JANT 한국항행학회논문지
Journal of Advanced Navigation Technology

기반 기술

J. Adv. Navig. Technol. 24(5): 438-443, Oct. 2020

QuTFTP: UDP 기반의 빠른 파일전송

QuTFTP: Quick UDP Trivial File Transfer Protocol

김 병 국
대한항공 항공기술연구원

Byoung-Kug Kim
Koreanair, R&D Center, Daejeon, 34054, Korea

[요 약]

이더넷을 기반으로 한 네트워크에서 노드 간 파일전송을 위해 대표적으로 사용되는 프로토콜로 TCP/IP 기반의 FTP (file transfer protocol)와 UDP/IP 기반의 TFTP (trivial file transfer protocol)가 있다. 단순 기능을 수행하는 펌웨어 수준의 소프트웨어를 탑재한 내장형 시스템의 경우 자원(프로세서의 성능 및 메모리 용량 등)의 한계로 인해 네트워크 스택의 경우 IP와 UDP의 조합으로 구축한 경우가 많다. 따라서 이를 기반으로 한 TFTP가 많이 선호되고 있다. 예를 들어 환경감지용 센서 또는 부트로더, 그 외 데스크톱 PC의 PXE (preboot execution environment) 부팅 기능 등에 널리 사용되고 있다. TFTP는 파일전송을 위해 stop-and-wait 방식의 운영으로 인해 전송과정에서 많은 대기시간이 발생한다. 본 논문에서는 이 전송 대기시간을 최소화하여 최종적으로 기존 TFTP보다 더 빠르고 호환성을 제공하는 QuTFTP (quick UDP trivial file transfer protocol)를 제안한다.

[Abstract]

To transfer files between nodes on network based on Ethernet, file transfer protocol (FTP) on TCP/IP and trivial file transfer protocol (TFTP) on UDP/IP are mostly used. Due to the lack of resources (processor, memory and so on) in the embedded system where we generally use for simple works with small firmware like ones; many of the systems implement only UDP/IP for their network stacks. Thus, TFTP is greatly to be preferred. For examples, environmental sensor devices for sensor networks, Boot Loader for general embedded device and preboot execution environment (PXE) boot for PC provide the TFTP. The logic of TFTP is simple for file transmission but, there is Stop-And-Wait problem during the process which occurs long blocking time. In this paper, we propose an algorithm which called QuTFTP(Quick UDP Trivial File Transfer Protocol) to reduce the length of the blocking time and to be compatible with the legacy TFTP.

Key word : QuTFTP, FTP, TFTP, File transfer protocol, Trivial file transfer protocol.

https://doi.org/10.12673/jant.2020.24.5.438

Received 28 September 2020; **Revised** 29 September 2020
Accepted (Publication) 14 October 2020 (30 October 2020)

***Corresponding Author; Byoung-Kug Kim**

Tel: +82-43-868-6280
E-mail: byoungkugkim@koreanair.com

Ⅰ. 서 론

네트워크를 구성하는 노드(node) 간 파일전송을 위해 대표적으로 FTP(file transfer protocol)[1]와 TFTP(trivial file transfer protocol)[2][3]가 사용된다. FTP는 TCP/IP를 기반으로 운영되고 TFTP는 UDP/IP를 기반으로 한 응용프로그램용 프로토콜들이다. 파일 전송의 경우 수신단의 데이터의 완결성은 반드시 보장되어야 한다. 이에 전송데이터의 신뢰성을 보장하는 TCP/IP가 주로 적용된다. 이를 위해 TCP의 경우 전송되는 세그먼트(segment)에 다량의 헤더정보(최소 20바이트)를 갖는다. 따라서 소량의 데이터를 SDU(service data unit)를 갖는 세그먼트 전송 시 TCP는 헤더의 상대적 크기로 인해 비효율적일 수 있다.

TCP의 신뢰성 보장을 처리하기 위한 복잡한 로직으로 인해 단순기능을 수행하는 펌웨어 수준의 소프트웨어를 탑재한 내장형 시스템의 경우 자원(프로세서의 성능 및 메모리 용량 등)의 한계로 인해 네트워크 스택의 경우 TCP를 제외한 IP와 UDP의 조합으로 구축한 경우가 많다. 따라서 파일전송 시 이를 기반으로 한 TFTP가 많이 선호되고 있다. 예를 들어 환경감지를 위한 센서네트워크에서의 각종 센서노드, 임베디드시스템 내 커널이미지 다운로딩 및 적재용 부트로더(boot loader), 그 외 PC환경에서 네트워크 부팅을 위한 CMOS (complementary metal oxide semiconductor) BIOS (basic input / output system)용 PXE (preboot execution environment) Boot 기능 등 널리 사용되고 있다.

TFTP(trivial file transfer protocol)는 파일을 고정된 블록크기로 나누어 세그먼트를 순차적으로 송신한다. 송신노드는 매 블록의 전송 시 정상적인 수신을 확인하기 위해 수신 노드로부터 ACK 메시지를 회신 받는다. 이후 다음 블록을 송신 후 ACK 회신의 동작을 마지막 블록의 송신 때까지 반복한다. 즉 데이터의 전송은 Stop-And-Wait 방식으로 처리된다. 이 방식은 전송과정에서 많은 대기시간이 발생하는 문제를 가지며 이로 인해 전송지연 및 수율(data throughput)이 낮아진다. 본 논문에서는 이 전송 대기시간을 최소화하여 최종적으로 기존 TFTP보다 더 빠른 파일전송이 이루어지는 QuTFTP(Quick UDP TFTP)를 제안한다.

이를 위해 본 논문에서는 II에 TFTP를 비롯한 유사 대표적인 연구들[2-9]을 몇 가지 살펴본다. III은 본 논문에서 제안한 QuTFTP를 제안한다. 그 후 IV에서 제안한 기법을 실험을 통해 성능을 측정 후 다른 연구들과 비교한다. 그리고 마지막으로 본 제안의 우수성을 입증 후 결론을 맺는다.

Ⅱ. 관련 연구

2-1 TFTP

TFTP는 RFC-1350(request for comments - 1350)[2]으로 등록되어있다. UDP를 기반으로 파일을 송/수신하기 위한 기능을 위해 패킷을 세부적으로 다섯 가지로 정의한다. 파일의 송신요청 및 수신요청을 위한 그림1의 (a)와 (b)와 같은 RRQ(read request)와 WRQ(write request) 패킷, 블록화된 파일의 내용을 전달하기 위한 데이터 패킷(그림에서 (c)에 해당), 각각의 수신패킷에 대한 ACK 회신용 패킷(그림의 (d)에 해당), 그 외 전송오류 등의 예외를 정의하는 Error 패킷으로 정의한다.

초기 TFTP는 데이터의 블록을 512바이트로 고정하였으나, 이더넷 환경의 발전에 따른 유연성을 제공하기 위해 전송 초기에 블록의 크기를 동적으로 정의할 수 있도록 RFC-2348[3]을 통해 제안되었다. 이를 위한 패킷의 정의로 그림 1의 (b)와 같이 기존의 (a)의 뒷부분에 블록의 크기를 위한 항목(명칭: blksize)이 추가되었다.

그림 2는 TFTP를 통한 sender와 receiver 간 파일전송의 예를 보여준다. Sender는 최초 파일이름을 포함한 WRQ 패킷을 receiver에게 보낸다. WRQ 패킷을 받은 receiver는 패킷 내 명시된 파일명과 동일한 빈 파일을 생성 후 데이터를 받을 준비가 되었음을 알리는 의미로 ACK 패킷을 sender에게 회신한다. ACK 패킷을 수신한 sender는 파일의 내용을 블록의 크기만큼 데이터 패킷에 담고 receiver에게 송신 후 해당 데이터 패킷에 대한 ACK를 받는다. ACK를 받으면 다음 블록 데이터를 송신하는 기능을 반복적으로 수행한다. sender와 receiver 간 전송되는 데이터 패킷에 대하여 마지막임을 판단하는 기준은 블록의 크기이다. receiver는 수신한 데이터 패킷의 블록크기가 지정 크기보다 작은 경우 마지막 데이터 패킷으로 간주하고 이후 작성한 파일을 닫고 TFTP 기능을 종료한다.

UDP 세그먼트는 best effort 서비스에 의존하기 때문에 채널의 연결지향성이나 전송데이터의 신뢰성을 보장하지 않는다. 따라서 TFTP는 이를 보완하는 방안으로 각각의 송신데이터에 대한 ACK 메시지를 수신받는 구조로 동작한다. 일반적으로 ACK를 받기 위해 대기하는 시간은 $RTT \times 1.5$로 한다. 이 대기시간을 벗어난 경우 sender는 직전의 송신블록에 대해 loss로 간주하고 해당메시지를 재전송하는 방식으로 동작하여 최종적으로 전송된 파일에 대해 신뢰성을 보장한다.

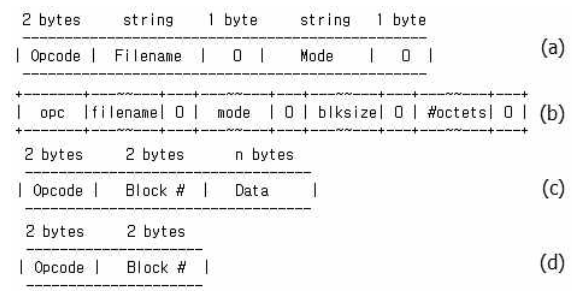

그림 1. TFTP 패킷 형태
Fig. 1. TFTP Packet Formats.

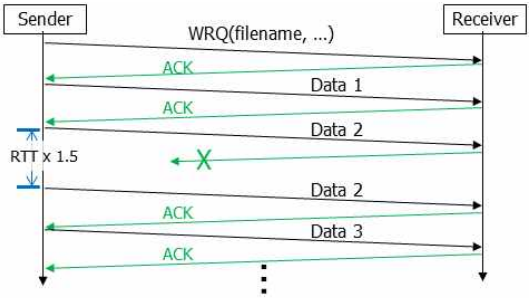

그림 2. TFTP 패킷 흐름 예
Fig. 2. TFTP Packet Flows Example.

Receiver는 수신한 블록데이터에 대하여 파일에 기록할 때, 각 데이터에 명시된 블록의 번호(Block ID)와 크기(Block Size)를 적용하여 파일에 오프셋(offset)을 설정 후 내용을 기록한다. 블록의 크기는 RFC-1350에서는 512로 정해져 있으나 이더넷 환경의 발전에 따라 RFC-2348에서 사용자가 임의의 값으로 지정할 수 있도록 재정의되었다.

수식 (1)은 receiver에서 파일의 오프셋 결정을 위한 표현식을 보여준다. BlockCycle은 증가되는 BlockId의 순환 횟수를 의미한다.

$$File\ Offset = BlokSize \bullet BlockId \bullet (BlockCycle - 1) \quad (1)$$

데이터 송신후 ACK를 받고 이후 작업을 수행하는 Stop-And-Wait 방식은 전송과정에서 많은 대기시간이 발생하여 결과적으로 수율(data throughput)이 낮아지는 단점을 갖는다.

TFTP의 단순한 전송기능에 따른 응용[4][5]과 많은 제약을 (예: 보안, 속도) 해결하기 위한 다양한 연구들[6][7]이 있으며 대표적으로 DTFTP[8]와 UDT[9]가 있다.

2-2 DTFTP

DTFTP(Dynamic Trivial File Transfer Protocol)[8]는 TFTP의 Stop-And-Wait 방식에 따른 속도저하를 해결하기 위해 TCP에서 사용되는 슬라이딩 윈도우(sliding window) 개념을 도입시켰다. 기본 설계를 위한 타깃을 무선랜(WLAN) 환경으로 간주하고 이곳에서 TFTP의 성능향상을 목표로 한다.

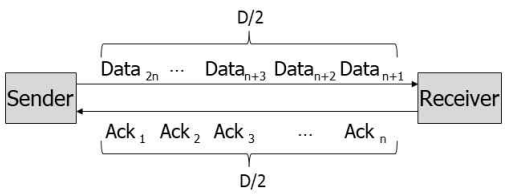

그림 3. DTFTP 패킷 흐름 예
Fig. 3. DTFTP Packet Flows Example.

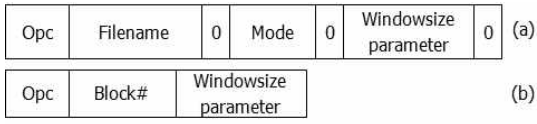

그림 4. DTFTP 메시지 형태
Fig. 4. DTFTP Message Formats.

무선랜 환경은 유선랜 대비 지연이 더 크고 네트워크의 상태 변화가 더 큰 점을 고려하여 sender와 receiver간 RTT (round trip time)값을 고려하여 윈도우 크기를 결정한다.

그림 3은 무선랜 환경에서 DTFTP를 통해 파일전송이 이루어질 때 RTT/2시간 동안 전송되는 데이터블록과 그에 맞는 ACK가 흐름사이에 존재하는 것을 보여준다. 그림에서는 윈도우의 크기를 n개로 했을 때의 한 예시가 되는 것이고, n의 값은 RTT값에 의해 산술된다.

DTFTP에서 사용되는 메시지는 receiver가 sender에게 파일을 요청하는 메시지(RRQ)와 확인 메시지(ACK)를 일부 개량하였고, 그 내용으로는 그림 4와 같이 윈도우크기 지정을 위한 항목(windowsize parameter)이 더 추가된다.

그림의 (a)는 DTFTP를 위한 RRQ 메시지의 형태를 보여준다. 최초 윈도우 크기를 설정하기 위한 항목으로 Windowsize Parameter 항목이 사용되고 (b)는 블록데이터에 대한 각각의 ACK 메시지의 형태로 이곳에서는 운영 중 윈도우 크기가 가변화되어 서로 간 윈도우 크기에 대해 동기를 맞춰 유기적으로 동작하기 위한 용도로 사용된다.

DTFTP의 시험은 유선랜(100M 이더넷) 과 무선랜(802.11b/g) 기반의 서로 다른 두 노드 간 5MB 크기의 파일전송을 통해 성능을 측정한다. 시험결과 각 무선의 링크상태에 따라 2~4개의 윈도우 크기로 운영했을 때 최상의 성능이 있었고, 최종적으로 TFTP대비 54% 속도 향상이 있음을 증명하였다. 메시지의 형태가 기존 TFTP와 달라 이 기능을 위해서는 두 쌍 간 오직 DTFTP를 통해서만 파일전송이 가능하며, 기존 TFTP와 호환될 수 없는 단점을 갖는다.

2-3 UDT

UDT (UDP-based data transfer)[9]는 TCP의 복잡한 동작구조와 헤더크기에 따른 페이로드(payload)의 용량감소 등을 극복하기 위해 UDP를 기반으로 연결지향성 신뢰성있는 논리적인 통신채널 지원하도록 미들웨어 형태로 구성되고 응용프로세스를 위한 전용의 소켓 인터페이스(UDT 소켓)를 제공한다. 그림 5는 UDT에 대한 계층적인 구조를 보여준다.

송/수신 데이터의 신뢰성을 제공하기 위해 acknowledgement 기법과 빠른 데이터 전송을 위해 슬라이딩 윈도우 개념을 함께 적용하였다. 그리고 접속 쌍 간 데이터 송/수신과정에서 측정된 RTT값을 이용하여 윈도우 내 송신데이터의 간격을 조절한다.

송/수신을 위한 메시지의 형태가 정해져 있으며 용도에 따라 데이터 패킷(data packet)과 제어(control) 패킷으로 정의된다.

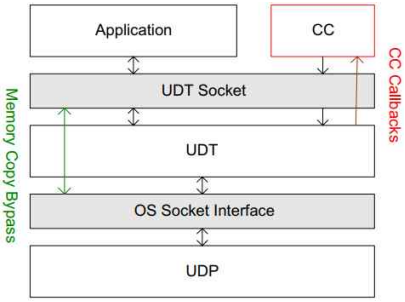

그림 5. UDT 계층적 구조도
Fig. 5. UDT in the Layer-ed Architecture.

데이터 패킷은 sender와 receiver간 데이터 송/수신을 위해 사용되고, 제어 패킷은 receiver와 receiver간 통신 채널의 제어를 위해 사용된다. 두 패킷에는 서로 간 송신시각을 측정하기 위해 타임스탬프(time stamp) 정보를 갖는다. 타임스탬프 정보를 활용하여 채널의 혼잡 및 흐름제어를 위해 윈도우내 전송메시지들의 송신간격을 조절한다. 흐름 제어를 위해 구성될 수 있는 알고리즘은 기존의 TCP 계열의 다양한 로직들로 구성될 수 있다는 게 큰 특징이다.

UDP를 기반으로 동작하기 때문에 전송되는 메시지의 크기에서부터 동일한 SDU에 대해 TCP대비 크기가 작다. 따라서 결과적으로 더 높은 데이터 전송률을 갖는다.

UDT는 별도의 미들웨어적인 특성과 임베디드시스템의 제약적인 상황을 고려하면 프로세스의 부피가 커지는 단점이 존재한다. 그리고 현재 많은 영역에서 파일전송을 위해 사용 중인 TFTP 와는 동작상의 원리가 근본적으로 다르므로 DTFTP처럼 쉬운 적용이 불가하다.

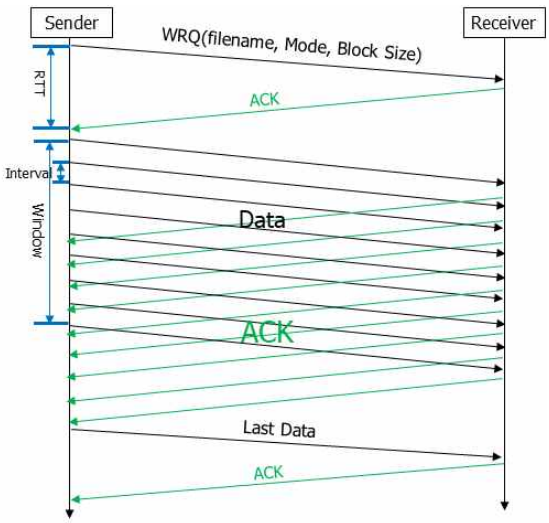

그림 6. QuTFTP의 데이터 흐름 예
Fig. 6. The Data Flows Example of QuTFTP.

Ⅲ. 제안 알고리즘

본 논문의 QuTFTP (quick UDP trivial file transfer protocol)는 기존의 TFTP[2][3]의 stop-and-wait 기법에 따른 낮은 데이터 전송률을 보완하기 위해 제안되었으며 기존 TFTP에서 정의된 패킷 형태의 변형 없이 사용한다. 그리고 receiver의 경우 기존의 TFTP의 수신처리와 동일한 동작을 갖기 때문에 데이터 수신부는 호환성을 갖는다.

QuTFTP는 기존 DTFTP[8]와 UDT[9] 등과 같이 데이터 송신 시 슬라이딩 윈도우 기법을 활용한다. 그리고 UDT처럼 흐름제어를 위해 메시지의 송신간격(interval)을 조절한다. 다만, 전송 패킷에 타임스탬프 등의 추가사항 없이 기존의 TFTP와의 동작 호환을 위해 정의된 패킷의 구조를 그대로 활용하는 게 이 QuTFTP의 특징이다.

그림 6은 QuTFTP를 통한 sender와 receiver 간 데이터 송/수신 흐름을 보여준다. 그림에서와같이 sender는 송신할 블록데이터들에 대하여 ACK 수신이 없더라도 receiver에게 윈도우 크기의 개수만큼 순차적으로 보낸다. 윈도우에는 각 송신된 블록메시지에 대하여 블록의 ID, 타임스탬프 그리고 재전송 횟수의 정보를 보관한다. 그리고 송신 블록메시지에 대해 대기시간 ($RTT \times 1.5$ 또는 1초(고정값))이내에 ACK를 수신하게 되면, 각 해당되는 블록을 윈도우에서 삭제한다. 대기시간 내 ACK가 수신되지 않으면 해당 블록메시지는 재전송을 수행하게 되며 최종적으로 이 재전송의 횟수가 3을 초과하게 되면 전송실패로 간주되고 이후 메시지전송을 모두 멈춘다. 윈도우에 대하여 비워진 공간(또는 슬롯, slot)이 존재하면 다음 블록메시지 송신을 위한 용도로 활용된다.

반면, 블록메시지를 수신한 receiver는 각각의 수신메시지에 대하여 기존의 TFTP와 동일한 방식으로 ACK를 회신하고 WRQ(write request)에서 정의된 파일에 블록을 기록한다.

Sender의 경우 송신 메시지에 대한 응답(ACK 수신) 속도에 따라 윈도우 내 점유된 슬롯의 비율이 달라진다. 송신간격 대비 ACK의 응답이 느린 경우 점유된 슬롯들에 대한 윈도우 내 비율이 점차 높아지게 되고 결국 포화상태에 도달하면 더는 송신이 불가하게 된다. 반면, ACK의 응답이 송신 간격보다 빠를 경우 윈도우내 점유슬롯의 비율은 점차 낮아지게 되고 결국 기존 TFTP와 같이 대기시간이 발생하게 된다. 송신간격은 블록메시지의 송신과 ACT의 회신 간의 RTT와 상당히 연관이 있다.

실제 네트워크 환경에서는 전송중인 블록메시지와 ACK 간의 RTT 변화가 상당히 크다. 따라서 운영 중 채널 상태를 꾸준히 확인하여 상황에 맞는 적절한 송신 간격을 부여하는 것이 가장 효율적일 것이다. 이를 위해 QuTFTP는 윈도우 내 점유슬롯의 비율을 감시(monitoring)하여 송신 간격을 변화시킨다.

그림 7은 QuTFTP에서 동작하는 송신 간격 규칙을 보여준다. 데이터 전송 중 윈도우 내 20%미만이 점유슬롯이 있을 때 기존 송신 간격에 0.9배를 적용하여 단위 시간당 전송횟수를 증가시킨다.

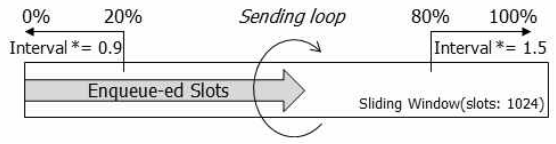

그림 7. QuTFTP의 송신간격 제어로직
Fig. 7. Sending Interval Decision of QuTFTP.

운영 중 80% 이상의 점유상태가 되면 송신 간격을 1.5배 증가시켜 데이터의 손실 등을 줄이고 채널을 안정화 한다. 20%에서 80% 사이에서는 고정된 송신간격을 갖게 되지만, 외부의 네트워크 점유상태에 따라 RTT의 변화 및 데이터 손실로 인해 점유율은 달라질 수 있다. 이를 적극적으로 반영하기 위해 이 구간을 줄이는 것보다 일부 완만한 흐름을 위해 60%의 정적구간을 할당하였다.

Ⅳ. 실험 및 결과

4-1 실험 환경

TFTP가 주로 임베디드시스템에서 활용되고 있음을 고려하여 표 1과 같이 Raspberry Pi라는 ARM 계열의 CPU를 탑재한 타깃보드(target board)를 receiver로 활용하였다.

파일 전송에는 일반적으로 파일시스템에 기록하는 작업(job)도 포함된다. 그러나 타깃보드의 경우 microSD를 탑재(규격: HC, 쓰기: 48MB/s, 읽기: 68MB/s)하고 있으며, 순수 1GB 파일에 대한 기록을 측정한 결과 30여 초의 시간이 소요되었다. 수신된 블록들에 대하여 파일로 기록하는 처리(write()함수 호출)까지 측정하기에는 커널(kernel)내 스케줄러 및 정책에 의해 시간상 상당한 가변폭이 발생되었다. TFTP를 통한 전송 결과 4,096 블록크기로 전송했을 때 211~243초가 소요되었고, 이는 7.048%의 오차 범위에 해당된다. 따라서 순수 블록메시지의 완전 전송에 걸리는 시간을 측정하기 위해 모든 수신블록들은 가상 파일(파일: /dev/null)에 기록처리 하였다.

블록의 크기별 RTT의 변화에 따른 전송속도를 측정하기 위해, 기존 TFTP에서 제공되는 가변 크기(512, 1024, 2048, 4096, 8192, 16384, 32768)로 전송하였다. 그리고 크기별 전송은 TFTP와 QuTFTP로 수행하였고, 각각의 결과를 비교하였다.

표 1. 실험 환경
Table 1. Test Environments.

Items	Operation Environments
sender	- H/W : Xeon E3-1240, 16G(RAM), 1G LAN - S/W : Kali Linux
receiver	- H/W : Raspberry Pi 3 B+, 1G(RAM), 100M LAN - S/W : Raspbian Linux 10
etc	- Gigabit Switch Hub (× 1) - Network Storage (× 1) - PC(× 6, general web-surfing)

4-2 실험 결과

표 2는 TFTP와 QuTFTP의 블록 크기별 1GB 파일을 전송하는데 걸리는 시간을 측정하여 보여주고 블록별 차이를 함께 나타내고 있다. 표에서 알 수 있듯이 QuTFTP의 경우 기존 TFTP에 비해 1.9~2.8 배 이상의 성능을 제공하는 것을 확인 할 수 있다. 그리고 그림 8은 표의 결과를 그래프로 보여주고 있다.

결과에서 알 수 있듯이 블록의 크기가 증가할수록 높은 성능을 갖는다. 블록의 잦은 전송은 많은 대기시간과 불필요한 헤더 정보(UDP, TFTP, QuTFTP Overheads)가 추가로 붙기 때문에 다수의 전송일수록 비효율적이다. 결과표를 통해 유추할 수 있듯이 송신 블록의 크기를 최대화할수록 더 높은 성능을 낼 수 있다. 그러나 단위 UDP 세그먼트당 수용할 수 있는 페이로드(payload)의 최대크기(65,535)의 제약으로 인해 64KB 이상은 단일 전송이 불가능하다.

본 시험을 위해 윈도우 크기는 20으로 고정 후 시험을 수행하였다. 그림 9는 32,768 크기를 갖는 블록데이터에 대하여 송신간격(Interval)의 변화에 따른 윈도우 내 ACK 대기 개수(Qsize)의 변화를 함께 보여준다. 윈도우 내 점유슬롯이 20%에 해당하는 4개 미만일 때, 송신간격이 짧아지고 80%에 해당하는 16개 이상일 때에는 송신간격이 길어져 결과적으로 데이터의 과한 전송에 따른 손실이 방지되었다.

표 2. 실험 결과
Table 2. Test Results.

Block Size	512	1024	2048	4096	8192	16384	32768
TFTP	989.27	518.8	314.71	173.3	106.37	69.83	52.69
QuTFTP	351.04	190.04	119.4	65	42.54	33.73	27.63
Ratio	2.818	2.730	2.636	2.666	2.500	2.070	1.907

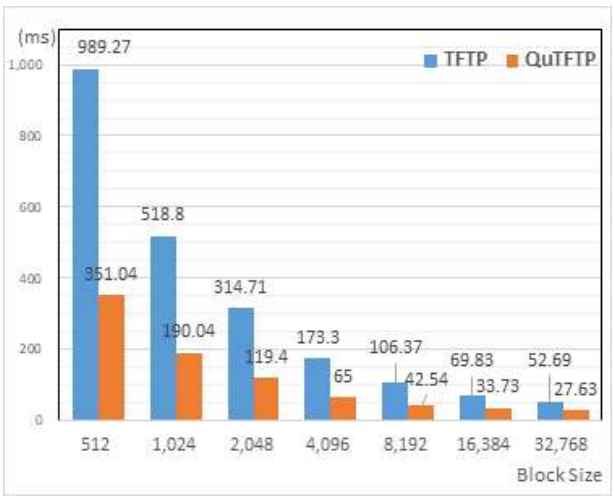

그림 8. 1GB 전송 완료시간
Fig. 8. Completion Time for transmission of 1GB.

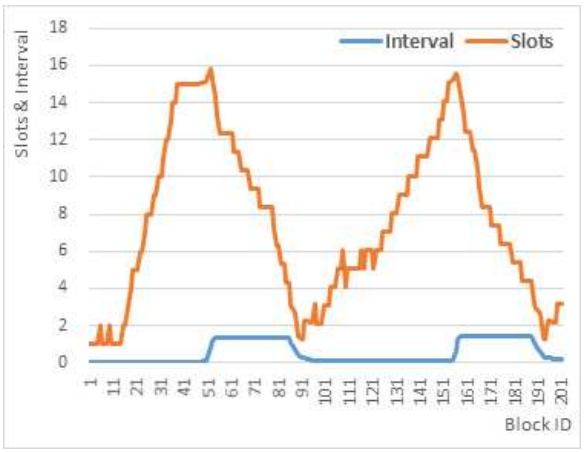

그림 9. 송신간격과 윈도우슬롯점유율 관계
Fig. 9. Relationship between Intervals and Window Slots.

그림에서 알 수 있듯 송신간격(파란색 선)의 변화를 통해 윈도우 내 점유슬롯이 제어되고 결과적으로 송신 버퍼에서 흐름이 제어됨으로써 데이터 손실을 방지함과 동시에 파일 데이터의 빠른 전송이 가능함을 알 수 있다.

Ⅴ. 결 론

본 논문에서는 기존의 TFTP의 Stop-And-Wait 방식에 따른 낮은 데이터 전송률은 대기시간의 단축을 위해 전송당 블록의 크기를 최대화하는 게 가장 효율적일 것이다. 본 논문에서는 기존 TFTP에서 블록 크기별 전송률도 측정하였으나 이 방식은 TFTP의 근본적인 문제를 해결하지 못한다.

본 논문에서는 이를 극복하기 위해 슬라이드 윈도우 기법의 적용과 윈도우의 점유슬롯을 모니터링하여 전체적으로 전송흐름을 제어하는 QuTFTP를 제안하였다. 그리고 TFTP와 동일한 조건에서 2~3배의 더 빠른 데이터 전송률을 가짐을 실험을 통해 입증하였다. 아울러, 기존 TFTP에서 정의된 고유의 메시지를 수정 없이 사용하기 때문에 본 제안은 DTFTP와 UDT와는 달리 호환성을 갖는다는 것이 특징이자 큰 장점이다.

References

[1] FTP : File Transfer Protocol, IETF RFC – 765.
[2] TFTP : Trivial File Transfer Protocol (Rev. 2), IETF RFC – 1350.
[3] TFTP Blocksize Option, IETF RFC – 2348.
[4] Anja Veslinovic *et al.*, “ARINC 615A data loader using ethernet interface for aircrafts,” *In Proceeding of Telecommunication Forum,* Vol. 26, Belgrade, Serbia, Nov. 2018.
[5] Sumit Jain et al., “IHMU: Remote installation of OS/ system software/ application software/ patch installation/ version anomaly detection and system health monitoring in distributed system,” *In Proceeding of International Conference on Computational Intelligence & Communication Technology(CICT),* Vol. 3, Ghaziabad, India, Feb. 2017.
[6] J. S. Park, M. K. Noh, and S. H. Kim, “Performance measurement and evaluation of open source based parallel transfer tools for end to end transfer efficiency maximization,” *The Journal of Korean Institute of Communications and Information Sciences,* Vol. 42, No. 10, pp.1999-2011, Oct. 2017.
[7] M. A. Mat, N.N. Mohamed and H. Hashim, “A lightweight and secure TFTP protocol for smart environment,” *In Proceedings of 2012 International Symposium on Computer Applications and Industrial Electronics*, Kota Kinabalu: Malaysia, pp. 302-306, Dec. 2012.
[8] J. Li, W. Yang, and G. Wang, “A cross layer design for improving trivial file transfer protocol in mobile transparent computing,” *In Proceedings of 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing,* Zhangjiajie: China, pp. 1872-1877, Nov. 2013.
[9] Y. Gu and R. L. Grossman, “UDT: UDP based data transfer for high-speed wide area networks,” *Computer Network*, Vol. 51, No. 7, pp. 1777-1799, May 2007.

김 병 국 (Byoung-Kug Kim)
2004년 8월 : 고려대 통신시스템기술협동 (공학석사)
2011년 8월 : 고려대 전자컴퓨터공학과 (공학박사)
2011년 9월 ~ 2013년 8월 : 동양미래대학 소프트웨어정보과 전임강사
2013년 8월 ~ 현재 : 대한항공 항공기술연구원 시스템개발팀 근무中
※관심분야 : 센서네트워크, 유비쿼터스컴퓨팅, 임베디드시스템, 운영체제, 네트워크 미들웨어