논문 21-46-12-21 The Journal of Korean Institute of Communications and Information Sciences '21-12 Vol.46 No.12
https://doi.org/10.7840/kics.2021.46.12.2270

비동기식 지터 상한 보장 네트워크의 구현

권 주 혁◆, 정 진 우°

Implementation of a Jitter Bound Guarantee Network Without Time-Synchronization

Juhyeok Kwon◆, Jinoo Joung°

요 약

네트워크의 시간 동기화 없이 목적지 근처에서의 버퍼를 사용해 지터의 상한을 보장하는 방법이 최근 제안되었다. 이 기술은 버퍼에서 사전에 정한 규정에 따라 특정 시각까지 패킷을 보관하는 방식으로 지터의 상한을 보장하면서 단대단 지연 시간의 상한도 보장한다. 하지만 해당 기술은 버퍼에서의 컷 쓰루(cut-through)가 가능하다는 조건이 필요하다. 본 논문에서는 컷 쓰루가 불가능한 경우에도 지연 시간과 지터 상한을 보장할 수 있다는 것을 증명하였다. 더 나아가 제안하는 기술을 간단한 소규모 임베디드 시스템에 구현하여 제로 지터를 달성할 수 있음을 보였다. 임베디드 시스템은 프로그래머블 하드웨어인 xCORE-200 eXplorerKIT를 사용하였다. 실험은 이상적인 상황과 현실적인 상황을 상정하고 진행하였다. 이상적인 상황에서는 소스와 목적지, 버퍼 모듈을 하나의 디바이스에, 네트워크 모듈을 다른 디바이스에 구현해 클록 드리프트가 없으며, 네트워크의 무적위적인 지연 시간이 없다. 현실적인 상황에서는 소스와 네트워크를 같은 디바이스에, 다른 하나에 버퍼와 목적지를 구현해 소스와 목적지간 클록 드리프트를 발생시켰고 네트워크에서 무작위 지연 시간을 겪게 하였다. 구현 결과 이상적인 상황에서는 제로 지터를 달성하였고 현실적인 환경에서는 지터의 상한보다 훨씬 작은 수준의 지터를 달성하였다. 이를 통해 제안하는 기술이 특별한 하드웨어와 네트워크의 도움 없이 단말의 버퍼와 소프트웨어 코딩만으로 지터 상한 보장이 가능하고 조건에 따라 제로 지터를 달성할 수 있음을 보였다.

Key Words : End-to-end latency guarantee, jitter guarantee, embedded system, buffer, time-stamp

ABSTRACT

It has been recently proposed a method that guarantees jitter upper bounds by using a buffer near the destination, without time synchronization of the network. In this paper, we demonstrate that the latency and jitter upper bounds can still be guaranteed even when cut-through is impossible. Furthermore, it is shown that the proposed technique can be implemented with a simple small-scale embedded system to achieve zero-jitter. The experiments have been conducted in an ideal case and a realistic case. In the ideal case, the source, the destination, and the buffer are implemented in the same device and the network module in the other device. The realistic case implements the source and the network on a single device and the buffer and destination on the other device. In this case it suffers from clock drift between the source and the destination, and random latencies in the network. As a result of the implementation, we achieved zero jitter in an ideal situation and a

※ 본 연구는 과학기술정보통신부 '정보통신방송연구개발사업'의 지원을 받아 수행되었음. (과제고유번호: 2018-0-00846)
◆ First Author : 상명대학교 컴퓨터과학과, Sangmyung University, Department of Computer Science, juhuk98@naver.com, 학생회원
° Corresponding Author : 상명대학교 휴먼지능정보공학과, Sangmyung University, Department of Human-centered AI, jjoung@smu.ac.kr, 정회원
논문번호 : 202107-173-B-RN, Received July 20, 2021; Revised September 27, 2021; Accepted October 4, 2021

level of jitter much smaller than the upper bound of jitter in a realistic environment. It has been shown that the proposed technology can guarantee the upper jitter bound only by software coding at end nodes, without a help from a special hardware or a network, and achieve zero jitter according to conditions.

Ⅰ. 서 론

스마트 팩토리, 차량 내 통신 등의 분야에서 많게는 수 ms에서 적게는 수십 ns 수준의 지터를 요구하고 있다[1-3]. 기존 지터 상한 보장 기술로 네트워크의 시간 동기화를 통한 방식이 있으나 비용이 많이 들고 구현이 어렵다는 단점이 있다. 이를 위해 네트워크의 시간 동기화 없이 목적지 근처에서 버퍼를 사용해 지터의 상한을 보장하는 기술이 제안되었다[4]. 이 기술은 버퍼에서 사전에 정한 규정에 따라 일정 기간 패킷을 보관하는 방식으로 단대단 지연 시간의 상한과 지터의 상한을 보장한다. [4]에서는 플로우들에 대해서 지연 시간 상한을 보장하는 네트워크가 존재한다는 전제가 필요한데, 이러한 조건을 만족하는 기술로 ITU-T Recommendation Y.3113표준 등을 들 수 있다[5,6]. 기존 IEEE TSN[7]에서 제시하는 기술은 시간 동기화된 노드에서 사전에 정한 트래픽 정보를 바탕으로 슬롯 단위 스케줄링을 통해 지터의 상한을 보장한다. 하지만 시간 동기화는 이를 위한 하드웨어가 필요할 수 있고 시간 단위의 정확도가 요구하는 수준보다 낮을 수 있다. 또한 슬롯 단위의 스케줄링은 대규모의 네트워크에서 많은 시간과 자원을 소모한다[8,9]. 이에 비하여, [4]에서 제안하는 방안은 대규모 네트워크에서도 시간 동기화 없이 지연 시간 지터의 상한을 보장하는 것이 증명되었다. 본 논문에서는 [4]의 핵심 기술을 확장하여 컷 쓰루(cut-through)가 불가능한 환경에서도 지터 상한을 보장함을 보였고, 이를 프로그래머블 하드웨어를 사용해 구현해보았다. 또한 클록 드리프트가 발생하는 환경에서도 지터를 보장할 수 있도록 상대적 시간 동기화 방법을 제안하였고 이를 통해 이론상 지터 상한보다 훨씬 작은 지터를 달성하였다.

논문의 구성은 다음과 같다. 2장에서는 이전에 제안된 시간 동기화를 통한 지연 시간 보장과 지터 보장 방식에 대해 알아본다. 3장에서는 제안하는 방식의 개념과 이론, 법칙을 알아보고 그 법칙을 바탕으로 구현을 위한 새로운 법칙을 제안한다. 4장에서는 제안하는 기술의 구현을 위해 고려해야 하는 내용을 다룬다. 5장에서는 모든 조건이 만족한 환경에서의 구현과 결과를 다룬다. 6장에서는 클록 드리프트가 발생하는 현실적인 환경을 고려해 상대적 시간 동기화 방법을 제안하고 이를 구현한 내용과 그 결과를 살펴본다. 7장에서는 결과를 통해 얻은 결론을 서술한다.

Ⅱ. 관련 연구

네트워크에서 지연 시간의 상한을 보장하기 위한 기술로는 Integrated Services(IntServ), Differentiated Services(DiffServ), asynchronous traffic shaping(ATS)[7], flow aggregate interleaved regulator (FAIR)[5,6]가 있다. IntServ는 1) 인입하는 플로우가 특정 도착 곡선(arrival curve)을 따르고, 2) 모든 플로우의 인입 속도의 합은 링크 용량보다 작고, 3) 플로우의 도착 곡선(arrival curve) 이상의 서비스 곡선(service curve)을 따르면 최대 지연 시간을 보장한다는 이론을 따른다. 하지만 조건 3을 만족하는 스케줄러의 구현이 매우 복잡하기에 IntServ보다 구조가 간단한 DiffServ가 제안되었다. DiffServ는 패킷을 8개 또는 32개의 클래스로 구분하고 클래스별로 우선순위가 다른 큐를 할당하는 방법이다. 우선순위가 높은 큐에 할당된 트래픽에 대해서는 조건 1과 조건 2만 만족한다면 최대 지연 시간을 보장할 수 있다. 하지만 DiffServ는 같은 큐에 속한 플로우들의 최대 버스트 크기에 비례해 최대 지연 시간을 갖는 한계가 있다. 이러한 특성으로 인해 순환이 존재하는 네트워크 구조에서는 DiffServ의 최대 지연 시간은 기하급수적으로 커지게 되어 지연 시간을 보장할 수 없다. DiffServ의 구조는 유지하면서 플로우의 최대 버스트 크기를 강제로 유지하는 방법이 ATS이다. IEEE 802.1 TSN 표준에서 제안된 ATS는 모든 노드에 스케줄러와 나란히 interleaved regulator(IR)를 구현해 최대 버스트 크기를 유지한다. ATS에서 1) 스케줄러가 FIFO 특성을 유지하고, 2) IR이 모든 플로우의 인입 특성을 재현하고, 3) IR이 패킷을 전송해야 하는 시각에 즉시 전송할 수 있고, 4) IR에서 컷 쓰루를 지원하면 IR을 구현해도 최대 지연 시간을 증가시키지 않는다. ATS는 순환이 존재하는 네트워크 구조에서도 지연 시간을 보장할 수 있다. ATS는 모든 노드에 구현된 IR로 인해 평균 지연 시간이 증가한다. 이에 모든 노드가 아닌 단위 네트워크의 끝 단에만 IR을 구현하는 방식

인 FAIR가 제안되었다[5]. FAIR는 네트워크를 동일한 구조를 갖는 단위 네트워크의 조합으로 보고 단위 네트워크 끝 단에만 IR을 구현한다. 또한 단위 네트워크에 입출력 포트를 공유하는 플로우를 통합플로우(flow aggregate, FA)로 묶고, 스케줄러가 통합플로우 단위로 스케줄링해 FIFO 특성을 유지하며 FA 별로 IR을 구현한다. FAIR는 단위 네트워크로 이루어진 대규모 네트워크에서 최대 지연 시간을 보장할 수 있다.

IEEE TSN[7]에서 소규모 네트워크를 위한 지터 및 지연 시간 최소화 기술을 제안하였다. 이 기술은 네트워크의 모든 노드를 시간 동기화하고 슬롯 스케줄링을 통해 노드들 간을 조정하는 기술이다. 동기화된 노드들은 사전에 정한 트래픽 플로우에 따라 슬롯을 할당한다. 플로우에 맞춰 슬롯을 할당하는 것이 복잡하지만 다양한 플로우들의 요구사항을 만족시킬 수 있다. 그러나 슬롯 스케줄링은 시간과 자원을 많이 사용해 대규모 네트워크나 무작위 플로우가 발생하는 환경에는 적용하기 어렵다. 고정된 슬롯을 할당한다 해도 TDMA와 유사한 성능을 보인다. 또한 시간 동기화 기술에 대한 어려움이 있다. 시간 동기화를 위해서는 시간 동기화가 가능한 하드웨어가 필요하고 시간 동기화를 하더라도 동기화된 수준이 요구치에 미달할 수 있다. 시간 동기화 기술은 network time protocol(NTP)과 precision time protocol(PTP)이 있다. NTP는 애플리케이션 단에서 시간을 기록하는 방식으로 1 ms 단위까지만 보장할 수 있다. PTP는 마스터 클록과 슬레이브 클록 간의 메시지 교환을 통해 시간을 동기화하는 기술이다. 소프트웨어 단에서 구현할 수 있기에 수십 us까지 보장할 수 있고 하드웨어의 지원이 있으면 1 us 이하까지 보장할 수 있다. 하지만 마스터 클록에서 슬레이브 클록까지 걸리는 지연 시간과 슬레이브 클록에서 마스터 클록 까지 걸리는 지연 시간이 동일하다는 가정을 전제한다. 이 가정은 대규모 네트워크일수록 달성하기 어렵기에 시간 동기화를 사용하는 지터 보장 기술들은 일반적인 네트워크 환경에서 적용하지 못하고 있는 실정이다.

Ⅲ. 지터 상한 보장 네트워크

그림 1은 [4]에서 제안하는 네트워크의 구조이다. a_n은 n번째 패킷이 소스에서 출발한 시각, b_n은 n번째 패킷이 버퍼에 도착한 시각, c_n은 n번째 패킷이 버퍼에서 나간 시각이다. 지터의 상한을 보장하기 위해서는 네트워크가 단대단 지연 시간의 상한을 보장하고 패킷은 타임스탬프를 가져 소스에서 출발한 시각을 기록해야 한다. 네트워크와 목적지 사이에 버퍼가 있어 사전에 결정한 일정 기간 패킷을 보관한다. 이를 통해 소스와 버퍼, 목적지의 정밀한 시간 동기화 없이도 소스와 버퍼의 상대적 시간 동기화만으로 지터의 상한을 보장할 수 있다. 버퍼는 네트워크의 끝 단에 있을 수 있고 목적지에 있을 수도 있다.

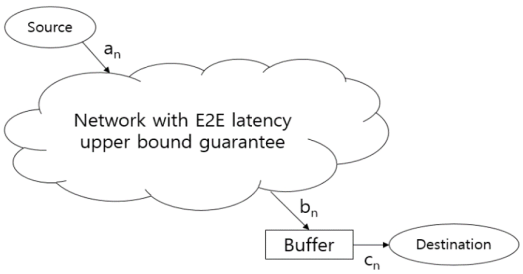

그림 1. 지터 상한 보장을 위한 네트워크 구조
Fig. 1. Network architecture that guarantees upper jitter bound

다음과 같은 법칙으로 버퍼에서 나가는 시각을 결정한다. U는 네트워크의 단대단 지연 시간의 상한, W는 네트워크의 단대단 지연 시간의 하한, m은 지터 조절 변수로 $W \le m \le U$를 만족하는 값이다.

$$c_1 = b_1 + m - W \tag{1.1}$$

$$c_n = \max\{b_n, c_1 + (a_n - a_1)\} \tag{1.2}$$

버퍼는 두 개의 법칙에 따라 패킷을 $c_n - c_1$이 도착간격(inter arrival time)$(a_n - a_1)$과 동일하도록 보관한다. 패킷이 네트워크에서 큰 지연 시간을 겪어 $c_n - c_1 > a_n - a_1$이 되는 경우 패킷을 컷 쓰루한다 $(c_n = b_n)$. [4]는 전체 네트워크의 단대단 지연 시간 $(c_n - a_n)$의 상한$(m + U - W)$을 보장하고 지터의 상한$(U - m)$을 보장함을 증명하였으며 시뮬레이션을 통해 $m = U$일 때 제로 지터를 보장하였다.

그림 2는 a_n, b_n, c_n의 관계를 도식화한 것이다. 표1에서 본 연구에서 사용한 기호와 그 설명을 명시하였다.

앞서 살펴본 법칙 (1.1)과 (1.2)는 컷 쓰루가 가능한 환경을 가정한 것이다. 컷 쓰루가 불가능한 경우 $b_n = c_n$을 만족할 수 없어 지터의 상한을 보장할 수 없다. 본 연구에서는 이를 극복하기 위해 변수 g_n을 추가하여 실제 상황을 더욱 정확히 반영한다. g_n은 n번째 패킷이 버퍼에서 겪는 처리 지연으로 타임스탬

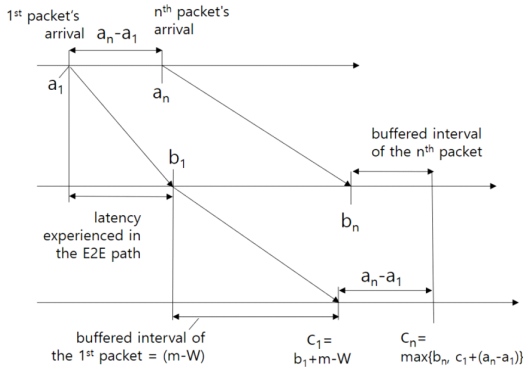

그림 2. a_n, b_n, c_n의 관계
Fig. 2. Relationship between a_n, b_n and c_n

표 1. 기호 설명
Table 1. Symbol description

Symbol	Quantity
a_n	n번째 패킷이 소스에서 출발한 시각
b_n	n번째 패킷이 버퍼에 도착한 시각
c_n	n번째 패킷이 버퍼에서 나간 시각
U	이론상 네트워크의 단대단 지연 시간의 상한
W	이론상 네트워크의 단대단 지연 시간의 하한
m	지터 조절 변수, $W \le m \le U$
g_n	n번째 패킷의 버퍼에서의 처리 지연, $c_n - b_n \ge g_n$
g^*	$\max_n g_n$, $m - W \ge g^*$

프를 읽거나 저장, 전송하는 데 걸리는 시간을 포함하며 패킷을 보관하는 시간은 포함하지 않는다. $c_n - b_n \ge g_n$을 만족한다. $\max_n g_n = g^*$라고 하자.

Theorem 1: $m - W \ge g^*$를 만족할 때 다음 법칙 (2.1)과 (2.2)는 단대단 지연 시간의 최대치 $(m + U - W)$를 보장하며 지터의 최대치$(U - m)$를 보장한다.

$$c_1 = b_1 + m - W \tag{2.1}$$
$$c_n = \max\{b_n + g^*, c_1 + (a_n - a_1)\} \tag{2.2}$$

Theorem 1의 증명

Part 1. 단대단 지연 시간의 상한 보장

모든 n에 대해 $b_n - a_n \le U$, $m - W \ge g^*$이므로

$$\begin{aligned} c_n - a_n &= \max\{b_n + g^*, c_1 + (a_n - a_1)\} - a_n \\ &= \max\{b_n - a_n + g^*, b_1 + m - W - a_1\} \\ &\le \max\{U + g^*, U + m - W\} = m + U - W \end{aligned}$$

이다. 따라서 $c_n - a_n \le m + U - W$를 만족한다.

Part 2. 지터의 상한 보장

n번째 패킷과 1번째 패킷의 지터를 $r_n = r_{n1} = (c_n - a_n) - (c_1 - a_1)$으로 정의한다. 법칙 2.1과 2.2에 따라 r_n은

$$\begin{aligned} r_n &= (c_n - c_1) - (a_n - a_1) \\ &= \max\{b_n - c_1, a_n - a_1\} - (a_n - a_1) \\ &= \max\{b_n - c_1 - a_n - a_1, 0\} \\ &= \max\{b_n - b_1 - m + W - (a_n - a_1), 0\} \\ &= \max\{(b_n - a_n) - m + W - (b_1 - a_1), 0\} \\ &\le \max\{U - m + W - W, 0\} = U - m \end{aligned}$$

이다. i번째 패킷과 j번째 패킷의 지터인 r_{ij}는

$$\begin{aligned} r_{ij} &= \left|(c_i - c_1) - (a_i - a_1) - (c_j - c_1) + (a_j - a_1)\right| \\ &= \left|r_i - r_j\right| \end{aligned}$$

이다. $0 \le r_i \le U - m$, $0 \le r_j \le U - m$이므로 $r_{ij} \le U - m$을 만족한다. ■

Ⅳ. 지터 상한 보장 네트워크 구현 고려사항

본 논문에서는 앞 장에서 설명한 기술을 간단한 임베디드 시스템에 구현하여 제로 지터를 실제로 달성할 수 있는지 알아보고자 한다. 기술 구현을 통해 지터 상한 보장을 달성하려면 다음 3가지 조건을 만족하도록 구현해야 한다.

첫째, 소스와 버퍼의 클록 속도가 동일해서 소스와 버퍼의 상대적 시간이 같은 속도로 흘러야 한다. 소스의 클록 속도가 더 빠르다면 $a_1 < b_1$ 이면서 $a_n > b_n$이 되어 모순이 생길 수 있다. 또한 버퍼에서 보관하는 동안 소스와 목적지에서는 버퍼의 시간보다 더 많은 시간이 지나게 된다. 이렇게 증가한 시간만큼 지터가 발생한다. 시간이 지날수록 증가 폭이 점점 커지기에 지연 시간이 발산하게 되고 지터도 커진다. 반대로 버퍼의 클록 속도가 더 빠르다면 계산한 c_n이 버퍼의 현재 시각보다 작게 되어 버퍼가 수신하자마자 송신하게 된다. 버퍼에서 사전에 정한 지연 시간을 겪지 않기에 지터가 발생한다. 클록 속도가 동일하다는 보장이 없는 경우에는 플로우의 전송 시작 시점과 버퍼 도착 시점인 a_1과 b_1을 주기적으로 갱신해야 한다. 이

방식은 단대단 지연 시간의 발산을 막고 같은 a_1과 b_1이면 지터의 상한을 보장하지만, a_1과 b_1이 갱신될 때마다 전체 플로우의 지터가 발생한다.

둘째, 버퍼와 목적지를 연결하는 링크의 전송 속도가 일정해야 한다. 이는 동일한 길이를 갖는 패킷에 대해 전송 지연으로 인한 지터가 발생하지 않게 하기 위함이다. 다른 링크는 이 조건을 만족할 필요는 없지만, 이론상 지연 시간 최대치를 알아야 하기에 전송 속도의 최소치를 예측할 수 있어야 한다.

셋째, 버퍼는 동시 송수신이 가능하거나 송신을 우선해야 한다. 버퍼에서 수신 중에 송신해야 하는 경우가 있다. 이때 동시 송수신이 가능하다면 문제가 없으나 동시 송수신이 불가능하다면 수신 프로세스로 인해 지연된 만큼 지터가 추가로 발생한다. 버퍼의 수신 프로세스의 경과 시간(elapsed time)을 줄이면 발생하는 지터를 줄일 수 있지만, 상한을 보장할 수는 없다. 버퍼의 동시 송수신이 불가능한 환경에서 지터의 상한을 보장하려면 송신 프로세스에 수신 프로세스가 영향을 주지 않도록 해야 한다. 간단한 방법으로는 모든 수신이 끝난 뒤에 송신을 시작하도록 버퍼가 패킷을 보관하는 시간을 늘릴 수 있다. 하지만 이 방법은 평균 지연 시간을 큰 폭으로 증가시킨다. 다른 방식으로 버퍼를 추가하여 송신 프로세스와 수신 프로세스를 분리할 수 있다. 수신을 담당하는 버퍼와 송신을 담당하는 버퍼를 따로 구현해 송신과 수신이 서로 주는 영향을 없애는 방법이다. 이 경우 버퍼 간의 통신으로 인해 지터가 발생할 수 있기에 송신 버퍼에서 송신 프로세스를 실제 송신 시각보다 일찍 시작할 필요가 있다. 송신 프로세스가 시작하면 실제 송신 시각이 될 때까지 대기한 후에 송신한다.

V. 이상적인 상황 구현 및 실험

5.1 하드웨어 및 프로세스

본 논문에서는 XMOS사의 xCORE-200 eXplorerKIT를 사용해 그림 3과 같이 구현하였다. xCORE-200 eXplorerKIT는 xCORE-200 multicore MCU의 evaluation board로 네트워크, 오디오 등의 분야에서 사용하는 플랫폼이다[10]. xCORE-200 eXplorerKIT는 프로그래머블 하드웨어로 디버그 어댑터를 통해 PC에서 제작한 코드를 보드에서 실행시킬 수 있다. 또한 전이중 통신(full-duplex)인 이더넷 인터페이스를 지원하고 이더넷 인터페이스의 전송 속도는 10, 100, 1000 Mbps를 지원한다. 구현은 전송

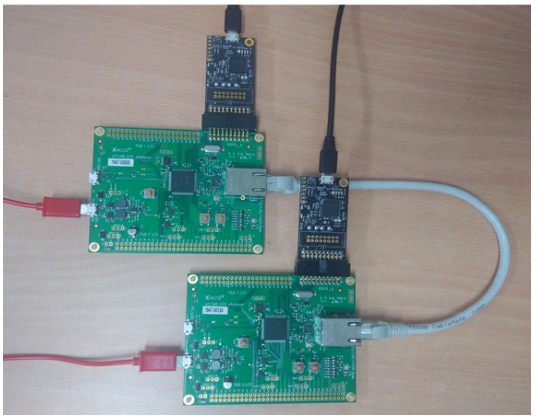

그림 3. xCORE-200 eXplorerKIT로 구현한 결과물
Fig. 3. Results implemented with xCORE-200 eXplorerKIT

속도 1000 Mbps 환경에서 진행하였다.

xCORE-200 multicore MCU는 2개의 타일로 이루어져 있고 타일은 8개의 논리적인 코어로 이루어져 있다. 타일 간 통신은 물리적인 회로로 구현되어 있으며 일정한 속도를 갖는다. 코어 간 통신 또한 일정한 속도를 가짐을 확인했다. 타일별로 메모리가 존재해 코드상에 변수나 상수를 저장하고 버퍼를 만드는 데 사용하였다. 클록 속도는 100 MHz로 10 ns가 최소 제어 단위이다.

xCORE-200 eXplorerKIT는 프로그래머블 하드웨어이므로 컷 쓰루를 지원하지 않는다. 이에 모든 패킷을 버퍼에 넣는 방식으로 구현한다. 따라서 버퍼가 패킷을 보관하는 법칙은 (2.1)과 (2.2)를 따른다.

구현에서는 시스템이 안정되는 시간이 필요하다. 시스템이 안정되지 않았는데 버퍼에 들어오는 첫 번째 패킷을 기준으로 c_1을 정하면 지터가 발생할 수 있다. 이를 위해 시스템이 안정된 후에 들어오는 패킷을 첫 번째 패킷으로 보고 a_1과 b_1을 구해 c_1을 정한다. 시스템이 안정되는 시점은 a_1과 b_1을 매 버스트마다 갱신해 지연 시간이 경험적인 평균과 유사해지는 순간으로 정했다. 본 논문에서는 41번째 패킷을 기준으로 a_1과 b_1을 정하였다.

이상적인 상황은 그림 4와 같이 2개의 노드로 구성되며 패킷의 경로는 노드 1 → 노드 2 → 노드 1로 두 홉만큼 이동한다. 모든 노드는 앞서 설명한 디바이스로 구현하였고 이더넷 인터페이스에서 패킷을 처리하는 부분과 역할을 수행하는 부분으로 구성된다. 이더넷 인터페이스는 LAN 케이블을 통해 패킷을 송신하거나 수신하는 기능을 한다. 이더넷 인터페이스는 별도의 타일에 구현되어 있어 자체적인 버퍼를 갖고 있

다. 이로 인해 구현한 다른 프로세스와 영향을 주고받지 않는다. 지터 상한 보장 기술을 위한 프로세스는 소스, 네트워크, 목적지, 버퍼가 있다.

소스는 250 byte 길이 RTP 패킷 20개를 5 ms마다 보낸다. 소스에서 송신 프로세스의 경과 시간(elapsed time)은 도착 간격(inter arrival time)을 결정하고 조절할 수 있다. RTP 패킷은 bit 단위로 입력할 수 있다. RTP 패킷 헤더의 타임스탬프 필드에 출발 시각을 입력한다. 타임스탬프는 32 bit 길이로 시스템의 클록의 단위와 동일한 10 ns 단위로 입력한다.

네트워크는 받은 패킷을 다시 내보낸다. 처리 지연 외에 지연 시간을 겪지 않고 이더넷 인터페이스와 별도로 구현되어 있기에 약간의 지연 시간만을 겪는다.

목적지는 패킷을 받은 때의 시각과 받은 패킷의 타임스탬프를 보고 지연 시간을 계산한다. 패킷을 받는 즉시 시간을 기록하기에 단대단 지연 시간 계산으로 인한 지연 시간은 적용되지 않는다. 단대단 지연 시간 계산을 위해 목적지의 클록은 소스의 클록에 맞춰 시간 동기화를 해야 한다. 본 구현에서는 소스와 목적지를 동일한 디바이스에 구현했기에 시간 동기화 없이 단대단 지연 시간을 구하였다.

버퍼는 패킷을 버퍼에 넣고 나가는 시각을 계산해 시간이 되면 전송한다. 버퍼의 크기는 1518 * 30 byte이고 추가로 송신 시각을 계산한 값을 저장할 메모리도 할당한다. 패킷의 송신 시각은 법칙 (2.1)과 (2.2)를 따라 정한다. 네트워크의 지연 시간의 상한(U)과 하한(W)은 구현에서 관찰하고자 하는 대상이 아니기에 U는 충분히 큰 시간인 200 us, W는 0 us로 정하였다.

사용하는 시스템이 프로그래머블 하드웨어이기에 버퍼는 동시 송수신이 불가능하다. 따라서 제로 지터를 달성하기 위해 송신 버퍼와 수신 버퍼를 따로 구현하였다. 버퍼의 기능을 상세히 나누면 수신, 계산, 저장, 송신으로 나눌 수 있다. 수신 버퍼는 수신과 계산 기능을 갖는다. 수신 버퍼에서 패킷을 수신하면 패킷과 계산한 송신 시각을 송신 버퍼로 전송한다. 송신 버퍼는 받은 두 데이터를 저장하고 송신 시각보다 약간 이른 시간에 송신 프로세스를 수행한다. 송신 프로세스를 수행해도 즉시 패킷을 송신하지 않고 실제 송신 시각이 될 때까지 기다린 후 송신한다. 송신 프로세스를 수행하는 시각은 계산한 송신 시각에서 저장 기능의 경과 시간(elapsed time)과 수십 ns를 뺀 시각이다. 수십 ns를 추가로 뺀 이유는 최악의 상황인 저장이 끝나자 마다 송신을 해야 하는 경우에 다른 프로세스를 수행하기 위한 전환 중에 약간의 지연 시간이 걸릴 수 있기 때문이다. 성능의 비교를 위해 단일 버퍼만을 사용하는 경우 또한 구현하였다. 이 경우는 수신이 송신에 영향을 줄 수 있어 지터가 발생할 수 있지만, 수신 프로세스의 경과 시간(elapsed time)을 최소화하여 지터를 최소화하였다.

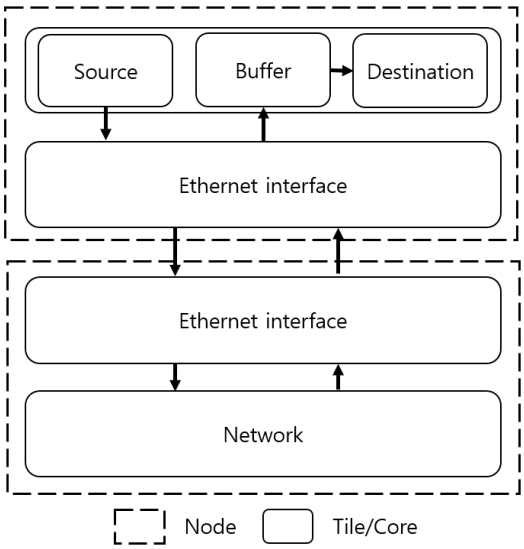

그림 4. 이상적인 상황의 토폴로지와 프로세스 진행
Fig. 4. The processes and the topology in the ideal situation

버퍼의 구현은 네트워크의 끝 단에서 구현하거나 목적지에 구현하는 방법이 있다. 네트워크의 끝 단에 구현하면 목적지에 부담을 덜 주는 이점이 있다. 버퍼가 네트워크의 끝 단에 구현이 가능한지 알아보기 위해 두 시스템의 클록 속도를 측정해 보았다. 만약 클록 속도가 같다면 네트워크의 끝 단에 버퍼를 구현해도 되지만 다르다면 소스, 목적지를 구현한 디바이스에 버퍼를 구현해야 한다. 두 시스템의 클록 속도는 1초 동안 6 us 차이가 난다. 따라서 버퍼는 소스, 목적지와 같은 디바이스에 구현해 같은 클록 속도를 갖도록 했다.

5.2 결과

디바이스 1은 소스, 버퍼, 목적지 프로세스를 수행하고 디바이스 2는 네트워크 프로세스를 수행한다. 이더넷 인터페이스는 1 Gbps의 전송 속도를 가진다. 버퍼의 구현은 이중 버퍼 방식과 단일 버퍼 방식을 구현해 결과를 얻었다. 결과는 15초 동안 전송한 패킷들의 단대단 지연 시간 결과를 얻었다. 시뮬레이션 한번 동안 60,000개의 단대단 지연 시간을 얻었고 시뮬레이션은 총 15번 반복했다. 패킷 전송 초반에는 시스템들이 안정되지 않기에 초반 40개의 패킷은 무시하고 이

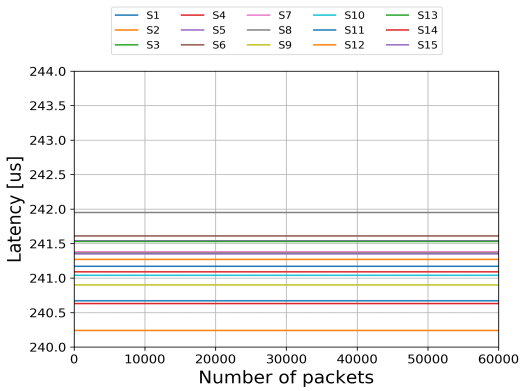

그림 5. 이중 버퍼를 구현한 이상적인 상황에서의 지연시간
Fig. 5. Latency in the ideal situation with double buffer

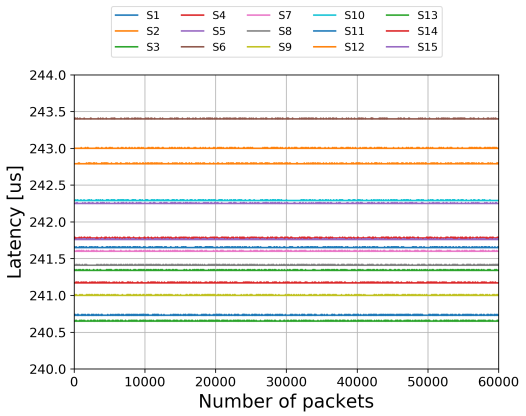

그림 6. 단일 버퍼를 구현한 이상적인 상황에서의 지연시간
Fig. 6. Latency in the ideal situation with a single buffer

후 패킷을 기준으로 지터 상한 보장 기술을 적용하였다. 이중 버퍼를 구현한 경우는 그림 5와 같이 0 us의 지터를 가졌으며 평균 지연 시간은 240 - 242 us이다. 단일 버퍼를 사용한 경우는 그림 6과 같이 0.01 us의 지터를 가졌으며 평균 지연 시간은 240.5 - 243.5 us이다. 0.01 us의 지터는 xCORE-200의 클록 속도가 100 MHz이므로 발생할 수 있는 최소 지터인 클록 속도 수준의 지터이다. 조건을 만족하면서 버퍼에서 지연되는 시간을 바꿔도 이중 버퍼는 제로 지터를 달성했고 단일 버퍼는 0.01 us의 지터를 달성했다.

Ⅵ. 현실적인 상황 구현 및 실험

6.1 프로세스 변경점

4장에서, 제안한 모든 조건을 만족하는 이상적인 토폴로지에서는 제로 지터의 달성이 가능함을 보였다. 이번 장에서는 현실적인 환경에서 지터 상한 보장 기술을 구현하고자 한다. 소스와 버퍼를 다른 디바이스에 구현하여 두 프로세스 간 시간 동기화 없이 클록 드리프트가 일어나는 상황에서 지터를 보장하고자 한다. 또한 패킷이 네트워크에서 무작위의 지연 시간을 겪게 해 실제 네트워크와 유사하게 동작하도록 하였다.

현실적인 상황은 그림 7과 같이 두 개의 노드로 구성되며 패킷의 경로는 노드 1 → 노드 2로 한 홉 이동한다. 모든 노드는 xCORE-200 eXplorerKIT로 구현하였다. 노드 1은 소스, 네트워크 프로세스를 수행하고 노드 2는 버퍼, 목적지 프로세스를 수행한다.

네트워크 프로세스는 들어오는 패킷이 50 - 500 us 범위에서 50 us 단위로 무작위의 지연 시간을 겪게 한다. 버퍼 프로세스는 단대단 지연 시간의 상한(U)을 600 us로 변경했다. 네트워크에서 겪는 최대 지연 시간은 500 us이지만 실제로는 송신 프로세스의 경과 시간(elapsed time), 전송 지연 등을 더 겪게 된다. 이를 고려해 100 us의 여유를 두어 600 us로 정하였다. 단대단 지연 시간의 하한(W)은 50 us로 변경했다.

소스 프로세스와 목적지 프로세스는 5장의 내용과 동일하다.

소스와 버퍼의 클록 속도가 다른 환경에서는 클록 속도 차이에 의한 지터가 발생한다. 이러한 지터를 줄이기 위해 a_1과 b_1을 주기적으로 갱신하는 방법을 앞서 제안했으나 이 방식은 갱신하는 순간 지터가 발생해 제로 지터를 달성하지 못하며 발생하는 지터를 제

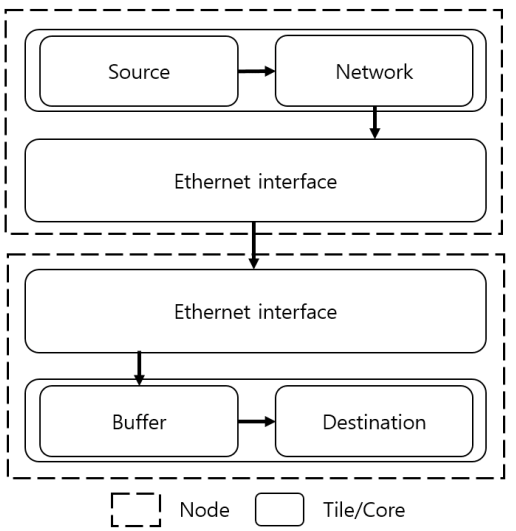

그림 7. 현실적인 상황의 토폴로지와 프로세스 진행
Fig. 7. The processes and the topology in the realistic situation

어하지 못한다. 본 논문에서는, 주기적 클록 갱신을 통한 동기화보다 간단한 상대적 시간 동기화를 통해 버퍼의 클록을 소스의 클록에 맞춰 값을 갱신하는 방법을 제안한다. 버퍼 프로세스에 알고리즘 1을 추가해 매 패킷의 수신 시 b_1을 갱신하면 Theorem 2에 따라 지터의 상한을 보장할 수 있다.

Theorem 2 상대적 시간 동기화: $b_n - b_1 = \beta$, $a_n - a_1 = \alpha$라 할 때, $W - U \le \beta - \alpha \le U - W$를 만족하면 지터의 최대치 $2(U-W)$를 보장한다.

Theorem 2의 증명:

법칙 (2.1)과 (2.2)를 만족하면서 제로 지터를 달성하는 n번째 패킷의 송신 시각은 $c_n = b_1 + (U-W) + (a_n - a_1)$이다. c_n을 b_n에 대해 풀면 $c_n = b_n - (b_n - b_1) + (U-W) + (a_n - a_1)$.

$a_n - a_1$을 α, $b_n - b_1$을 β라 하면 $c_n = b_n + (U-W) - (\beta - \alpha)$이다. 이 중 클록 드리프트를 겪는 부분은 $\beta - \alpha$이다. $a_n + W \le b_n \le a_n + U$이므로 $\beta - \alpha$는 $W - U \le \beta - \alpha \le U - W$를 만족한다.

Case 1: 버퍼의 클록 속도가 빠른 경우

임의의 양의 실수 t에 대해 $b_n' = b_n + t$, $b_n' - b_1' = \beta'$이면 $a_n + W + t \le b_n' \le a_n + U + t$가 되므로 $\beta' - \alpha$의 범위는 $W - U + t \le \beta' - \alpha \le U - W + t$가 된다.

$\beta' - \alpha \le U - W + t$는 $\beta - \alpha$의 범위를 벗어나므로 범위를 만족하려면

$\beta' - \alpha - t \le U - W$,

$b_n' - (b_1' + t) - \alpha \le U - W$.

$b_1' = b_1 + 0$이므로 b_1을 $b_1 + t$로 갱신하면 범위를 만족한다.

$\beta' - \alpha > U - W$인 경우 버퍼의 클록 속도가 빠른 경우이므로 $t \approx k = \beta' - \alpha - U + W$로 근사할 수 있다.

따라서 b_1을 $b_1 + k$로 갱신하면 $\beta' - \alpha = U - W$로 범위를 만족한다. 이때 발생할 수 있는 지터의 최대치는 $\max|t-k| = \max|(U-W) - (\beta - \alpha)| = 2(U-W)$이다.

Case 2: 버퍼의 클록 속도가 느린 경우

임의의 음의 실수 t에 대해 $b_n' = b_n + t$, $b_n' - b_1' = \beta'$이면 $a_n + W + t \le b_n' \le a_n + U + t$가 되므로 $\beta' - \alpha$의 범위는 $W - U + t \le \beta' - \alpha \le U - W + t$가 된다.

$W - U + t \le \beta' - \alpha$는 $\beta - \alpha$의 범위를 벗어나므로 범위를 만족하려면

$W - U \le \beta' - \alpha - t$,

$W - U \le b_n' - (b_1' + t) - \alpha$.

$b_1' = b_1 + 0$이므로 b_1을 $b_1 + t$로 갱신하면 범위를 만족한다.

$W - U > \beta' - \alpha$인 경우 버퍼의 클록 속도가 빠른 경우이므로 $t \approx k = \beta' - \alpha + U - W$로 근사할 수 있다.

따라서 b_1을 $b_1 + k$로 갱신하면 $W - U = \beta' - \alpha$로 범위를 만족한다. 이때 발생할 수 있는 지터의 최대치는 $\max|t-k| = \max|(U-W) - (\beta - \alpha)| = 2(U-W)$이다. ■

알고리즘 1 상대적 시간 동기화 알고리즘

갱신한 b_1: b_1'
$a_n - a_1$: α
$b_n - b_1$: β

if $(\beta - \alpha < W - U)$ then
  $b_1' = b_1 + (\beta - \alpha - W + U)$
else if $(\beta - \alpha > U - W)$ then
  $b_1' = b_1 + (\beta - \alpha - U + W)$
else $b_1' = b_1$
end if

6.2 결과

디바이스 1은 소스, 네트워크 프로세스를 수행하고 디바이스 2는 버퍼, 목적지 프로세스를 수행한다. 이더넷 인터페이스는 1 Gbps의 전송 속도를 가진다. 이중 버퍼 방식과 단일 버퍼 방식을 구현해 결과를 얻어 비교하였다. 버퍼의 클록 속도가 소스의 클록 속도보

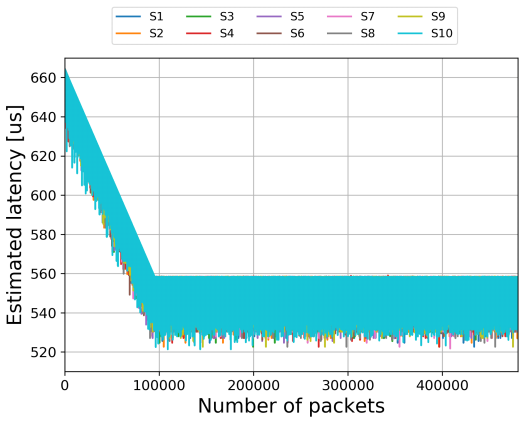

그림 8. 이중 버퍼를 구현한 현실적인 상황에서의 지연 시간
Fig. 8. Latency in the realistic situation with double buffer

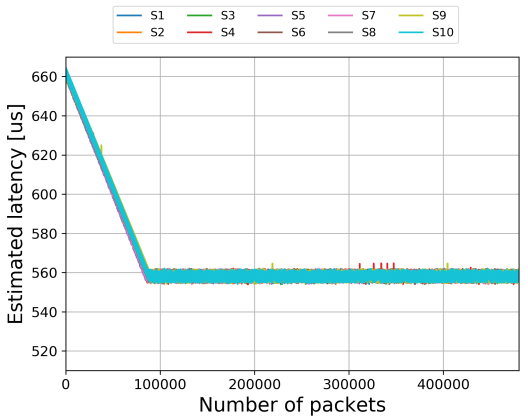

그림 9. 단일 버퍼를 구현한 현실적인 상황에서의 지연 시간
Fig. 9. Latency in the realistic situation with a single buffer

다 빠른 환경에서 구현했다. 결과는 발산이 일어나는 충분히 긴 시간인 120초 동안 진행했다. 본 구현에서는 소스와 목적지를 다른 디바이스에 구현하였기에 정확한 단대단 지연 시간을 구할 수 없었다. 이에 각 노드에서 걸린 지연 시간의 합으로 추정 지연 시간을 구하였다. 추정 지연 시간을 사용해 지터를 구하면 오차가 발생하나 경향을 보는 데에는 지장이 없다고 판단해 지터를 구했다. 시뮬레이션 한번 동안 480,000개의 추정 지연 시간을 얻었고 시뮬레이션은 총 10번 반복했다. 패킷 전송 초반에는 시스템들이 안정되지 않기에 초반 40개의 패킷은 무시하고 이다음 패킷을 기준으로 지터 상한 보장 기술과 상대적 시간 동기화 알고리즘을 적용하였다. 제안하는 알고리즘은 첫 번째 패킷의 지연 시간에 따라 결과가 달라진다. 동일한 환경에서 결과를 비교하기 위해 첫 번째 패킷의 무작위 네트워크 지연 시간은 50us로 고정한 상태로 진행하였다. 단일 버퍼만 사용한 경우는 그림 9와 같이 108 - 111 us의 지터를 가졌고 이중 버퍼를 구현한 경우는 그림 8과 같이 137 - 143 us의 지터를 가졌다. 두 방식 모두 일정 기간 추정 지연 시간이 꾸준히 감소하는데 이는 버퍼의 클록 속도가 소스의 클록 속도보다 더 빨라 버퍼에서 겪는 지연 시간이 줄어들었기 때문이다. 또 일정 시간이 지나면 더이상 추정 지연 시간이 감소하지 않는데 이는 알고리즘 1이 작동해 일정 수준의 지연 시간을 보장해주기 때문이다. 이렇게 알고리즘 1이 작동한 상태에서 이중 버퍼 방식은 40 us 미만의 지터를 갖고 단일 버퍼 방식은 10 us 미만의 지터를 갖는다. 이중 버퍼와 단일 버퍼 방식 모두 지터 최대치인 1.1 ms보다 훨씬 작은 지터를 가졌다. 또한 a_1과 b_1을 주기적으로 갱신하는 방법보다도 더 작은 지터를 가졌다. 이로써 알고리즘 1이 비록 이론상 보장하는 지터 최대치는 크지만 실제 구현 환경에서는 훨씬 작은 지터 최대치를 달성할 수 있음을 보였다.

Ⅶ. 결 론

본 논문에서는 버퍼를 사용한 지터의 상한 보장 기술을 알아보고 구현을 통해 제로 지터 달성이 가능한지 알아보았다. 구현에 사용한 시스템은 프로그래머블 하드웨어인 XMOS사의 xCORE-200 eXplorerKIT를 사용하였다. 제로 지터 달성을 위해서는 버퍼와 소스의 클록 속도가 동일해야 하고 전송 속도가 일정해야 하며 버퍼가 동시 송수신이 가능해야 한다. xCORE-200 eXplorerKIT는 일정한 전송 속도를 보장한다. 첫 번째 이상적인 상황 구현에 있어서, 토폴로지는 2개의 디바이스로 이루어져 디바이스 하나에 소스, 버퍼, 목적지를 구현하였고 다른 디바이스에 네트워크를 구현하였다. 패킷은 두 홉 거리를 이동한다. 버퍼, 소스를 같은 디바이스에 구현해 동일한 클록 속도를 갖게 하였고 소스와 목적지를 동일한 디바이스에 구현해 시간 동기화 없이 단대단 지연 시간을 구하였다. 버퍼의 동시 송수신을 위해 수신 버퍼와 송신 버퍼를 별도로 구현하였고 비교를 위해 단일 버퍼로도 구현하였다. 구현 결과 이중 버퍼 구현은 제로 지터를 달성하였고 단일 버퍼 구현은 최소 수준의 지터를 보장하였다. 두 번째, 현실적인 상황 구현에서의 토폴로지는 2개의 디바이스로 이루어져 디바이스 하나에 소스, 네트워크를 구현하였고 다른 디바이스에 버퍼, 목적지를 구현하였다. 두 디바이스의 클록 속도가 다르기에 제안한 상대적 시간 동기화 알고리즘을 버퍼에 적용하였다. 또한 네트워크에서 무작위 지연 시간을 겪도록 하여 더 현실적인 상황을 실현하였다. 구현 결과 이중 버퍼 구현은 140 us 수준의 지터를 달성하였고 단일 버퍼 구현은 110 us 수준의 지터를 달성해 두 방식 모두 이론상 지터 상한보다 훨씬 작은 지터를 달성하였다. 이를 통해 제안하는 지터 상한 보장 기술은 특별한 하드웨어와 네트워크의 도움 없이 단말의 버퍼와 소프트웨어 코딩만으로 지터 상한 보장이 가능하고 조건에 따라 제로 지터를 달성할 수 있음을 보였다.

본 논문에서는 이상적인 환경에서의 지터 보장 기술 구현과 현실적인 환경에서의 지터 보장 기술 구현을 다루었다. 이중 현실적인 환경에서의 지터 보장 기

술 구현은 정확한 지터를 측정할 수 없어 추정값을 사용해 결과를 구했다. 향후에는 클록 드리프트가 있는 환경에서의 정확한 지터 측정 방법과 이를 적용한 현실적인 환경 구현 결과에 대해 연구할 계획이다.

References

[1] L. Lachello, P. Wratil, and A. Meindl, "*Industrial ehernet facts*," Fredersdorf: Ethernet Powerlink Standardization Group, 2017.

[2] D. Cavalcanti, et al., "Extending accurate time distribution and timeliness capabilities over the air to enable future wireless industrial automation systems," in *Proc. IEEE,* vol. 107, no. 6, pp. 1132-1152, 2019.

[3] A. M. Romanov, F. Gringoli, and A. Sikora, "A precise synchronization method for future wireless TSN networks," *IEEE Trans. Ind. Info.,* vol. 17, no. 5, pp. 3682-3692, 2020.

[4] J. Joung and J. Kwon, "Zero jitter for deterministic networks without time-synchronization," *IEEE Access,* vol. 9, pp. 49398-49414, 2021.

[5] Recommendation ITU-T Y.3113 (2021), *Requirements and framework for latency guarantee in large scale networks including IMT-2020 network.*

[5] J. Joung, "Framework for delay guarantee in multi-domain networks based on interleaved regulators," *Electronics*, vol. 9, no. 3, p. 436, 2020.

[7] *IEEE 802.1 Time-Sensitive Networking Task Group Home Page*, http://www.ieee802.org/1/pages/tsn.html

[8] F. Durr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. and Syst.*, pp. 203-212, Oct. 2016.

[9] M. L. Raagaard, et al., "Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing," *2017 IEEE FWC*, Santa Clara, CA, USA, 2017.

[10] *XMOS*, https://www.xmos.ai/xcore-200/.

권 주 혁 (Juhyeok Kwon)

2020년 8월 : 상명대학교 휴먼지능정보공학과 학사

2020년 9월~현재 : 상명대학교 지능정보공학과 석사

<관심분야> 유무선통신, 네트워크, 임베디드 시스템

정 진 우 (Jinoo Joung)

1992년 2월 : KAIST 전자공학과 학사

1994년 8월 : NYU 전기전자공학과 Master

1997년 8월 : NYU 전기전자공학과 Ph.D.

1997년 10월~2005년 2월 : 삼성전자 종합기술원

2005년 3월~현재 : 상명대학교 휴먼지능정보공학과 교수

<관심분야> 유무선통신, 네트워크, 임베디드 시스템

[ORCID:0000-0003-3053-9691]