환하여 CoAP 서버에게 데이터를 요청하기 위해 go-coap(github.com/plgd-dev/go-coap/v3) 라이브러리를 사용하여 CoAP 클라이언트를 생성한다. 생성된 CoAP 클라이언트는

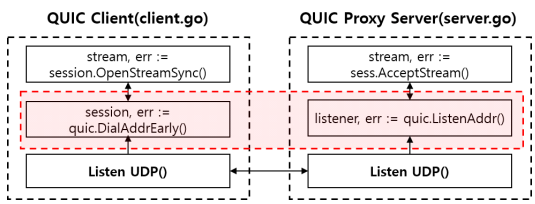

그림 3. QUIC 서버와 클라이언트 구현
Fig. 3. Implementation for QUIC server and client

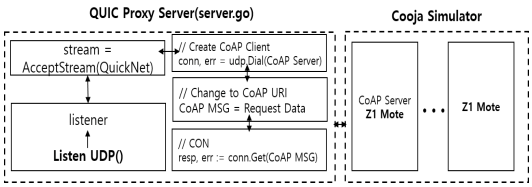

그림 4. CoAP 서버와 클라이언트 구현
Fig. 4. Implementation for CoAP server and client.

네트워크 시뮬레이터인 Cooja 시뮬레이터에서 구성된 CoAP서버에게 접속하여 데이터를 요청하게 되며, 요청을 받으면 해당하는 자원을 센싱하여 QUIC Proxy 서버에게 응답을 보내고 Proxy 서버는 이를 QUIC 클라이언트로 반환한다. 그림 4는 QUIC Proxy Server와 CoAP Server의 구현을 나타낸다[15].

3.2 HTTP Proxy Server 및 HTTP Client

HTTP 클라이언트는 C#언어로 개발되었으며, 웹페이지를 통해 라즈베리파이 내의 HTTP Proxy 서버를 통해 CoAP 서버에게 데이터를 요청하고 응답을 받아 웹페이지에 표시해준다. HTTP Proxy 서버는 HTTP 요청을 수신하면 해당 URI을 파싱하여 CoAP-URI로 변환하여 변환한 메시지를 통해 CoAP 서버에 데이터를 요청하고 응답을 받아 HTTP 클라이언트에게 전송하는 역할을 수행한다. 그림 5는 HTTP Proxy 서버의 메시지 변환 과정을 나타낸다.

CoAP-URI의 HOST는 Node.js로 구현된 HTTP Proxy 서버가 설치되어 있는 라즈베리파이의 주소, PORT는 CoAP의 기본 포트인 5683, PATH는 CoAP 서버의 주소, CoAP_method는 GET, POST, PUT, DELETE 중 하나의 값, RESOURCE는 자원의 경로를 나타낸다.

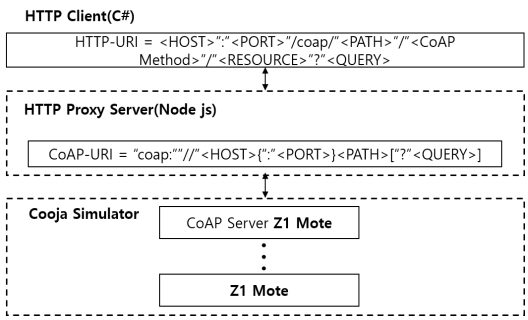

그림 5. HTTP Proxy 서버의 메시지 변환
Fig. 5. Message transformation of HTTP Proxy server

Ⅳ. 실험 설정 및 방법론

본 논문에서는 시뮬레이션 도구로써 Cooja 시뮬레이터를 사용하였다. 시뮬레이션은 무선 손실률 5%, 10%, 15% 일 때 CoAP Basic, CoCoA, Adaptive RTO 혼잡제어 알고리즘에 대하여 수행되었다. 네트워크 환경은 Fig. 6.과 같이 4개의 WSN 노드로 구성되었으며 Z1 mote를 사용하였다. 1번 노드는 외부 인터넷과 WSN을 연결하는 Border-Router가 구현되어 있고, 시리얼 통신을 통해 외부와 연결되어 있다. 나머지 2, 3, 4번 노드는 er-coap-server가 설치되어 있고 각 노드는 직선상 45m 간격으로 배치되어 있다. 모든 노드는 1, 2, 3, 4번 순서대로 멀티홉으로 연결되어 있다[6].

혼잡제어 알고리즘의 성능 평가 지표는 그림 9의 ㉮ CoAP RTT 구간으로 RTT평균(Ave RTT), RTT 총합(Total Time) 2개의 값으로 제시된다. RTT 총합은 실험을 수행한 1,000개의 요청/응답 성공 시간으로 총 통신 시간을 알 수 있는 지표이며 이는 재전송 시 발생하는 통신 지연을 처리함에 있어서의 효율성을 평가할 수 있는 하나의 지표가 될 것이다.

또한, 수집된 정보를 사용자에게 전송하기 위한 효율적인 프로토콜을 확인하기 위해 2가지의 실험 Case를 통해 QUIC와 HTTP의 성능을 분석한다. 그림 7과 같이 본 실험을 진행하기 위해 각 프로토콜의 Client는

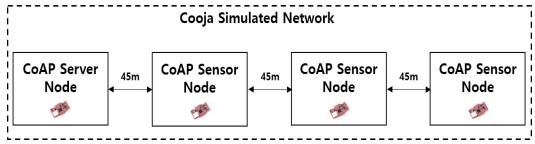

그림 6. Cooja Simulator에서 사용된 무선 네트워크 구성
Fig. 6. Wireless network configuration used in Cooja Simulator

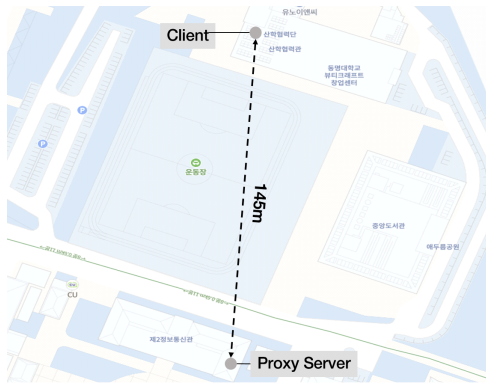

그림 7. 동명대학교 내 시스템 설치 위치
Fig. 7. System installation location at Tongmyong University

동명대학교 8번 건물(학생복지관)에 설치 되며, 각 프로토콜의 Proxy Server는 16번 건물(제2 정보통신관)에 설치하여 진행한다. 두 건물의 직선 거리는 약 145m 정도이다.

4.1 Case 1 : QUIC 기반 IoT 서비스 전송 시스템 성능 평가

성능 평가를 위해 TCP를 사용하는 시스템의 구성은 아래 그림 8과 같으며 CoAP의 혼잡제어 알고리즘은 가장 우수한 성능을 보인 Adaptive RTO 알고리즘이 사용된다. 테스트를 위한 통신 절차는 그림 9에서 보듯이 아래와 같은 절차로 진행된다.

① Client는 QUIC 통신을 이용하여 Proxy에게 센싱 데이터를 요청한다.
② Proxy는 CoAP GET요청을 Cooja 시뮬레이터의 CoAP 서버로 전송한다.
③ Cooja 시뮬레이터의 CoAP 서버는 요청에 대한 응답을 Cooja 시뮬레이터로부터 수신한다.
④ Client에게 전송하게 되고 Client에서 응답받은 RTT를 계산한다.

5%, 10%, 15%의 손실 환경에서 Client가 서버에 연속적으로 1,000개의 요청 메시지를 보내어 응답을 성공할 때까지 실험을 수행하게 된다. 성능 지표는 Fig 9.의 ㉰ Total RTT에서 ㉯ CoAP RTT를 제외한 ㉮

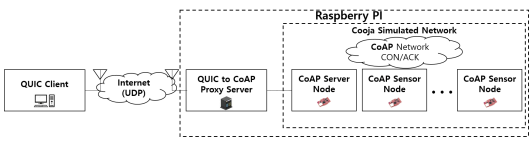

그림 8. QUIC 기반 IoT 서비스 전송 시스템 구성도
Fig. 8. Configuration diagram of QUIC-based IoT service transmission system.

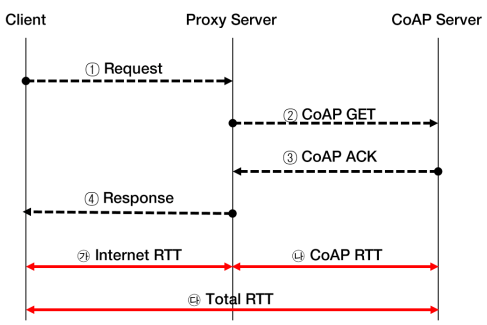

그림 9. 테스트를 위한 통신 절차 및 성능 지표
Fig. 9. Communication procedures and performance metrics for testing

Internet RTT값으로 프로토콜의 성능을 분석한다.

4.2 Case 2 : HTTP 기반 IoT 서비스 전송 시스템 성능 평가

그림 10은 HTTP 기반 IoT 서비스 전송 시스템 구성도이다. QUIC 대신에 TCP를 사용하는 HTTP Client와 HTTP to CoAP Proxy Server로 대체하여 동일한 횟수의 요청 메시지를 보내어 응답을 수신할 때까지의 통신 시간을 측정한다. 통신 절차는 Case 1과 같다.

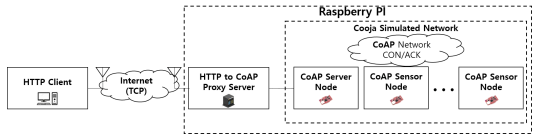

그림 10. HTTP 기반 IoT 서비스 전송 시스템 구성도
Fig. 10. Configuration diagram of HTTP-based IoT service transmission system.

V. 실험 결과

5.1 혼잡제어 성능 측정 결과

5% 손실 환경에서의 RTT 측정 결과를 표 1에 요약하였다. RTT 측정은 Proxy가 CoAP GET요청을 Cooja 시뮬레이터의 CoAP 서버로 전송하고 CoAP 서버는 요청에 대한 응답을 Cooja 시뮬레이터로부터 받아 Proxy에게 전송하는 시간을 말한다. 즉, Fig 8.의 CoAP RTT를 말한다. Adaptive RTO가 기존 CoAP Basic 알고리즘 보다 37.4% 향상된 성능을 보였으며, CoCoA 알고리즘보다 2.3% 우수한 성능을 보였다. Fig. 11.와 같이 RTT 변화폭도 Adaptive RTO 알고리즘이 다른 알고리즘에 비해 안정적임을 확인할 수 있다. 이는 일정한 횟수의 구간 동안 계산된 RTO값을 유지하여 간헐적으로 발생한 높은 RTT의 사용을 제한하고, RTT의 평균을 이용하여 급격한 RTT 변화에 안정적으로 대응하는 Adaptive RTO 알고리즘의 특성 때문일 것이다[6].

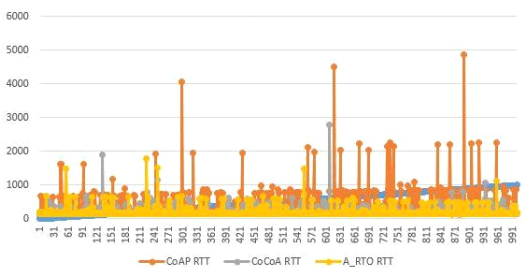

그림 11. 5% 손실 환경에서의 RTT 변화량
Fig. 11. Variation of RTT in 5% loss environment

표 1. 5% 손실환경에서의 RTT 성능 측정(단위 : ms)
Table 1. Measurement of RTT performance in 5% loss environment(unit : ms)

구간	Basic	CoCoA	Adaptive RTO	
	RTT	RTT	RTT	α
1	293.06	198.21	210.13	0.05
2	302.28	214.27	214.53	0.04
3	310.61	230.49	233.63	0.06
4	282.44	208.61	193.53	0.07
5	314.6	219.69	211.03	0.05
6	349.33	198.81	203.29	0.04
7	392.3	224.32	193.24	0.08
8	348.04	180.14	195.1	0.04
9	396.09	228.37	216.59	0.04
10	332.14	223.34	205.98	0.06
Total	3320.8	2126.2	2077	

10% 손실 환경에서의 RTT 측정 결과를 표 2.에 요약하였다. Adaptive RTO가 기존 CoAP Basic 알고리즘 보단 48.2% 향상된 성능을 보였으며, CoCoA 알고리즘보단 6.9% 우수한 성능을 보였다. 또한, 15% 손실 환경에서의 RTT 측정 결과를 표 3.에 요약하였으며, Adaptive RTO가 기존 CoAP Basic 알고리즘 보단 58.3% 향상된 성능을 보였으며, CoCoA 알고리즘보단 8.9% 우수한 성능을 보였다.

표 2. 10% 손실환경에서의 RTT 성능 측정(단위 : ms)
Table 2. Measurement of RTT performance in 10% loss environment(unit : ms)

구간	CoAP Basic	CoCoA	Adaptive RTO	
	RTT	RTT	RTT	α
1	580.09	310.8	309.46	0.1
2	570.36	246.73	245.64	0.08
3	652.76	270.02	297.14	0.11
4	578.21	297.48	264.83	0.07
5	425.01	255.72	219.88	0.09
6	476.53	337.25	315.34	0.09
7	374.8	239.18	271.18	0.08
8	490.19	285.85	245.47	0.09
9	437.86	302.83	234.73	0.14
10	514.07	291	234.95	0.07
Total	5099.8	2836.8	2638.6	

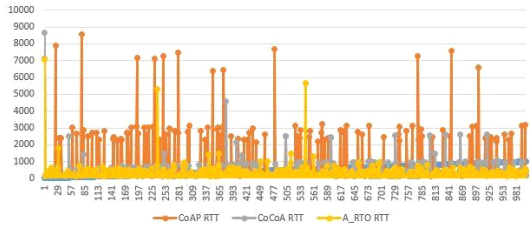

그림 12. 10% 손실 환경에서의 RTT 변화량
Fig. 12. Variation of RTT in 10% loss environment

표 3. 15% 손실환경에서의 RTT 성능 측정(단위 : ms)
Table 3. Measurement of RTT performance in 15% loss environment(unit : ms)

구간	CoAP Basic	CoCoA	Adaptive RTO	
	RTT	RTT	RTT	α
1	577.97	313.85	293.06	0.15
2	768.6	341.16	302.28	0.14
3	757.22	323.99	310.61	0.2
4	810.54	323.87	282.44	0.16
5	1270.95	435.44	314.6	0.14
6	720.07	307.57	349.33	0.11
7	554.18	365.65	392.3	0.09
8	922.89	451.82	348.04	0.12
9	861.26	400.06	396.09	0.17
10	732.95	383.35	332.14	0.19
Total	7976.63	3646.76	3320.89	

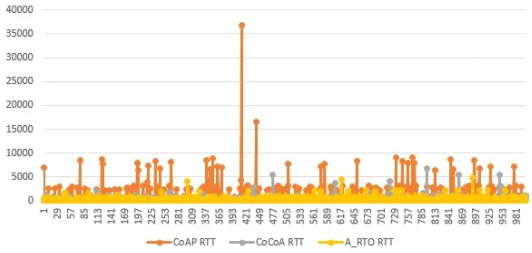

그림 13. 15% 손실 환경에서의 RTT 변화량
Fig. 13. Variation of RTT in a 15% loss environment

5.2 IoT 서비스 전송을 위한 QUIC와 HTTP 프로토콜의 성능 분석

QUIC와 HTTP 프로토콜간 요청/응답 통신 시간(RTT) 측정을 통해 IoT 서비스의 효율적인 전송 프로토콜을 분석 한다. 이를 위해 HTTP Proxy Server와 QUIC Proxy Server가 각각 구현 하였다.

온전히 각 프로토콜간 요청/응답에 걸린 시간을 측정하기 위해 그림 9에서 설명하였듯이 각 프로토콜의

Total RTT에서 CoAP RTT를 뺀 시간으로 분석을 진행하였다. CoAP 혼잡제어 성능 평가와 동일하게 5%, 10%, 15% 손실환경을 구성하였으며, 표 4에 5% 손실환경, 표 5 10% 손실환경, 표 6에 15% 손실환경에서의 각 프로토콜 구간별 RTT 평균을 요약하였다.

5% 손실환경에서의 통신 테스트에선 QUIC 프로토콜 RTT 평균 시간이 82.87msec.으로 측정되었으며 HTTP 프로토콜의 평균 RTT 90.37msec. 보다 7.5msec 정도 성능이 우수하게 나왔으며 10% 손실환경 테스트에선 QUIC 프로토콜이 평균 80.56msec, HTTP 프로토콜이 89.87msec보다 9.32msec, 15% 손실환경 테스트에선 QUIC 프로토콜이 평균 75.46msec, HTTP 프로토콜이 89.88msec보다 14.41msec 정도 평균 통신 시간이 우수했다.

표 4. 5% 손실 환경에서의 QUIC와 HTTP 프로토콜간 RTT 성능 측정(단위 : ms)
Table 4. RTT performance measurements between QUIC and HTTP protocols in a 5% loss environment (unit: ms)

구간	QUIC			HTTP		
	CoAP RTT	Total RTT	UDP	CoAP RTT	Total RTT	TCP
1	202	283	80.9	206.7	297.0	90.3
2	260.3	342.9	82.6	211.9	302.2	90.2
3	247.2	329.8	82.6	214.4	304.4	90.0
4	211.4	295.4	84.0	221.2	311.5	90.3
5	238.8	317.1	78.3	230.5	321.0	90.4
6	213.8	297.5	83.7	220.1	310.7	90.5
7	237.5	319.0	81.5	192.8	282.9	90.0
8	214.4	297.	82.9	243.6	334.4	90.8
9	235.1	317.4	82.2	285.2	375.3	90.1
10	229.3	318.9	89.6	229.4	320.0	90.5

표 5. 10% 손실 환경에서의 QUIC와 HTTP 프로토콜간 RTT 성능 측정(단위 : ms)
Table 5. RTT performance measurements between QUIC and HTTP protocols in a 10% loss environment (unit: ms)

구간	QUIC			HTTP		
	CoAP RTT	Total RTT	UDP	CoAP RTT	Total RTT	TCP
1	345.1	421.5	76.4	268.2	358.6	90.4
2	332.9	408.1	75.2	241.4	331.6	90.2
3	297.4	381.0	83.5	331.5	421.0	89.5
4	347.3	431.2	83.9	287.5	376.9	89.4
5	309.4	388.5	79.0	291.6	381.2	89.5
6	364.1	443.8	79.6	312.8	402.5	89.6
7	265.8	348.4	82.5	228.4	319.5	91.0
8	311.4	393.0	81.6	232.7	321.9	89.2
9	275.6	357.0	81.4	242.9	332.8	89.9
10	283.2	365.3	82.1	264.4	353.9	89.5

표 6. 15% 손실 환경에서의 QUIC와 HTTP 프로토콜간 RTT 성능 측정(단위 : ms)
Table 6. RTT performance measurements between QUIC and HTTP protocols in a 15% loss environment (unit: ms)

구간	QUIC			HTTP		
	CoAP RTT	Total RTT	UDP	CoAP RTT	Total RTT	TCP
1	356.3	434.4	78.1	376.9	467.5	90.6
2	448.5	517.5	69.0	427.8	517.4	89.5
3	437.2	515.3	78.1	319.5	409.9	90.4
4	483.1	551.8	68.6	298.1	387.7	89.5
5	443.	519.7	76.2	322.4	412.3	89.8
6	430	506.7	76.6	336.2	425.2	89.0
7	411.4	490.2	78.8	285.5	375.0	89.5
8	543.	621.5	78.3	386.4	477.2	90.8
9	429	499.8	70.7	315.7	406.1	90.3
10	376.0	455.8	79.7	292.8	381.9	89.0

본 실험 결과로 보아 CoAP Basic과 CoCoA의 혼잡제어 알고리즘은 간헐적으로 발생하는 RTT의 급격한 변화의 환경에서는 성능이 저하되는 것을 확인하였으며 이런 상황에서는 네트워크 상황을 반영하는 Adaptive RTO 알고리즘의 성능이 우수함을 확인하였다. QUIC와 HTTP 프로토콜간 3차에 걸친 실험에서 각각 9.05%, 11.5%, 19.1% 성능이 높게 측정되었으며, 테스트 평균 13.21% 정도 우수한 성능을 보였다.

Ⅵ. 결 론

본 논문에서는 제한된 리소스를 가진 장치에 최적화된 경량 프로토콜인 CoAP의 서비스를 효율적으로 전송하기 위해 QUIC/UDP 및 HTTP/TCP 기반 프록시 서버를 개발하였다. 개발된 프록시 서버의 전송 성능 평가를 위해 5%, 10%, 15% 손실 환경에서 각 1,000번의 요청/응답에 따른 RTT 측정을 각각 진행하였으며, QUIC/UDP 기반 전송 프로토콜이 전통적인 HTTP/TCP 프로토콜 보다 각 9.05%, 11.5%, 19.1% 우수한 성능을 보였다. 또한, CoAP의 혼잡제어 알고리즘의 성능을 분석하기 위해 CoAP Basic, CoCoA, Adaptive RTO 각 알고리즘을 5%, 10%, 15%의 무선 손실 환경에서 클라이언트가 서버에 연속적으로 1,000

개의 요청 메시지를 보내어 응답을 성공할때까지 실험을 수행하였다. 성능 지표로 구간별 RTT 평균과 RTT 총합을 측정한 결과 무선 손실환경 5%에서는 Adaptive RTO 알고리즘이 기존 CoAP Basic과 CoCoA 알고리즘 보다 각각 37.4%, 2.3% 향상된 성능을 보였으며, 10% 손실 환경에서 48.2%, 6.9%, 15% 손실 환경에선 58.3%, 8.9% 성능 향상을 보였다. 본 논문에서는 시뮬레이터 상에서 실험을 수행하였다. 향후, 개발된 QUIC Proxy와 WSN 노드를 사용하여 다양한 인터넷 응용 서비스별로 테스트를 수행할 예정이다.

References

[1] Busan Technopark, "*Survey Report on Technology Trends for Specific Industries of Busan Regional Innovation Cluster*," Dec. 2023.

[2] Z. Shelby, K. Hartke, and C. Bormann, "*Constrained Application Protocol*," RFC 7252, Jun. 2014.

[3] C. Bormann, A. Betzler, C. Gomez, and I. Demirkol, *CoAP Simple Congestion Control/ Advanced, 2018*, from https://tools.ietf.org/html/draft-ietf-core-cocoa-03

[4] I. Florea, R. Rughinis, L. Ruse, and D. Dragomir, "Survey of standardized protocols for the internet of things," *2017 21st Int. Conf., IEEE, Control Syst. and Comput. Sci. (CSCS),* pp. 190-196, 2017.

[5] A. K. Demir and F. Abut, "Comparison of CoAP and CoCoA congestion control mechanisms in grid network topologies," *Demir ve Abut / GUFBED CMES*, pp. 53-60, 2018.

[6] S.-Y. Choi and Y.-Y. Choo, "Design of adaptive RTO algorithm for efficient congestion control in CoAP-based wireless lossy environment," *J. Inst. Control, Robotics and Syst.*, vol. 29, no. 10, pp. 826-833, 2023.

[7] M. Belshe, R. Peon, and M. Thomson, "*Hypertext Transfer Protocol Version 2 (HTTP/2)*," RFC 7540, May 2015.

[8] A. Langley, et al., "The QUIC transport protocol: design and Internet-scale deployment," *SIGCOMM '17: Proc. Conf. ACM Special Interest Group on Data Commun.*, pp. 183-196, Aug. 2017.

[9] J. Iyengar and M. Thomson, "*QUIC: A UDP-Based Multiplexed and Secure Transport*," RFC 9000, May 2021.

[10] H.-B. Nam and S.-J. Koh, "Performance analysis of quic protocol for web and streaming services," *KIPS Trans. Comp. and Comm. Sys.*, vol. 10, no. 5 pp. 137-144, 2021.

[11] M. Thomson and S. Turner, "Using TLS to Secure QUIC," *Internet-Draft Draft-Ietf-QUIC-tls-27*, IETF Secretariat, 2020.

[12] H. Lee, D. Kim, and Y. Kwon, "TLS 1.3 in practice: How TLS 1.3 contributes to the internet," in *Proc. Web Conf. 2021,* WWW'21, pp. 70-79, Ljubljana Slovenia, Apr. 2022, Association for Computing Machinery: New York, NY, USA, 2021.

[13] T. Dierks and E. Rescorla, "*The transport layer security (TLS) protocol version 1.2*," RFC 5246, Aug. 2008.

[14] F. Fernández, M. Zverev, P. Garrido, J. R. Juárez, J. Bilbao, and R Agüero, "Even lower latency in IIoT: Evaluation of QUIC in industrial IoT scenarios," *Sensors*, 2021.

[15] F. Fernández, M. Zverev, P. Garrido, J. R. Juárez, J. Bilbao, and R. Agüero, "Even lower latency in IIoT: Evaluation of QUIC in industrial IoT scenarios," *Sensors*, vol. 21, no. 17, p. 5737, 2021. (https://doi.org/10.3390/s21175737)

최 수 영 (Su-young Choi)

2008년 2월 : 동명대학교 컴퓨터미디어공학과 졸업

2010년 2월 : 동명대학교 컴퓨터미디어공학과 석사

2020년 3월~현재 : 동명대학교 컴퓨터미디어공학과 박사과정

2019~현재 : 동명대학교 컴퓨터공학과 초빙교수

<관심분야> WSN, IoT, 저전력 통신, 실시간 시스템

[ORCID: 0009-0007-4794-3961]

추 영 열 (Young-yeol Choo)

1986년 2월 : 서울대학교 제어계측공학과 공학사
1988년 2월 : 서울대학교 제어계측공학과 공학석사
2002년 2월 : 포항공과대학 컴퓨터공학과 공학박사
1988년 6월~1994년 6월 : 포항산업과학기술연구원 선임연구원.
1994년 7월~2002년 8월 : 포스코 기술연구소 책임연구원
2002년 9월~현재 : 동명대학교 컴퓨터공학과 교수
2005년 1월~7월 : 독일 Fraunhofer IESE Visiting Scientist
2006년 11월~2012년 12월 : 유비쿼터스 항만 ITRC 센터장
<관심분야> 컴퓨터 네트워크, IoT, 네트워크 보안, 실시간 시스템
[ORCID:0000-0002-6497-2584]