논문 20-45-09-12 The Journal of Korean Institute of Communications and Information Sciences '20-09 Vol.45 No.09
https://doi.org/10.7840/kics.2020.45.9.1570

MPTCP 스케줄러 연구 동향 고찰과 문제점 분석

김 건 환•, 박 창 훈*, 김 정 근*, 엄 원 주*, 송 영 준*, 서 원 경**, 조 유 제°

MPTCP Scheduler Research Trends and Problem Analysis

Geon-Hwan Kim•, Chang-Hoon Park*, Jeong-Keun Kim*, Won-Ju Eom*, Yeong-Jun Song*, Won-Kyeong Seo**, You-Ze Cho°

요 약

최근 고속 이동통신에 대한 높은 요구를 충족시키기 위해 모바일 장치 내에는 다중 네트워크 인터페이스가 장착되어 있다. IETF (Internet Engineering Task Force)는 다중 네트워크 인터페이스를 동시에 사용하기 위해 MPTCP (Multipath TCP)를 제안했다. 여러 경로를 사용하는 MPTCP는 최상의 단일 경로를 사용하는 TCP보다 더 나은 성능을 제공해야 한다. 하지만, MPTCP에 대한 많은 연구로부터 이종 네트워크 환경에서는 MPTCP의 처리량이 최상의 단일 경로를 사용하는 단일 TCP의 처리량보다 나쁠 수 있다는 것이 밝혀졌다. MPTCP의 성능은 종단 간 지연, 경로의 대역폭, 수신 버퍼의 크기, 손실률 및 혼잡상황과 같은 다양한 네트워크 특성에 의해 제한될 수 있다. MPTCP 스케줄러는 이러한 네트워크 특성을 기반으로 어떤 서브플로우로 패킷을 얼마나 보낼지 결정하는 역할을 한다. 본 논문에서는 현재까지 소개된 MPTCP 스케줄러를 분석하고 이들의 장점과 한계에 대해 논의하며, 우수한 MPTCP 스케줄러 설계를 위한 권장 사항을 제시한다.

키워드 : 다중 경로 TCP, 스케줄링, MPTCP 스케줄러, 왕복 지연 시간, 수신 버퍼
Key Words : Multipath TCP, scheduling, MPTCP scheduler, Round trip time, Receiver buffer

ABSTRACT

Mobile devices are equipped with multiple network interfaces to meet the increasing demand for high-speed mobile communication. The Internet Engineering Task Force (IETF) has proposed Multipath TCP (MPTCP) to simultaneously utilise multiple network interfaces. MPTCP should provide better performance using a single best-path TCP. However, many studies on MPTCP have revealed that under heterogeneous network conditions, the throughput performance of MPTCP is not as good as that of the single best-path TCP. MPTCP performance is constrained by different network characteristics such as the difference in end-to-end delays, path bandwidths, receiver buffer size, loss rate, and congestion. How to send packets on the basis of these parameters is the decision of the MPTCP scheduler. In this paper, we analyse the existing MPTCP schedulers, discuss their advantages and limitations, and propose recommendations for a better MPTCP scheduler design.

※ This research was supported by Next Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. NRF-2017M3C4A7083676) and in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2018R1A6A1A03025109) and in part by supported by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1A2C1006249).
• First Author : School of Electronics Engineering, Kyungpook National University, kgh76@ee.knu.ac.kr, 학생회원
° Corresponding Author : School of Electronics Engineering, Kyungpook National University, yzcho@knu.ac.kr, 종신회원
* School of Electronics Engineering, Kyungpook National University, {pch4495, kjg818, dnjswn9612, syj5385}@knu.ac.kr, 학생회원
** Department of Military Electronic Communication, Yeungjin University, wkseo@yju.ac.kr
논문번호 : 202003-060-B-RN, Received March 20, 2020; Revised June 12, 2020; Accepted June 22, 2020

Ⅰ. 서 론

최근의 스마트폰은 Wi-Fi와 이동통신 네트워크를 동시에 사용하여 데이터 전송량을 늘릴 수 있다. <그림 1>은 이러한 다중 인터페이스/경로의 사용에 대한 시나리오를 나타낸다.

IETF (Internet Engineering Task Force)에서 제안한 MPTCP (Multipath TCP)[1]는 TCP에서 지난 10년 동안의 가장 큰 변화로, 신뢰성 및 다운로드 속도 향상, 대기시간 단축, 비용 절감 그리고 끊임없는 이중 연결을 제공하여 처리량을 향상시킬 수 있다. 현재 삼성 다운로드 부스터[2], KT GiGA Path[3], Apple Siri[4]와 같은 상용 제품에서 다중 경로의 동시 사용을 지원하고 있다. 또한, 최근의 연구 개발을 통해 파일 다운로드, 웹 브라우징, 스트리밍과 같은 다양한 응용 프로그램에서도 다중 경로 연결을 활용하고 있다.

MPTCP를 사용하면 단일 데이터 스트림을 여러 경로로 나눠서 전송할 수 있다. 데이터 스트림을 분할하여 호스트 간의 다중 경로 (서브플로우)를 통해 전송함으로써 자원 활용을 극대화하고 종단 간 처리량을 높일 수 있다. 또한, 두 개 이상의 경로를 사용함으로써 링크 또는 네트워크의 문제에 대한 빠른 대처와 견고함을 제공하고, 멀티 홈 네트워크에서는 통신 인터페이스를 변경하는 동안 끊임없는 연결을 지원할 수 있다.

하지만 MPTCP는 다중 경로를 이용하기 때문에 기존 단일 경로 TCP에서 주로 다뤄지던 혼잡제어, 흐름제어, 오류제어와 관련된 이슈 외에도 서브플로우 간에 새로운 문제들이 발생한다. 서브플로우별로 전송되는 세그먼트에 대한 순서 번호를 새롭게 부여하고 이를 토대로 수신 측에서 전체 데이터 스트림을 재순서화해야 하며, 서브플로우 간의 패킷 스케줄링, 단일 경로 TCP와의 공평성 등을 고려해야 한다. MPTCP의 설계 목표는 아래와 같다[1,5]:

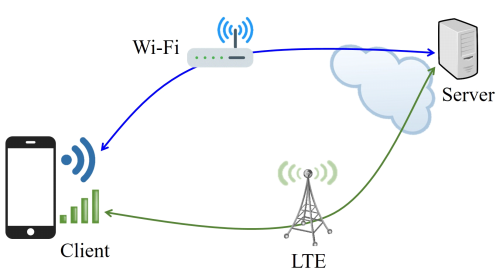


그림 1. 다중 인터페이스/경로 시나리오 예
Fig. 1. Example of multiple interface/path scenario

Fairness: MPTCP 연결 내의 여러 서브플로우가 단일 TCP 플로우와 병목 링크를 공유할 때, MPTCP 연결의 총처리량은 단일 TCP와 같아야 한다.

Performance: MPTCP가 사용하고 있는 경로 중에서 최상의 경로를 단일 TCP 플로우가 사용할 때의 처리량보다 MPTCP의 모든 서브플로우의 처리량 합이 더 높아야 한다. 이 목표를 충족해야만 MPTCP의 배치에 대한 동기가 보장된다.

Efficiency: MPTCP는 다수의 경로 중 더욱 효율적인 경로를 선호해야 하며 혼잡이 적은 경로를 통해 더 많은 트래픽을 전달해야 한다.

현재 MPTCP는 리눅스 커널에 구현되어 있으며[6], 이를 이용한 다양한 분석과 시뮬레이션들이 스마트폰 (Apple iOS7) 및 데이터 센터와 같은 실제 사용 사례와 함께 문헌에 소개되었다[7]. 그러나 테스트베드와 시뮬레이션 실험으로부터 기대에 미치지 못하는 낮은 총처리량과 트래픽 분배의 불균형이 보고되었고[8], 최악의 경우에는 이종 네트워크 (heterogeneous network)에서 MPTCP 연결의 총처리량이 최상의 단일 경로를 사용하는 TCP 연결의 처리량보다 더 낮을 수 있다[9].

하나의 MPTCP 연결에 다수의 이종 경로가 존재하는 경우에는 몇 가지 문제가 발생할 수 있다. MPTCP 연결 내의 모든 서브플로우는 수신 측에서 하나의 수신윈도우를 공유한다. 만약 다수의 경로가 서로 다른 종단 간 지연 시간을 가지면 데이터 패킷은 수신 단에 순서가 맞지 않게 전달될 수 있다[10]. 즉, 빠른 서브플로우를 통해 이미 수신 측에 도착한 데이터는 느린 서브플로우로 전달 중인 데이터가 수신 측에 도달하기까지 버퍼에서 대기해야 한다. 이와 같은 상황이 발생하면 수신 측에서의 버퍼 요구량은 증가하고, 패킷이 전송 중에 손실된다면 손실된 패킷을 성공적으로 수신할 때까지 응용 계층으로의 전달이 더욱 지연된다[11]. 이 문제는 수신 측의 버퍼가 충분하지 않고 제한적일 때 더욱 악화된다[12].

이러한 문제를 해결할 수 있는 스케줄러의 설계는 MPTCP의 성능에 영향을 미치는 중요한 요소이다[13]. MPTCP 스케줄러는 특정 스케줄링 정책에 따라 트래픽을 여러 경로로 분배하는 역할을 한다. 이는 TCP 혼잡제어의 역할과는 다르며, 데이터를 전송할 서브플로우를 고르고 각 서브플로우로 전송할 데이터의 양을 결정하는 것이다. 따라서, MPTCP 배치를 통해 최대의 성능 향상을 얻기 위해서는 스케줄러와 혼잡제어를 함께 고려해야 하며, 이를 조정하는 송신 측의 역할이 중요하다[14].

<그림 2>는 MPTCP와 스케줄러의 추상적인 구조를 나타낸다. 응용 계층의 관점에서 MPTCP는 단일 표준 TCP 소켓을 사용하는 것으로 여겨진다. 그러나 하위 스택에는 기존의 TCP 연결처럼 동작하는 다수의 서브플로우가 존재한다[15]. 즉, 응용 프로그램은 MPTCP를 구현하기 위해서 별도의 수정을 할 필요가 없다.

현재까지 많은 MPTCP 스케줄러가 제안되었지만, 이들의 고려사항과 목표는 굉장히 다양하다. 우수한 스케줄러를 설계하기 위해서는 기존 스케줄러의 장단점을 먼저 이해해야 한다. 따라서, 본 논문에서는 먼저 스케줄러 설계에 있어 중요한 네트워크의 매개변수 및 경로 특성에 대해서 알아본다. 이어서 기존의 MPTCP 스케줄러에 대한 포괄적인 검토를 제공하고 이들의 장점 및 단점에 대해 논의한다. 이러한 검토를 토대로, 우수한 MPTCP 스케줄러를 설계하기 위한 권장 사항을 제시한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 MPTCP의 스케줄링 매개변수와 MPTCP의 성능을 결정하는 스케줄러의 역할, 그리고 MPTCP 스케줄러가 해결해야 하는 과제에 대해 논의한다. Ⅲ장에서는 기존에 소개된 MPTCP 스케줄러의 동작 원리와 특징에 대해서 살펴본다. Ⅳ장에서는 비교 연구들을 소개하고 우수한 MPTCP 스케줄러 설계를 위한 권장 사항을 제시한다. 마지막으로, Ⅴ장에서 본 논문의 결론을 맺는다.

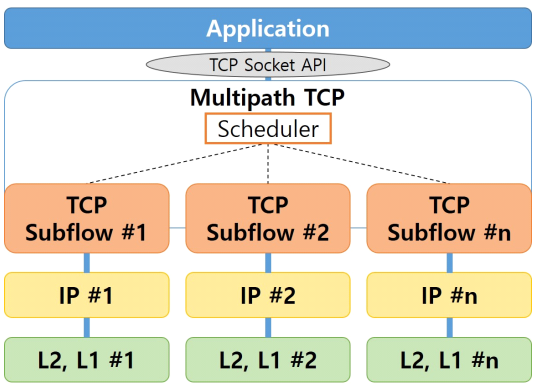


그림 2. MPTCP의 구조
Fig. 2. The architecture of MPTCP

Ⅱ. MPTCP 스케줄링에서의 문제점과 매개변수

Out-of-order 도착과 HoL (Head-of-Line) 블로킹은 MPTCP 스케줄러가 피해야 할 심각한 상황들이다. 본 장에서는 이 상황들을 묘사하기 위한 두 가지의 시나리오를 제시한다. 또한, 스케줄링 결정에 사용되는 매개변수를 살펴보고 이들이 MPTCP의 성능에 미치는 영향을 알아본다.

2.1 MPTCP 스케줄링

MPTCP 연결은 응용 계층의 버퍼로부터 단일 데이터 스트림을 받아서 여러 서브플로우로 분배한다[16]. 서브플로우의 혼잡윈도우 (cwnd)에 여유가 있는 경우, 스케줄러의 기능은 <그림 3>과 같이 요약된다[17]. 각 서브플로우는 독립적인 하나의 TCP 연결처럼 동작하며 혼잡 및 오류제어와 같은 경로 관리 기능을 가진다.

MPTCP 연결은 데이터 레벨과 서브플로우 레벨의 두 순서 번호를 유지 및 관리한다. 응용 프로그램은 데이터 레벨의 다중 경로 송신 버퍼에 데이터를 전달하고, 스케줄러는 응용 프로그램으로부터의 데이터를 전달할 서브플로우를 지정한다. 이어서, 데이터 레벨 순서 번호를 사용하여 선택된 서브플로우를 통해 신뢰할 수 있는 순서대로 데이터를 전송하고, 서브플로우 레벨 순서 번호는 각 서브플로우의 흐름제어를 위해 사용된다. 각 서브플로우는 일반적인 TCP 연결과 동일한 방식으로 할당된 데이터를 전송한다. <그림 4>는 MPTCP의 데이터 전송 과정을 나타낸다. MPTCP 연결의 모든 서브플로우는 하나의 수신 버퍼를 공유하고 동일한 수신윈도우 (rwnd)를 광고한다[1].

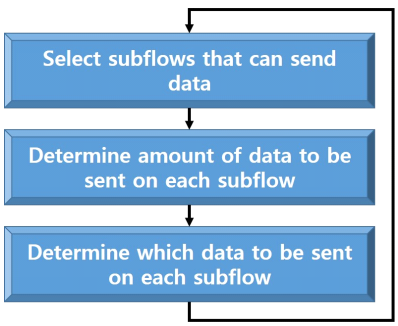


그림 3. MPTCP 스케줄러의 기능
Fig. 3. The function of MPTCP scheduler

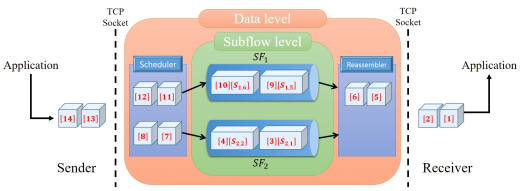


그림 4. MPTCP의 데이터 전송 과정
Fig. 4. Data transfer process of MPTCP

각 서브플로우 간의 왕복 지연 시간 (Round Trip Time, RTT)이나 단방향 지연 (One Way Delay, OWD)이 다를 경우, 송신 측에서 순서대로 전송된 패킷이 서로 다른 시간에 수신 측에 도착하여 out-of-order 문제를 일으킨다. MPTCP 스케줄러의 주요 목표는 패킷이 가능한 낮은 종단 간 지연으로 송신 측에서 전송된 순서대로 수신 측에 도착하도록 하는 것이다. 경로의 특성을 고려하지 않은 스케줄링은 성능 저하를 초래할 수 있으므로 스케줄러는 각 경로의 특성을 정확하게 알아야 한다. 또한, 지연 시간의 잦은 변화는 스케줄러가 이에 신속하게 대응하지 못하기 때문에 스케줄링을 더욱 어렵게 한다.

2.2 MPTCP 스케줄링에서의 문제점

MPTCP 스케줄링의 개요에 이어서 MPTCP 성능 저하의 주요 원인인 out-of-order 도착과 HoL 블로킹 문제에 대해 살펴본다.

2.2.1 Out-of-order 도착 문제

다수의 서브플로우가 서로 다른 지연 시간을 가지면 패킷이 송신 측으로부터 전송된 순서와 다르게 수신 측에 도착할 가능성이 증가한다[10]. 기본 MPTCP 구현에서는 서브플로우의 RTT 차이에 따라 이러한 비순차적 도착 문제가 발생한다.

MPTCP 연결에서 발생할 수 있는 out-of-order 문제에 대한 시나리오를 다음과 같이 고려한다[18].

SF1: OWD=100ms, cwnd=4	Scheduler=minRTT
SF2: OWD=10ms, cwnd=4	

<그림 5>에서 SF2는 4개의 전송 가능한 데이터 (1-4)를 가지고 있으며, (5-8) 데이터 패킷은 SF1을 통해 보낼 준비가 되어 있다. 사용 가능한 cwnd를 가득 채워서 SF2로 전송한 후 나머지 (5-8) 패킷은 SF1로 전송된다. 응용 프로그램에서 34까지의 데이터 패킷이 전송될 준비가 되었다고 가정하면, MPTCP의 기본 스케줄러의 스케줄링 정책에 따라 (9-34) 패킷은 모두 SF2로 전송된다. 하지만, SF2를 통해 수신 측에 도달한 (1-4)와 (9-34) 데이터 패킷은 SF1을 통해 전달 중인 (5-8) 패킷이 수신되지 않은 상태이므로 out-of-order 도착 패킷으로 간주되어 (5-8) 패킷이 수신될 때까지 응용 계층으로 전달될 수 없다.

만약 응용 계층에서 보낼 준비가 된 (1-8) 데이터 패킷이 존재한다면, 이 8개 패킷의 단방향 총 전달 지연은 100ms가 된다. 하지만 (5-8) 패킷 또한 SF1이

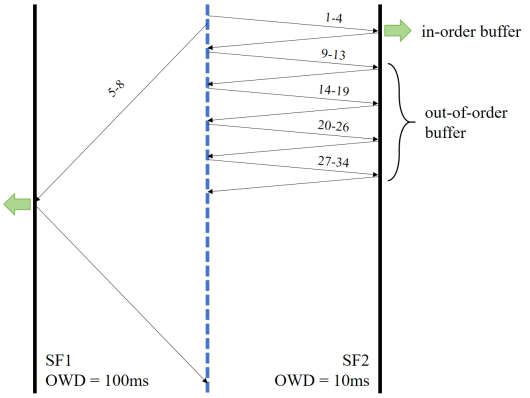


그림 5. Out-of-order 도착 문제[18]
Fig. 5. Out-of-order arrival problem[18]

아닌 SF2로 할당된다면, (1-4) 패킷 전송에 대한 응답을 기다린 후 (5-8) 패킷을 전송하여 단방향 총 전달 지연은 30ms가 소요된다.

반면, (1-4) 패킷과 (5-30) 패킷을 모두 SF2로 전송하고 (31-34) 패킷을 SF1로 보낸다면 단방향 총 전달 지연은 100ms로 기존과 같지만, 모든 데이터 패킷은 MPTCP 수신 측에 순서대로 도착하기 때문에 기다릴 필요 없이 응용 계층으로 전달된다.

위의 시나리오로부터 경로의 지연 시간을 고려하지 않은 스케줄링은 out-of-order 문제와 그로 인한 긴 종단 간 지연을 유발하는 것을 알 수 있다. 우수한 MPTCP 스케줄러는 수신 측에 패킷이 순서대로 도착할 수 있도록 전달할 패킷의 양과 각 서브플로우의 전파 지연 차이에 대한 영향을 파악한 뒤 적절한 순서로 스케줄링을 수행해야 한다.

2.2.2 Head-of-Line 블로킹 문제

패킷 재전송은 종종 수신 측에 버퍼 블로킹을 일으켜 처리량 저하를 유발한다[19]. 이를 HoL 블로킹 문제라고 한다[11]. <그림 6>의 SF2를 통해 전송된 데이터 패킷 1이 전송 중에 손실된다고 가정하면, 수신 측 버퍼는 패킷 1이 재전송되어 수신 측에 도착할 때까지 다른 패킷을 수신할 수 없는 상태가 된다. 따라서 시

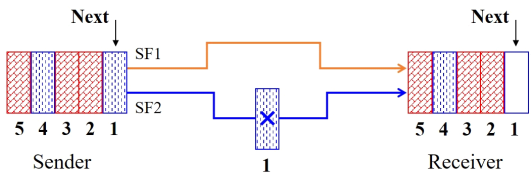


그림 6. Head-of-Line 블로킹[20]
Fig. 6. Head-of-Line Blocking[20]

스템의 전체적인 처리량이 저하된다. 이 외에도 송신 버퍼의 가장 앞에 위치하는 데이터 패킷이 상대적으로 느린 경로를 통해 전송된다면 이 패킷에 대한 ACK 메시지를 수신할 때까지 오랜 시간이 걸리기 때문에, 송신윈도우는 일시적으로 증가할 수 없어서 HoL 블로킹과 유사한 상황이 발생한다. HoL 블로킹 중에 송신윈도우 내의 모든 데이터 패킷이 이미 전송 중인 상태이면 빠른 경로로의 새로운 할당 또한 이루어질 수 없어서 성능이 감소한다.

2.3 MPTCP 스케줄러의 매개변수

2.3.1 왕복 지연 시간 (RTT)

RTT는 패킷이 송신되는 시점과 해당 패킷에 대한 ACK가 돌아오는 시점 사이의 시간을 반영한다. 리눅스에 구현된 MPTCP 커널은 기본 스케줄러로 minRTT 스케줄러를 사용하는데, 이 스케줄러는 다음 패킷을 전송할 때 사용 가능한 가장 빠른 경로를 선택한다. 하지만 RTT 기반 스케줄러는 MPTCP 혼잡제어 알고리즘의 목표와 일치하지 않을 수도 있다. 스케줄링 요소로 오직 RTT 만을 사용하는 것은 패킷 손실을 정확하게 예측할 수 없으므로 RTT 샘플은 완전히 신뢰할 수 있는 경로 특성 또는 용량 측정 방법이 아니다. 특히, 고속 네트워크에서 RTT 샘플은 너무 거칠어서 bursty 혼잡을 정확하게 측정하기 어려울뿐더러[21], 손실과 관련 없는 일시적인 대기열 변동은 샘플 RTT를 변경하여 라우터의 혼잡상황에 대한 평가의 신뢰성을 떨어뜨린다. 따라서, 우수한 스케줄러는 RTT 뿐만 아니라 네트워크의 혼잡상황에 근거하여 서브플로우를 선택해야 한다[22].

2.3.2 단방향 지연 (OWD)

RTT 샘플 측정값이 항상 대칭적인 것은 아니며 네트워크 내의 혼잡은 때때로 단방향으로 발생한다. 또한, 비대칭 라우팅[23,24] 그리고 LTE 및 Wi-Fi와 같은 무선 네트워크에서의 동작은 RTT의 비대칭을 초래한다. 따라서, 스케줄링을 위한 요소로 단순히 RTT 만을 사용하는 것보다 OWD를 사용하는 것이 더욱 유용할 수 있다. 특히, 넓은 대역폭이 필요하지 않은 지연에 민감한 응용 프로그램의 경우에는 최소 RTT를 이용해서 스케줄링하는 것보다 OWD를 이용해서 스케줄링할 때 더 많은 이점을 얻을 수 있다. 일반적으로 OWD를 계산하기 위해서 송신 측과 수신 측의 time stamp를 이용할 수 있지만, 일부 스케줄러는 응용 프로그램이 지연에 크게 민감하지 않은 경우에는 $RTT/2$를 OWD로 사용한다.

2.3.3 버퍼 크기 (buffer size)

서브플로우 간의 RTT가 서로 다른 경우엔 MPTCP 수신 측의 버퍼 요구량이 기존 TCP보다 증가한다[25]. 수신 측 버퍼 크기가 충분하지 않으면 MPTCP의 총 처리량이 저하될 수 있다. 따라서, 실제 네트워크에 MPTCP가 배치되기 위해서는 버퍼 크기에 의한 처리량 저하 문제를 분석하고 해결해야 한다. 일반적인 단일 경로 TCP에서 버퍼 크기에 의해 제한되는 최대 전송 중인 바이트의 수는 최대 대역폭을 활용하기 위해서 BDP (Bandwidth-Delay Product)에 도달해야 한다. MPTCP가 통합 대역폭을 최대로 활용하려면 데이터 레벨에서 전달 중인 최대 바이트의 수가 최대 (1)이 되어야 한다.

$$2\times\sum_{i=1}^{n}BW(i)\times\max_{i=1,..,n}RTT(i) \tag{1}$$

위의 식에서 $BW(i)$와 $RTT(i)$는 서브플로우 i의 대역폭과 RTT이며, n개의 서브플로우가 존재한다[26]. 식 (1)로부터 대역폭이 합산되고 최대 RTT가 고려되므로, MPTCP에 필요한 버퍼 크기는 단일 경로 TCP보다 더 크다는 것을 알 수 있다. Wi-Fi 또는 LTE 대역폭 통합과 같은 이종 무선 환경에서는 채널 품질이 낮은 하나의 무선 링크가 다른 무선 링크보다 더 높은 RTT를 가질 수 있어서 버퍼 요구량은 더욱 증가한다[27,28].

일반적으로 모바일 기기는 메모리 용량이 제한적이므로 최대 통합 대역폭을 달성할 수 있을 만한 충분한 버퍼 크기를 제공하지 못할 수 있다. 버퍼 크기가 충분하지 않아서 out-of-order 패킷이 삭제된 경우에는 이 패킷에 대한 재전송이 필요하다. 또한, 더 많은 양의 out-of-order 패킷이 수신 측에 축적된다면 일부 서브플로우의 cwnd가 일정 시간 동안 제한될 수 있다.

2.3.4 혼잡윈도우 (cwnd)

MPTCP의 스케줄링은 혼잡제어 알고리즘에 의해 제한될 수 있다는 점에 유의해야 한다. MPTCP 혼잡제어의 목표는 사용 가능한 경로 용량에 최대한 일치하도록 cwnd 크기를 동적으로 조절하여 최적의 혼잡 균형 및 우수한 응답성을 제공하는 것이다[29]. 하지만 네트워크의 품질이 항상 변화하기 때문에, 각 서브플로우가 이용 가능한 경로 용량을 정확하게 추정하는 것은 어려운 일이다. 초기 MPTCP는 coupled 혼잡제

어 알고리즘을 사용하여 다른 TCP 연결과의 공정성을 유지했으나, coupled 혼잡제어는 네트워크의 변화에 대해서 응답성이 느린 편이다[30].

coupled 혼잡제어[29,30]는 혼잡 회피 구간의 증가 부분에만 적용되며 slow start, 빠른 재전송 및 복구 알고리즘은 기존 TCP와 같다. 손실이 발생하게 되면 이를 감지하기 전에 서브플로우의 cwnd는 일시적으로 경로의 가용 용량을 초과할 수 있으며, 손실을 감지한 후에는 cwnd가 감소하여 새로운 slow start threshold 값이 경로의 가용 용량보다 낮을 수 있다. 또한, 가용 용량이 큰 경로의 경우에는 손실 후 cwnd가 최대 용량에 도달하는데 긴 시간이 걸리기 때문에 경로의 가용 용량 예상 수치로써 cwnd 또는 slow start threshold 값을 사용하는 것은 정확하지 않을 수 있다.

MPTCP 스케줄러 설계에 있어 앞서 소개된 다양한 경로 특성 중에서 서브플로우 간의 전파 지연 차이는 MPTCP의 성능에 상당한 영향을 미칠 수 있는 중요한 매개변수이며, 두 서브플로우가 서로 다른 RTT를 갖는 경우에는 MPTCP 연결의 전체 성능이 크게 저하될 수 있다[31]. 따라서, 스케줄러는 서브플로우 간의 지연 차이에 가장 주의해야 한다. 버퍼 요구량에 대한 실험에서는 일정한 크기 후에 수신 측 버퍼의 크기가 증가해도 MPTCP 성능이 향상되지 않는 것으로 나타났으며, 서브플로우의 개수가 많을수록 최적의 버퍼 크기를 찾는 것이 중요하다고 밝혔다[32]. cwnd 크기는 MPTCP 경로의 용량 추정에 사용될 순 있지만, 손실과 혼잡에 의해 자주 변화하기 때문에 추정의 정확도가 낮다. 이보다는 현재와 미래의 cwnd 크기를 추정하는 것이 MPTCP 경로의 용량 추정의 정확도를 높일 수 있다.

Ⅲ. 대표적인 MPTCP 스케줄러

리눅스 MPTCP 커널에서 minRTT 스케줄러가 기본 스케줄러로 적용된 후, 성능 개선을 위해 많은 스케줄러가 제안되었다. 본 장에서는 현재 문헌을 통해 소개된 대표적인 MPTCP 스케줄러들이 첫 번째 서브플로우를 선택할 때 어떠한 요소를 기준으로 스케줄링을 결정하는지와 그 특징에 대해서 살펴본다. <표 1>은 서브플로우 선택을 위한 주요 스케줄링 요소와 기준을 나타낸다.

1) minRTT (default): 리눅스에 구현된 MPTCP 커널은 기본 스케줄러로 minRTT 스케줄러[8,33]를 사용한다. RTT가 가장 낮은 서브플로우의 cwnd가 가득 찰 때까지 데이터를 할당한다. minRTT를 갖는 서브플로우의 cwnd가 가득 차면, 다음으로 낮은 RTT를 갖는 서브플로우를 통해 전송을 시작한다. 즉, 서브플로우의 RTT와 cwnd만을 고려하는 비교적 간단한 알고리즘이다. 하지만, 이종 네트워크 환경에서 minRTT 스케줄러를 사용했을 때 MPTCP 연결의 총처리량이

표 1. MPTCP 스케줄러의 스케줄링 요소, 기준 및 목표
Table 1. Scheduling elements, criteria and goals of MPTCP scheduler

스케줄러	스케줄링 요소	스케줄링 기준	스케줄링 목표
minRTT [8,33]	RTT, cwnd	RTT가 가장 낮은 서브플로우는 무엇인가?	처리량 향상
round robin[26]	cwnd	모든 서브플로우에 동등하게 패킷이 분배되었는가?	학문/실험 목적
DAPS [34,35]	OWD, RTT	수신 버퍼의 점유 기간을 어떻게 줄일 수 있는가?	HoL 블로킹 완화
BLEST [36]	outstanding packet, RTT	느린 서브플로우에 패킷을 얼마나 분배할 것인가?	HoL 블로킹 완화 재전송 최소화
ECF[37]	RTT, cwnd, 송신버퍼 크기	다음 패킷을 가장 빨리 보낼 수 있는 서브플로우가 무엇인가?	빠른 서브플로우의 이용률 최대화
DAPB[38]	RTT	MPTCP의 성능이 단일 TCP 성능보다 낮은가?	최소한 TCP 이상의 성능 보장
Path capacity based[22]	추정 용량, outstanding packet	다음 패킷 전송이 경로의 혼잡을 초래하는가?	처리량 향상
HSR[39]	cwnd, MSS, RTT	가장 높은 전송률을 갖는 서브플로우는 무엇인가?	처리량 향상
LWS[39]	outstanding packet, cwnd	cwnd에 가장 여유가 있는 서브플로우는 무엇인가?	처리량 향상
DEMS [40]	optimal chunk size	수신 측에 순서대로 도착하기 위한 chunk 크기는 얼마인가?	HoL 블로킹 완화 완료 시간 단축
Adaptive [31,41]	packet loss rate	모드 간 전환이 언제 이루어지는가?	처리량 향상 재전송 최소화
Load balance [42]	scheduling weight	최대 성능을 얻기 위한 weight 값이 얼마인가?	부하균형 최적화
Energy efficient [43]	NIC power consumption	어떻게 데이터 전송을 빨리 끝낼 수 있는가?	에너지 소모 최소화

최상의 단일 경로를 사용하는 TCP 연결의 처리량보다 더 낮을 수 있음이 보고되었다[8].

2) round robin: round robin 스케줄러[26]는 각 서브플로우의 cwnd 크기만을 고려하여 모든 서브플로우에 데이터를 순차 순환 방식으로 스케줄링한다. 즉, 전송할 데이터가 있으면 순차적으로 이용 가능한 cwnd가 있는 서브플로우에 데이터를 할당한다. 이 방식은 RTT나 손실과 같은 경로의 특성을 전혀 고려하지 않기 때문에 비교적 낮은 성능을 갖는다. 특히, 서브플로우 간의 지연 차이가 큰 경우에 HoL 블로킹, out-of-order 문제가 발생하고 경로를 효율적으로 사용하지 못한다.

3) delay aware packet: delay aware packet 스케줄러[34]는 패킷이 전송된 순서와 최대한 동일하게 패킷을 수신할 수 있도록 각 서브플로우의 단방향 지연에 근거하여 신중하게 패킷을 스케줄링한다. 경로 용량이 $\{c_1,c_2,...,c_n\}$이고 지연이 $\{d_1,d_2,...,d_n\}$인 사용 가능한 모든 서브플로우 n개에 대해 $lcm^2(d_i \in d_1,d_2,...,d_n)$ 시간 동안 전송할 N개의 패킷을 추정한다.

$$N = \sum_{i=1,2,...n} lcm(d_j \in d_1,d_2,...,d_n) \times \frac{c_i}{d_i} \quad (2)$$

이 스케줄러는 수신 측 버퍼의 점유율이 낮게 유지될 수 있도록 사용 가능한 경로에 N개의 패킷을 스케줄링한다. 수신 측을 향한 전방 지연에 대한 계산은 (3)과 같이 표현할 수 있다.

$$d_f = T_r - T_e - T_s \quad (3)$$

T_r은 수신 측에서의 time stamp, T_s는 송신 측에서의 time stamp, T_e는 SACK 메시지를 전송하기까지 수신 측에서 걸리는 시간이다.

후속 연구[35]에서는 수신 측 버퍼에서 패킷에 의해 소요되는 총 시간을 추정하고 이를 최소화하여, 앞선 방안[34]을 개선한 DAPS 알고리즘을 제안하였다. 이어서 [18]에서는 스케줄러가 서브플로우를 통해 패킷이 전송되는 정확한 시간을 미리 알고 있다고 가정함으로써 [34,35]의 연구를 확장했다. 따라서, 전송할 다음 패킷은 예상되는 지연이 가장 낮은 서브플로우로 할당된다.

DAPS 및 delay aware packet 스케줄러는 수신 측의 버퍼 요구량을 줄일 수 있지만, 단방향 지연의 추정이 잘못되면 성능 저하가 발생할 수 있다. 또한, DAPS의 스케줄링 시간에 대한 오버헤드를 개선해야 하며 송신 측의 cwnd에 미치는 영향을 검토해볼 필요가 있다.

4) BLEST: MPTCP에서 이종 경로는 수신 측에 HoL 블로킹과 out-of-order 문제를 일으켜 링크 용량의 통합을 어렵게 한다. BLEST (BLocking ESTimation-based) 스케줄러[36]는 HoL 블로킹을 최소화하기 위한 proactive 프로토콜이다.

BLEST는 어느 경로가 블로킹을 일으킬지 추정한 다음 HoL 블로킹을 예방하기 위한 스케줄링을 수행한다. MPTCP 송신윈도우 내의 전송할 세그먼트를 비교적 성능이 낮은 서브플로우로 전송할지, 성능이 우수한 서브플로우의 cwnd에 여유가 있을 때까지 기다린 후에 빠른 서브플로우를 사용할지 결정한다. 따라서, minRTT 스케줄러와는 다르게 느린 서브플로우의 cwnd에 여유가 있을지라도 이를 건너뛸 수 있다.

BLEST는 가장 느린 서브플로우의 RTT 동안 이보다 빠른 서브플로우를 통해 전달할 수 있는 데이터의 양 N을 계산한다.

만약 $N \times \delta > packet_{outstanding}$이라면, 느린 서브플로우는 사용되지 않는다. 여기서 δ는 N을 조절하기 위한 correction factor로, MPTCP 연결이 시작될 때 1로 설정된다. BLEST 스케줄러는 동종 네트워크 경로에서 minRTT 스케줄러만큼 잘 동작하며, 이종 네트워크에서는 minRTT 스케줄러보다 더 나은 성능을 보인다.

5) ECF: 이종 네트워크에서 빠른 서브플로우는 느린 서브플로우를 통한 전송이 완료될 때까지 유휴 상태에 머무른다. 이러한 대기시간을 줄이려면 되도록 빠르게 전달을 완료할 수 있는 경로로 패킷을 전송해야 한다. 수신 측에서 요구하는 패킷이 일찍 도착할수록 빠른 서브플로우의 유휴 상태를 빨리 해제할 수 있다. ECF (Earliest Completion First) 스케줄러[37]는 느린 서브플로우의 cwnd는 여유가 있고 빠른 서브플로우의 cwnd는 가득 차서 사용할 수 없을지라도, 느린 서브플로우를 통해 전송이 완료되는 시간과 빠른 서브플로우가 사용 가능해지기를 기다린 후에 이를 통해 전송이 완료되는 시간을 비교하여 스케줄링을 수행한다.

빠른 경로와 느린 경로의 RTT 및 cwnd를 RTT_{fast}, $cwnd_{fast}$, RTT_{slow}로 표현하면 ECF 스케줄러는 식 (4)를 만족할 때만 느린 서브플로우를 통해 다음 N개의 패킷을 전송한다.

$$RTT_{fast} + (\frac{N}{cwnd_{fast}}) \times RTT_{fast} \geq RTT_{slow} \quad (4)$$

ECF는 이종 경로를 통해 대규모 데이터를 전송하는 경우에 BLEST와 DAPS보다 잘 동작하며 더욱 단순하고 효율적인 스케줄러이다. 하지만 ECF 스케줄러의 실험 결과는 여전히 out-of-order 문제가 존재한다는 것을 보여준다.

6) DAPB: MPTCP의 성능이 이종 경로에서 단일 TCP의 성능보다 더 낮게 나타나는 문제점을 해결하기 위해 DAPB (Delay Alerted Path-Blocking) 스케줄러[38]가 제안되었다.

DAPB 스케줄러는 각 서브플로우의 지연 시간을 측정하여 긴 RTT를 갖는 경로는 차단하고, 짧은 RTT를 갖는 서브플로우만을 단일 TCP로 동작시켜 성능 저하 문제를 해결한다. 차단된 느린 경로에는 소량의 중복 패킷을 전송하여 지속적으로 RTT를 측정하고, 해당 경로의 RTT가 낮아진 경우에는 차단을 해제하여 MPTCP로 동작한다.

$$srtt_{threshold} = \max(3 \times srtt_{\min}, srtt_{\text{initial}}) \quad (5)$$

$srtt_i > srtt_{threshold}$일 경우, 스케줄러는 DAPB 알고리즘으로 분기하여 느린 서브플로우를 차단하고 빠른 서브플로우만을 사용한다. 반대의 경우에는 round robin 스케줄러로 동작한다. 이를 통해, DAPB 스케줄러는 경로 간 지연 차이가 큰 경우에도 최소한 단일 TCP만큼의 성능을 보장한다.

7) path capacity based: 우수한 스케줄러는 모든 서브플로우의 cwnd에 여유가 있을 때 혼잡이 덜한 경로로 더 많은 데이터를 전달할 수 있어야 한다. 기존 TCP의 cwnd와 ssthresh 값은 패킷 손실이 잦은 링크에서 경로의 용량을 정확히 반영하지 못하기 때문에, 실제 사용 가능한 용량보다 낮은 경우가 많다. 따라서, [22]에서는 정확한 경로 용량 측정에 초점을 맞춘 스케줄러를 제안하였다. 경로 용량 측정 요소인 $occupied$는 각 서브플로우에 대해서 다음과 같이 계산된다.

$$occupied_i = \frac{packet_{outstanding} + 1}{\text{capacity}} \quad (6)$$

계산된 경로 용량 추정치 $occupied$와 두 개의 혼잡 threshold 값인 γ와 δ를 사용하여 스케줄링을 수행한다.

$occupied \leq \gamma$ 일 때, 추가적인 패킷 전송은 주어진 경로에 대기열을 생성하지 않는다.

$\gamma < occupied \leq \delta$ 일 때, 추가적인 패킷 전송은 주어진 경로에 대기열을 생성한다. $occupied \leq \delta$을 만족하는 경로는 패킷을 보내는데 사용될 수 있으며, 다수의 경로가 이 조건을 만족한다면 가장 낮은 $occupied$ 값을 갖는 서브플로우가 사용된다.

$occupied > \delta$ 일 때, 추가적인 패킷 전송은 경로의 혼잡으로 인해 패킷 손실을 일으킨다. 모든 서브플로우가 이 조건을 만족한다면, 가장 낮은 $occupied$ 값을 갖는 서브플로우가 사용된다.

path capacity based 스케줄러는 RTT 기반의 스케줄러보다 더 나은 성능을 보여주지만, out-of-order 문제에는 초점을 두지 않는다. 이 스케줄러는 최상의 서브플로우에 대한 의존도가 높은 편이며 일부 서브플로우의 이용률이 낮다는 문제점이 존재한다.

8) HSR: Highest Sending Rate 스케줄러[39]는 경로의 goodput을 기반으로 우선권을 부여한다.

$$s_i = MSS_i \times \frac{cwnd_i}{RTT_i} \quad (7)$$

경로 i의 전송률 s_i는 MSS, RTT 및 cwnd로 얻어지고, 가장 높은 s_i 값을 갖는 서브플로우를 통해 데이터를 전달한다. HSR 스케줄러는 경로 간 큰 지연 차이를 갖는 환경에서 더 나은 성능을 보인다.

9) LWS: 경로 i의 여유 cwnd 공간은 식 (8)과 같이 계산되며, $cwnd_i$는 서브플로우 i의 현재 cwnd, f_i는 inflight 데이터를 나타낸다.

$$w_i = cwnd_i - f_i \quad (8)$$

$w_i < f_i$ 일 때는 손실에 의한 복구 상태에 있음을 의미하고, minRTT 스케줄러로 동작한다. 그 외의 경우에 LWS (Largest Window Space) 스케줄러는 더 큰 w_i값을 갖는 서브플로우에 우선권을 부여한다.

LTS (Lowest Time/Space)는 LWS 스케줄러의 변형이며 $RTT_i/cwnd_i$를 기반으로 우선권을 부여한다. 패킷 손실이 있는 환경에서 LTS 스케줄러가 LWS보다 나은 성능을 보인다. 하지만 전송속도의 차이가 있는 경로를 사용할 때는 LWS 스케줄러가 더 잘 동작한다.

10) DEMS: 대부분의 MPTCP 스케줄러는 가장 작은 순서 번호를 가진 패킷을 먼저 스케줄링하며, 이 패킷은 수신 측에 도달한 후 정렬된다. 사전에 경로의

특성을 파악한 후에 패킷을 전송하더라도 언제든 경로의 특성이 변할 수 있으므로 올바른 순서로의 도착을 항상 보장할 수는 없다. 수신 버퍼에서의 패킷 대기를 피하기 위한 가장 좋은 방법은 송신 측에서 패킷을 구별하여 전송하는 것이다.

DEMS (DEcoupled Multipath Scheduler)는 스케줄링에 가장 적합한 chunk 크기를 결정한다[40]. 각 chunk가 수신 측에 동시에 도착할 수 있도록 패킷을 스케줄링하며, 데이터 chunk는 응용 프로그램에서 정의한 바이트 블록으로 구성되는 이미지, Javascript, 오디오 snippet 또는 비디오 chunk와 같은 것이다.

11) adaptive: 특정한 시나리오에만 최적화된 스케줄러는 다른 환경에서는 효율적이지 않을 수 있다[41]. Adaptive 스케줄링은 각 경로의 특성을 기반으로 수행된다.

가. LAMPS: Loss Aware MPTCP Scheduler[41]는 각 서브플로우의 상태를 redundant, normal 두 가지 모드로 분류한다. normal 모드에서는 다른 서브플로우로 전송된 적이 없는 최소의 순서 번호를 가진 패킷을 전송한다. redundant 모드일 때는,

(1) ACK 메시지를 받지 못한 패킷
(2) 해당 서브플로우로 전송된 적이 없는 패킷
(3) (1)과 (2)를 모두 만족하면서 최소의 순서 번호를 갖는 패킷

위의 세 가지 조건에 따라서 패킷을 스케줄링한다. 해당 경로가 높은 패킷 손실률을 보이면 normal 모드에서 redundant 모드로 변경된다. 패킷 손실이 존재하는 환경에서의 state transfer time T는 아래와 같다.

$$T=\frac{\max(n_1,n_2)+1}{2\times\max(n_1,n_2)-1}+RTT \tag{9}$$

n_1은 연속된 두 번의 패킷 손실 이벤트 사이에 전송된 패킷의 수, n_2는 가장 최근에 발생한 패킷 손실 이후에 전송된 패킷의 수를 나타낸다.

나. path aware: 네트워크의 상태가 자주 변화하는 환경이나 전송 시작단계에서는 경로의 특성을 예측하기 어렵다. 경로의 특성 파악이 힘든 경우에는 redundant 모드로 중복 데이터를 전송하는 것이 더 적절할 수도 있다[31]. 따라서, 스케줄러는 경로의 특성을 파악한 후에 응용 프로그램의 요구사항을 기반으로 스케줄링 모드를 선택할 수 있다. redundant 모드는 큰 대역폭보다는 지연에 민감한 응용 프로그램에 더 적합하다.

12) load balance: 서브플로우 간의 부하 불균형은 MPTCP의 성능에 영향을 끼친다[42]. 패킷 x에 대해 각 서브플로우 i에 할당된 가중치 w_i는 $\sum w_i=1$과 같으며, 각 서브플로우에 할당된 가중치에 의해 각 서브플로우의 부하가 결정된다. 즉, $w_i=0$의 값을 갖는 서브플로우에는 패킷이 스케줄 되지 않는다. 스케줄링을 기다리는 데이터는 아래의 식을 만족하는 가중치 요소 Φ에 따라 서브플로우 i에 할당된다.

$$\Phi(x_1,x_2)=\int_{x_1}^{x_2} w_i(x)dx \tag{10}$$

load balance 스케줄러는 w_i에 따라 대기열의 데이터를 서브플로우에 할당하는 WRR (Weighted Round Robin) 스케줄링 알고리즘을 사용한다. WRR 스케줄링은 데이터 레벨의 송신 버퍼에 있는 데이터를 스케줄링 가중치에 따라 서브플로우에 할당한다. 이 스케줄링은 가장 느린 서브플로우에서 가장 빠른 서브플로우로 가중치를 이동시킴으로써 최적의 스케줄링에 도달한다.

13) energy efficient: 현재의 모바일 기기들은 복수의 네트워크 인터페이스를 가지고 있지만, 주로 하나의 인터페이스만을 사용해서 데이터를 수신한다. 모바일 기기에 있어 배터리 수명은 중요하며 불필요한 재전송과 부적절한 다중 경로의 사용은 과도한 배터리 사용을 유발한다. 다수의 경로를 사용한 빠른 전송 완료는 모바일 기기의 디스플레이나 프로세서와 같은 주된 에너지소모 요소의 활동을 줄일 수 있다.

따라서, energy efficient 스케줄러는 프로세싱, 디스플레이, 네트워크 인터페이스 등 모든 부분에서의 에너지소모를 줄이는 것을 목표로 한다[43]. 하지만, 이러한 에너지 효율적인 전송은 오직 에너지소모를 줄이는 것에만 초점을 두기 때문에 데이터 처리 성능이 저하될 수 있다.

Ⅳ. MPTCP 실험환경

본 장에서는 MPTCP 실험을 위한 환경들을 소개하고, 실제 테스트베드 실험을 통한 MPTCP의 총처리량 저하 문제와 대표적인 스케줄러 간 성능 평가 실험 결과를 제시한다.

4.1 MPTCP 실험환경 구성 방법

MPTCP 실험을 수행하기 위한 방법으로는 크게 테스트베드, 에뮬레이션 및 시뮬레이션이 있다. 먼저 테스트베드 실험은 실제 프로토콜을 기반으로 하며, 독립된 네트워크 환경을 구성하거나 실제의 인터넷을 이용하여 실험을 수행할 수 있다. 실제의 인터넷 환경이나 물리적인 장치를 통해 실험을 수행하기 때문에 가장 현실적이고 검증된 방법이지만, 독립된 네트워크를 구성한 경우가 아니면 통제 불가능한 간섭 트래픽과 다양한 네트워크 파라미터들에 의해 실험 결과를 이해하거나 분석하기 어려우며 동일한 시나리오의 실험에 대해 이전과 같은 결과를 얻기 힘들다. 즉, 테스트베드 실험은 세밀한 환경 설정 및 재현이 어렵고 환경 구성 및 실험에 많은 시간이 소모된다. 따라서, 설계 초기에는 변수를 통제할 수 있는 독립된 네트워크를 구성하여 검증하는 것이 적합하며 검증 이후에 더욱 다양한 실험을 위해서 실제 인터넷을 통한 성능 평가를 수행하는 것이 일반적이다.

시뮬레이션 방법은 비교적 간단하게 다양한 네트워크 토폴로지나 시나리오를 구성할 수 있는 장점이 있지만, 각 시뮬레이터의 프레임워크에서 새롭게 구현되어야 한다. 즉, 실제의 프로토콜이 아닌 시뮬레이터의 프레임워크에서 구현된 프로토콜을 기반으로 동작하며 제안 스케줄러를 실제의 테스트베드에서 실험하기 위해서는 실제 프로토콜로 새롭게 구현해야 한다. 현재 NS-3 시뮬레이터에는 MPTCP 모델들이 구현되어 있다[44-46].

실제 프로토콜을 기반으로 재현 가능한 실험을 하기 위한 방법으로는 네트워크 에뮬레이터가 있다. 에뮬레이터를 이용한 시나리오는 시뮬레이션으로 구성할 수 있는 시나리오만큼 방대하진 않지만, 일반적인 토폴로지나 시나리오는 쉽게 구현할 수 있다. 에뮬레이터는 실제 프로토콜을 기반으로 동작하기 때문에 시뮬레이터와는 달리 재구성할 필요가 없으므로 프로토콜의 검증이 간편하다. 대표적인 에뮬레이터인 Mininet[47]과 CORE[48]를 이용해서 MPTCP 실험을 수행할 수 있다.

4.2 MPTCP의 총처리량 저하 문제

MPTCP의 실제 테스트베드 실험으로부터 기대에 미치지 못하는 낮은 처리량이 보고되었고[8], 경로의 특성이 서로 다른 이종 네트워크에서는 MPTCP 연결의 총처리량이 최상의 단일 경로를 사용하는 단일 경로 TCP의 처리량보다 더 낮을 수 있다[38].

본 장에서는, 실제 테스트베드를 구성하여 이종 경로 시나리오에서는 MPTCP의 총처리량이 단일 경로 TCP의 처리량보다 더 낮을 수 있음을 실험을 통해 확인하였다.

이종 경로를 지나는 MPTCP의 성능 비교 실험을 위해 구성한 테스트베드 환경은 <그림 7>과 같다. 하나의 호스트에 두 개의 이더넷을 연결하여 다중경로 TCP 연결을 구성하였다. 각 PC는 Ubuntu 14.04 LTS 운영체제를 사용하여 MPTCP kernel v0.89를 설치하였다. 첫 번째 서브플로우가 지나는 경로는 대역폭과 RTT가 100Mbps, 10ms로 고정되었고 단일 경로 TCP를 위한 최상의 경로로 사용된다. 두 번째 서브플로우가 지나는 경로는 실험에 따라서 RTT가 변화하는 경로로써 이종 네트워크 환경을 구성하는 용도로 활용된다. 실험에서 MPTCP의 혼잡제어 알고리즘으로는 LIA[49], 스케줄러는 MPTCP 커널의 기본 스케줄러인 minRTT를 사용하였고 링크의 버퍼 크기는 1BDP로 설정하였다.

<그림 8>에서 서브플로우 2의 대역폭은 100Mbps로 고정한 상태로 RTT를 10ms부터 100ms까지 증가

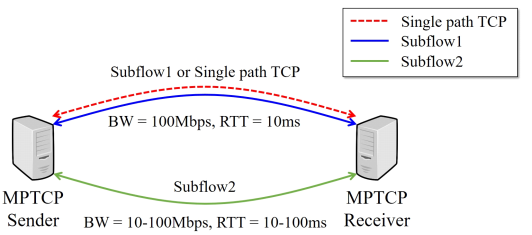


그림 7. 테스트베드 구성도
Fig. 7. Testbed configuration

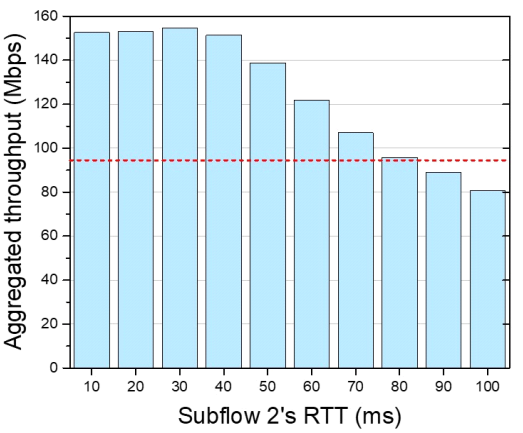


그림 8. 이종 경로에서 MPTCP의 총 처리량은 단일 TCP보다 낮을 수 있다
Fig. 8. The total throughput of MPTCP may be lower than a single TCP in a heterogeneous path

시킬 때의 총처리량을 나타낸다. 서브플로우 2의 RTT가 90ms일 때, 두 서브플로우의 총처리량은 서브플로우 1만을 사용하는 단일 경로 TCP의 처리량인 95Mbps보다 낮게 나타났다. 이 시점부터는 두 경로의 RTT 차이로 인해 과도한 out-of-order 문제와 HoL 블로킹이 발생하여 두 개의 경로를 사용하는 것이 오히려 총처리량의 저하를 이끌었다. 결국, 두 서브플로우 간 RTT 차이가 10배인 상황에서는 단일 TCP의 성능보다 약 10% 이상이나 낮은 총처리량을 보였다.

4.3 MPTCP 스케줄러 성능 비교

MPTCP 스케줄러의 성능을 평가하기 위해, Mininet 에뮬레이터를 이용하여 <그림 9(a), (b)>와 같은 실험환경을 구성하였다. 실험에서 평가한 스케줄러는 default, round-robin, ECF, BLEST, DAPS, OTIAS 스케줄러이다.

<그림 9(c)>로부터 동종 병목링크 환경에서는 ECF와 BLEST 스케줄러가 약간의 성능 향상을 보였지만, OTIAS와 DAPS 스케줄러는 잘못된 단방향 지연의 측정 등으로 인해 두 개의 경로를 충분히 활용하지 못하였기 때문에 default 스케줄러보다 낮은 성능을 보였다.

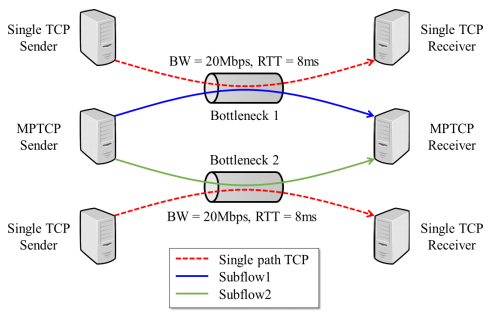
<그림 9(d)>의 이종 병목링크 환경에서 round-robin 스케줄러는 성능이 크게 감소했고 ECF와 BLEST는 default 스케줄러에 비해 더욱 개선된 처리량을 보였다.

현재까지 제안된 MPTCP 스케줄러들은 서로 다른 목적을 가지고 동작하기 때문에, 아래의 실험과 같은 비교적 단순한 예제에서나 특정한 환경에서는 default 스케줄러보다 더 낮은 성능을 보일 수도 있다. 그러므로, MPTCP 배치를 통해 최대의 성능을 얻기 위해서는 최적의 MPTCP 스케줄러 설계를 위한 연구가 필요하다.

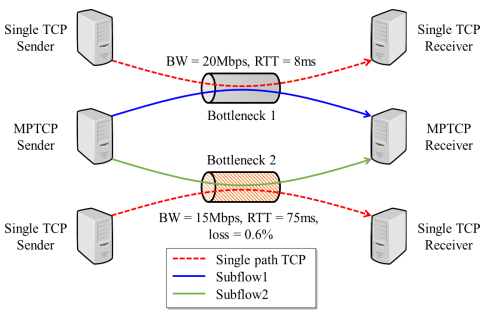
Ⅴ. MPTCP 스케줄러 설계를 위한 권장 사항

Ⅲ 장에서는 다양한 MPTCP 스케줄러에 대해서 살펴보았으며, 이들의 주요 목표는 out-of-order 도착을 피하고 HoL 블로킹을 방지하여 최대의 처리량을 달성하는 것이다. <표 2>에서 대표적인 MPTCP 스케줄러의 주요 장점과 한계를 요약한다.

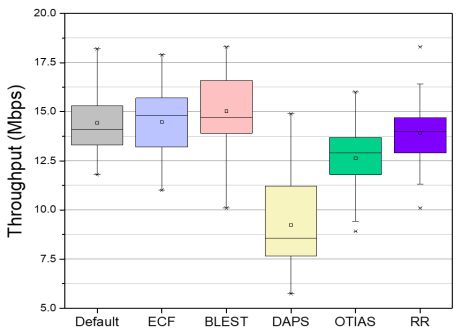
앞선 장에서 논의된 MPTCP 스케줄러는 다음과 같



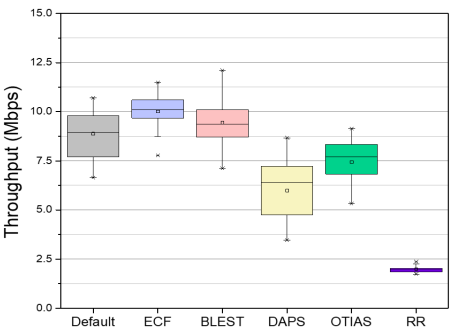
(a) 동종 병목링크 구성도



(b) 이종 병목링크 구성도



(c) 동종 병목링크에서 MPTCP 스케줄러별 성능



(d) 이종 병목링크에서 MPTCP 스케줄러별 성능

그림 9. MPTCP 스케줄러의 성능 평가를 위한 실험환경 및 결과
Fig. 9. Experimental environment and results for MPTCP scheduler performance evaluation

표 2. 주요 스케줄러의 장점과 한계
Table 2. Advantages and limitations of major scheduler

스케줄러	장점	한계
minRTT	간단한 디자인	지연 차이가 큰 환경에서 낮은 성능
round robin	간단한 디자인	이종 경로에서 낮은 성능
DAPS	수신 측 버퍼 요구량 감소	복잡한 디자인에 비해 낮은 성능 개선
BLEST	HOL 블로킹 문제 개선	느린 서브플로우의 낮은 활용률
ECF	이종 경로에서 우수한 성능	복잡한 디자인
DAPB	최악의 상황에도 단일 TCP만큼의 성능 보장	경로 간 지연 차이만을 고려함
Path capacity based	우수한 혼잡상황 회피	decision threshold 값을 측정하기 어려움
HSR	지연 차이가 큰 환경에서 우수한 성능	패킷 손실 환경에서 오작동의 가능성 존재
LWS	전송률 차이가 큰 환경에서 우수한 성능	빈번한 out-of-order 문제 발생
DEMS	out-of-order 문제 완화	소규모 데이터 전송에 적절하지 않음
Adaptive	패킷 손실이 심한 환경에서의 우수한 성능	낮은 대역폭 사용률
Load balance	경로 활용률 향상	out-of-order 발생 가능성을 무시
Energy efficient	에너지 효율 개선	오직 에너지 효율만을 고려함

이 분류할 수 있다[50]:

- best value 우선: 이러한 스케줄러는 일부 매개변수(RTT, cwnd, MSS) 또는 매개변수의 함수에 기초하여 서브플로우의 우선순위를 결정한다. 최상의 서브플로우의 cwnd를 가장 먼저 채우고 순차적으로 다른 서브플로우를 사용한다. 이들의 목적은 특정한 성능 목표치를 달성하는 것이다.
  - 예: minRTT, HSR, LWS.
- in-order 도착 우선: 이 범주의 스케줄러는 전파 지연 또는 추정된 cwnd와 같은 스케줄링 매개변수를 기반으로 송신 버퍼에서 패킷을 미리 스케줄링하여 송신 측에서 전송된 순서와 같은 순서로 수신될 수 있도록 한다. 이러한 스케줄러의 목표는 달성 가능한 최대 처리량에 도달하는 것이다.
  - 예: BLEST, ECF, DAPS.
- adaptive: 패킷 손실이나 지연 차이와 같은 경로 특성에 따라서 스케줄링 방식을 전환하는 종류의 스케줄러이다. 이들은 현 네트워크의 상태가 MPTCP를 사용하는 것이 유리한 경우에 한정하여 다중 경로를 사용하는 특징을 갖는다.
  - 예: loss-aware, path-aware 스케줄러, DAPB

위의 세 가지 범주의 스케줄러는 모든 유형의 통신에 적용할 수 있지만, adaptive 스케줄러는 종종 redundant나 백업 또는 단일 TCP 모드로 전환되기 때문에 MPTCP의 기능을 제한하기도 한다. 따라서, 이들은 수신 측에서의 in-order 도착률을 높이기 위해서 더 많은 개선이 필요하다.

잘못된 패킷 스케줄링 정책은 이상적인 총처리량을 달성하지 못할 뿐만 아니라, 최악의 경우에는 단일 TCP보다 낮은 처리량을 가질 수 있다. 높은 처리량을 얻기 위해서는 동적인 수신 버퍼 크기 상한선, 다중 경로 간 최적의 지연 차이, 응용 프로그램의 요구사항, 스케줄링 모드 간 전환을 위한 다양한 매개변수 등이 적절하게 고려되어야 한다. 지금까지 살펴본 MPTCP 스케줄러들의 특징을 토대로 우수한 스케줄러 설계를 위한 권장 사항을 다음과 같이 요약할 수 있다.

1) 응용 프로그램마다 대역폭, 지연 시간, 최소 비트 전송률 등 서로 다른 우선순위 및 요구사항이 존재한다. 단순히 일정한 전송률을 요구하는 응용 프로그램에 맞춰 스케줄링을 수행한다면, 순간적으로 많은 데이터 전송이 필요한 응용 프로그램에서는 최적의 효율을 내지 못할 수 있다. 개별 스케줄러들은 각기 다른 요구사항을 갖는 프로그램들에 완벽하게 동조할 순 없지만, 응용 프로그램의 요구사항을 충분히 인지하여 스케줄링을 수행해야 한다.

2) 적절한 양의 트래픽 분배와 더불어, 여러 개의 서브플로우를 사용할지 오직 하나의 플로우를 사용할지를 결정해야 한다. 이 결정은 짧은 데이터 전송

또는 손실이 심한 네트워크 시나리오에서 특히 중요하다. 이와 같은 결정을 확장하면, 대역폭보다 신뢰성과 지연 시간이 더 중요한 시나리오의 경우에는 redundant 스케줄링을 사용하여 요구사항을 충족할 수 있다. 또한, 최상의 경로만을 사용하고 나머지 서브플로우는 백업 모드로 전환하여 견고성, 끊임없는 이동성 및 링크 전환을 제공하여 QoS를 개선할 수 있다. 응용 프로그램이 단순히 넓은 대역폭을 요구할 때는 사용 가능한 모든 서브플로우를 활용할 수 있는 정책이 필요하다.

3) 리눅스 MPTCP의 기본 스케줄러는 이종 경로를

통한 스케줄링에서 성능 저하를 유발할 수 있다. 느린 서브플로우는 빠른 서브플로우에 대한 서비스 속도를 제한하기 때문에 스케줄러는 이들 간의 지연 차이를 계속해서 추적해야 한다. 과도한 지연 차이로 인해 성능 저하가 발생하는 경우, 스케줄러는 느린 서브플로우를 차단하거나 페널티를 가할 수 있다. 하지만 이 대안은 대역폭 사용을 제한하기 때문에 광범위한 대역폭이 필요한 응용의 경우에는 제외되어야 한다.

4) MPTCP에서 수신 측 버퍼 요구량은 Ⅱ장의 수식 (1)과 같이 RTT에 따라 달라진다. 무선 네트워크와 같은 손실이 심한 환경에서는 RTT의 변화가 매우 동적이기 때문에 고정된 버퍼 크기의 사용이 단점이 될 수 있다. 작은 크기의 버퍼는 송신 측의 성능을 제한할 수 있으며, 큰 버퍼는 긴 대기 지연을 초래할 수 있다. 따라서, RTT의 변화에 따라 동적으로 버퍼 상한선을 유지하는 것을 권장한다.

5) 다수의 매개변수를 기초로 한 스케줄링이 항상 최적인 것은 아니다. 하나의 매개변수로 인한 영향과 다른 매개변수로 인한 영향을 모델화하는 것이 중요하다.

6) 데이터 레벨 재전송과 서브플로우 레벨 재전송 간의 동기화가 잘 이루어져야 한다. 서브플로우 레벨에서 손실된 패킷이 다른 서브플로우를 통해 재전송되는 적극적인 데이터 레벨의 재전송 정책은 대역폭을 낭비할 순 있지만 응용 프로그램 간의 지연을 줄일 수 있다. 반면, 데이터 레벨 재전송이 일부 서브플로우 레벨 재전송의 타임아웃 이후에만 사용되는 보수적인 재전송 정책은 다른 서브플로우의 성능을 제한하는 결과를 초래할 수 있으므로 주의해야 한다[15].

Ⅵ. 결 론

MPTCP의 성능은 스케줄링 정책에 크게 의존한다. 형편없는 스케줄링 정책에 의한 무작위의 패킷 할당은 MPTCP의 성능을 악화시켜서 단일 TCP보다도 낮은 성능을 초래할 수 있다. 지금까지 많은 MPTCP 스케줄러가 문헌을 통해 제안되었지만, 서브플로우의 전체 용량을 완전히 활용하지 못하는 경우가 많으며 실제 인터넷에 적용 가능한 최적의 스케줄러는 아직 개발되지 않았다.

본 논문에서는 MPTCP의 성능을 악화시키는 문제들을 살펴보았고 이를 해결하기 위해 제안된 다양한 MPTCP 스케줄러의 동작 절차와 이들의 장점 및 한계에 대해 논의하였다. 또한, MPTCP 스케줄러의 개발과 성능 평가를 수행하기 위한 실험환경의 구현 방법을 소개하였고, 실제 실험을 통해 MPTCP의 기본 스케줄러의 문제점을 확인하였다. 현재까지 제안된 다양한 MPTCP 스케줄러가 특정한 환경이나 응용 프로그램에서는 좋은 성능을 보이지만, 항상 최상의 성능을 보이는 것은 아니었다.

이러한 검토를 토대로 우수한 MPTCP 스케줄러를 설계하기 위한 몇 가지 권장 사항을 제시하였다. 우수한 스케줄러는 응용 프로그램의 요구사항을 검토하고 그에 따라 트래픽을 정렬해야 하며, 네트워크 특성을 파악하여 경로를 선택하고 out-of-order 도착 및 HoL 블로킹을 피해야 한다.

References

[1] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, and S. Barre, "*TCP extensions for multipath operation with multiple addresses*," RFC 6824, IETF, version 18, 2019.

[2] *Samsung Download Booster*, 2016, [Online]. Available:http://www.samsung.com/uk/support/skp/faq/1061358.

[3] *Multipath TCP is pronounced GIGA Path*, 2015, [Online]. Available:http://blogmultipath-tcp.org/blog/html/2015/07/24/korea.html.

[4] *iOS: Multipath TCP Support in iOS 7*, 2017, [Online]. Available: https://support.apple.com/en-us/ HT201373.

[5] O. Bonaventure, M. Handley, and C. Raiciu, "An Overview of Multipath TCP," *USENIX Netw. Syst.*, vol. 37, no. 5, pp. 17-23, Oct. 2012.

[6] *MPTCP Linux Implementation*, [Online]. Available: http://www.multipath-tcp.org/.

[7] K. J. Grinnemo and A. Brunstrom, "A first study on using MPTCP to reduce latency for cloud based mobile applications," *IEEE Symp. Comput. and Commun.*, pp. 64-69, 2015.

[8] A. Singh, et al., "Performance evaluation of multipath TCP linux implementations," *Wrzburg Workshop on IP: Joint ITG and Euro-NF Workshop Visions of Future Generation Networks*, Aug. 2011.

[9] S. C. Nguyen, X. Zhang, T. M. T. Nguyen, and G. Pujolle, "Evaluation of throughput

optimization and load sharing of multipath TCP in heterogeneous networks," *Int. Conf. Wireless and Optical Commun. Netw.*, pp. 1-5, 2011.

[10] A. Alheid, D. Kaleshi, and A. Doufexi, "An analysis of the impact of Out-of-Order recovery algorithms on MPTCP throughput," *IEEE Int. Conf. Advanced Info. Netw. and Appl.*, pp. 156-163, 2014.

[11] S. Ferlin, T. Dreibholz, and O. Alay, "Multi-path transport over heterogeneous wireless networks: Does it really pay off?," *IEEE Global Commun. Conf.*, pp. 4807-4813, 2014.

[12] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *ACM SIGCOMM Comput. Commun. Rev. (CCR)*, vol. 32, no. 4, Aug. 2002.

[13] C. Paasch, R. Khalili, and O. Bonaventure, "On the benefits of applying experimental design to improve multipath TCP," *ACM CoNEXT*, Dec. 2013.

[14] W. Lei, "Congestion control and scheduling in multipath TCP," M.S. Thesis, University of Helsinki, Faculty of Sci., Dept. Computer Sci., 2017.

[15] M. Scharf and A. Ford, "*Multipath TCP (MPTCP) application interface considerations*," RFC 6897, IETF, Mar. 2013.

[16] R. Khalili, N. Gast, M. Popovic, and J.-Y. L. Boudec, "MPTCP is not Pareto-Optimal: Performance issues and a possible solution," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, Oct. 2013.

[17] F. Yang, P. Amer, and N. Ekiz, "A scheduler for multipath TCP," *ICCCN*, pp. 1-7, Nassau, Bahamas, Aug. 2013.

[18] F. Yang, Q. Wang, and P. Amer, "Out-of-order transmission for in-order arrival scheduling policy for multipath TCP," *WAINA*, pp. 749-752, 2014.

[19] J. Liu, H. Zou, J. Dou, and Y. Gao, "Rethinking retransmission policy in concurrent multipath transfer," *IIHMSP*, pp. 1005-1008, 2008.

[20] M. Li, et al., "Multipath transmission for the internet: A survey," *IEEE Commun. Surveys & Tuts.*, vol. 18, no. 4, pp. 2887-2925, 2016.

[21] J. Martin, A. Nilsson, and I. Rhee, "The incremental deploy-ability of RTT-based congestion avoidance for high speed TCP Internet connections," *ACM SIGMETRICS Int. Conf. Measurement and Modeling of Comput. Syst.*, pp. 134-144, 2000.

[22] F. Yang, P. Amer, and N. Ekiz, "A scheduler for multipath TCP," *ICCCN*, pp. 1-7, 2013.

[23] A. Froemmgen, J. Heuschkel, and B. Koldehofe, "Multipath TCP scheduling for thin streams: Active probing and one-way delay-awareness," *IEEE ICC*, pp. 1-7, 2018.

[24] M. Laner, et al., "A comparison between one-way delays in operating HSPA and LTE networks," *Int. Symp. Modeling and Optimization in Mob., Ad Hoc and Wireless Netw. (WiOpt)*, pp. 286-292, May 2012.

[25] S. H. Baidya and R. Prakash, "Improving the performance of multipath TCP over heterogeneous paths using slow path adaptation," *IEEE ICC*, pp. 3222-3227, 2014.

[26] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath TCP schedulers," in *Proc. ACM CSWS*, pp. 27-32, 2014.

[27] F. Zhou, et al., "The performance impact of buffer sizes for multi-path TCP in internet setups," *Int. Conf. Advanced Info. Netw. and Appl.*, pp. 9-16, 2017.

[28] M. Li, et al., "Tolerating path heterogeneity in multipath TCP with bounded receive buffers," *Comput. Netw.*, vol. 64, no. 4, pp. 1-14, 2014.

[29] R. Khalili, N. Gast, M. Popovic, and J. Y. L. Boudec, "*Opportunistic linked-increases congestion control algorithm for MPTCP*," IETF draft-khalilimptcp-congestion-control-05, Jul. 2014.

[30] A. Walid, et al., "*Balanced linked adaptation congestion control algorithm for MPTCP*," IETF draft-walid-mptcp-congestion-control-04, Jul. 2014.

[31] J. Zhu, et al., "*A path-aware scheduling scheme for multipath transport protocol*," IETF draft, Mar. 2018.

[32] H. Shi, et al., "Improving MPTCP throughput under heterogeneous networks," *Annu. Technical Conf. (USENIXATC 2018)*, pp. 719-730, 2018.

[33] J. Hwang and J. Yoo, "Packet scheduling for multipath TCP," *Int. Conf. Ubiquitous and Future Netw.*, pp. 177-179, 2015.

[34] G. Sarwar, R. Boreli, E. Lochin, A. Mifdaoui, and G. Smith, "Mitigating receivers buffer blocking by delay aware packet scheduling in multi-path data transfer," *Int. Conf. Advanced Info. Netw. and Appl. Wkshps.*, pp. 1119-1124, 2013.

[35] N. Kuhn, et al., "DAPS: Intelligent delay-aware scheduling for multipath transport," *IEEE Int. Conf. Commun.*, pp. 1222-1227, Jun. 2014.

[36] S. Ferlin, O. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," *IFIP Networking and Wkshps.*, pp. 431-439, 2016.

[37] Y. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," in *Proc. CoNEXT ACM*, pp. 147-159, 2017.

[38] M. S. Kim, J. Y. Lee, and B. Kim, "Improving the performance of multipath tcp using delay alerted path-blocking scheduler in heterogeneous networks," *J. IEIE*, vol. 54, no. 2, pp. 28-37, 2017.

[39] B. Y. L. Kimura, D. C. S. F. Lima, and A. A. F. Loureiro, "Alternative scheduling decisions for multipath TCP," *IEEE Commun. Lett.*, vol. 21, no. 11, pp. 2412-2415, 2017.

[40] Y. E. Guo, A. Nikravesh, Z. M. Mao, F. Qian, and S. Sen, "Accelerating multipath transport through balanced subflow completion," in *Proc. 23rd Annu. Int. Conf. Mob. Comput. and Netw.*, pp. 141-153, 2017.

[41] E. Dong, M. Xu, X. Fu, and Y. Cao, "LAMPS: A loss aware scheduler for multipath TCP over highly lossy networks," *IEEE Conf. Local Comput. Netw.*, pp. 1-9, 2017.

[42] K. Choi, Y. Cho, Aneta, J. Lee, S. Cho, and J. Choi, "Optimal load balancing scheduler for MPTCP-based bandwidth aggregation in heterogeneous wireless environments," *Comput. Commun.*, vol. 112, pp. 116-130, 2017.

[43] M. Morawski and P. Ignaciuk, "Energy Efficient MPTCP transmission scheduler implementation and evaluation," *Int. Conf. Syst. Theory, Control and Comput.*, pp. 654-659, 2017.

[44] B. Chihani and C. Denis, "A multipath TCP model for NS-3 simulator," arXiv:1112.1932, 2011.

[45] M. Kheirkhah, I. Wakeman, and G. Parisis, "Multipath-TCP in NS-3," arXiv:1510.07721, 2015.

[46] M. Coudron and S. Secci, "An implementation of multipath TCP in NS-3," *Comput. Netw.*, vol. 116, pp. 1-11, 2017.

[47] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for sofware-defined networks," in *Proc. 9th ACM SIGCOMM Wkshp. Hot Topics Netw.*, pp. 19:1-19:6, 2010.

[48] J. Ahrenholz, "Comparison of core network emulation platforms," in *Proc. IEEE Military Commun. Conf.*, pp. 166-171, Oct. 2010.

[49] C. Raiciu, D. Wischik, and M. Handley, "*Practical congestion control for multipath transport protocols*," Univ. College London, London, U.K., Tech. Rep., 2009.

[50] X. Corbillon, et al., "Cross-layer scheduler for video streaming over MPTCP," *Int. Conf. Multimedia Syst., ACM*, no. 7, pp. 1-12, 2016.

김 건 환 (Geon-Hwan Kim)

2015년 2월 : 경북대학교 전자공학부 졸업
2017년 2월 : 경북대학교 전자공학부 석사
2017년 3월~현재 : 경북대학교 전자공학부 박사과정
<관심분야> 차세대 이동 네트워크, 드론 ICT 융합 기술, TCP 혼잡제어, MPTCP 혼잡제어
[ORCID:0000-0003-2739-8939]

박 창 훈 (Chang-Hoon Park)

2019년 2월 : 경북대학교 전자공학부 졸업
2019년 3월~현재 : 경북대학교 전자공학부 석사과정
<관심분야> MPTCP 스케줄러
[ORCID:0000-0002-5066-2980]

김 정 근 (Jeong-Keun Kim)

2020년 2월 : 대구대학교 전자제어공학전공 졸업
2020년 3월~현재 : 경북대학교 전자공학부 석사과정
<관심분야> QUIC, MPTCP, MPQUIC
[ORCID:0000-0002-5577-6004]

엄 원 주 (Won-Ju Eom)

2020년 2월 : 경북대학교 전자공학부 졸업
2020년 3월~현재 : 경북대학교 전자공학부 석사과정
<관심분야> SDN, TCP 혼잡제어
[ORCID:0000-0003-4322-0882]

송 영 준 (Yeong-Jun Song)

2019년 2월 : 경북대학교 전자공학부 졸업
2019년 3월~현재 : 경북대학교 전자공학부 석사과정
<관심분야> TCP 혼잡제어, 미래 네트워크, 전송 계층 프로토콜
[ORCID:0000-0001-7586-5795]

서 원 경 (Won-Kyeong Seo)

2005년 : 경북대학교 전자전기컴퓨터학부 졸업
2007년 : 경북대학교 전자공학 석사
2012년 : 경북대학교 전자공학 박사
2012년~2014년 : 지웰주식회사 대표이사
2015년~현재 : 영진전문대학교 국방전자통신전공 교수
<관심분야> 차세대 이동 네트워크, 무선 애드혹 네트워크, 이동성 관리 기술, 차세대 전송 계층 프로토콜
[ORCID:0000-0001-5920-4858]

조 유 제 (You-Ze Cho)

1982년 : 서울대학교 전자공학과 졸업
1983년 : 한국과학기술원 전기전자공학 석사
1988년 : 한국과학기술원 전기전자공학 박사
1989년~현재 : 경북대학교 전자공학부 교수
1992년~1994년 : Univ. of Toronto in Canada, 객원교수
2002년~2003년 : NIST(미국국립표준연구소) 객원연구원
<관심분야> 차세대 이동 네트워크, 무선 애드혹 네트워크, 이동성 관리 기술, 차세대 전송 계층 프로토콜
[ORCID:0000-0001-9427-4229]