https://doi.org/10.7840/kics.2023.48.1.55

네트워크 지연 시간 보장을 위한 전역 완료 시간 기반 작업보존형 스케줄러

정 진 우◆

Work Conserving Scheduler with Global Finish Time for Network Latency Guarantee

Jinoo Joung◆

요 약

네트워크의 단대단 지연 시간 상한을 보장하는 방안에 관해서 많은 연구가 진행되고 있다. 스케줄러가 플로우들을 보호해서 다양한 주변 환경에서도 플로우에 대한 서비스를 일정 수준 이상으로 보장해 주는 것이 필요하다. 이를 위해 패킷 별 상태 정보를 네트워크의 최초 노드에서 생성하여 이를 패킷에 기록하여, 이후 노드들에서는 이들 기록만을 바탕으로 플로우 상태를 추론해서 서비스하는 방안이 제시되었다. 본 논문에서는 이러한 플로우 상태 추론을 위한 보다 간단한 방법을 제시한다. 제시한 스케줄러는 작업보존방식으로 동작하며, 같은 경로를 지나가는 모든 플로우의 패킷 간 서비스 순서가 경로 중간에 바뀌지 않는다는 특성을 가진다. 이에 따라 간단한 input port 별 FIFO 큐를 이용해서 해당 큐의 Head of Queue(HoQ)만을 비교함으로써 최소 완료 시간을 가진 패킷을 손쉽게 찾아낼 수 있다는 장점이 있다. 더 나아가 제시한 스케줄러의 일례가 packetized rate proportional server(PRPS)임을 증명하였다. 따라서 간단한 구현으로 확장성과 플로우 별 보호를 가능케 하여 기존 표준들에서 제시한 지연 시간 상한 수준을 뛰어넘는 성능을 동시에 확보하였다.

Key Words : end-to-end latency, deterministic networking, flow states, TSN, finish time

ABSTRACT

Guaranteeing end-to-end network latency bounds is required in many applications. Ideally, the flows should be protected in separate queues against the burst accumulation. Means to provide per-flow performance without maintaining flow states have been suggested in the literature. The global finish time, which is the essential of the packet state, is derived at the entering node of a flow, and then recorded as a meta-data to a packet, retrieved at the downstream nodes. The global finish time is usually modified according to the current state of the downstream nodes. In this work, we suggest a simple method that approximates the current flow state with the global finish time. The suggested method has the property that the service order of all the packets having the same path does not change during the path. By this property the problem of finding the minimum finish order has become much simpler, with a FIFO queue per input port and a HoQ examiner. It is proved that the proposed scheduler is a packetized rate-proportional server, therefore enjoys much better latency upper bounds and the improved scalability.

※ 본 연구는 상명대학교 교내 연구과제의 지원을 받아 수행된 연구임.
◆ First Author : 상명대학교 휴먼지능정보공학과, Sangmyung university, Department of Human-centered AI, jjoung@smu.ac.kr, 정회원
논문번호 : 202205-107-B-RN, Received September 18, 2022; Revised October 22, 2022; Accepted October 25, 2022

Ⅰ. 서 론

지연 시간 보장과 플로우 보호(protection)는 밀접한 관계가 있다. 플로우 보호의 정도는 지연 시간 상한값과 반비례한다. 크게 세 가지 수준의 플로우 보호 기술이 존재한다.

첫 번째 수준의 기술은 패킷 크기의 최대치 수준으로 플로우 간의 간섭을 제한하는 기술이다. Packetized generalized processor sharing (PGPS)과 weighted fair queuing (WFQ)이 이러한 기술의 대표적인 에이다[1]. 이러한 기술에서는 플로우 별 큐가 필요하며 플로우의 서비스 상태를 실시간으로 기록하여 정확하게 할당된 만큼의 서비스를 제공하는 것이 핵심이다. 이러한 기술을 적용하면 각 노드에서의 지연 시간 상한(upper bound)이 최대 패킷 길이와 비례한다. 하지만 이를 위해 많은 수의 플로우 별 상태 정보를 관리하고 기록하는 복잡성이 문제가 되어 실제로는 적용되지 않고 있다.

두 번째 수준의 기술은 플로우의 특성을 반영한 자세한 개별적 상태는 유지하지 않고, 대신 플로우의 고정적인 요구사항과 서비스 이력 정도만 기록하여 이를 바탕으로 앞으로의 서비스를 규정한다. Weighted round robin(WRR)과 deficit round robin (DRR)[2] 등이 이들 기술에 해당한다. 이러한 기술을 적용한 경우 각 노드에서의 지연 시간 상한이 플로우 별 최대 패킷 크기의 합에 비례한다.

세 번째 수준의 기술은 플로우들을 우선순위 별로 묶은 큐에 저장하고 단순하게 순서대로 큐를 서비스하는 것이다. Differentiated Services(DiffServ)에서 제시하는 방안이 이러한 기술의 대표적인 에이다[3]. 이러한 기술을 적용한 경우 지연 시간 상한은 플로우 별 최대 버스트 크기의 총합으로 결정된다. 최대 버스트 크기는 플로우 별로 허용된, 최초 발신지에서 한 번에 보낼 수 있는 데이터의 총량을 의미한다. 일반적으로 많은 수의 패킷을 묶은 개념을 나타내며, 따라서 패킷 길이보다는 상당히 큰 수준이다. 따라서 이들의 총합에 비례하는 지연 시간 상한은 경우에 따라 수용하기 어려운 수준일 수 있다. 더구나 이러한 기술을 적용한 네트워크에서는 네트워크 토폴로지 상에 순환이 존재하는 경우, 최대지연 시간이 노드를 거침에 따라 증폭되어, 네트워크 사용률 수준에 따라 지연 시간 상한을 보장할 수 없는 상황이 발생할 수 있다[4].

따라서 첫 번째 수준의 기술을 인터넷에 적용하는 방안이 연구되어 왔으나, 수백만 개에 이르는 플로우 상태를 실시간으로 코어 노드에서 관리하는 것은 불가능하다. 이의 해결을 위해 플로우의 상태 정보 관리 없이 패킷에 필요한 정보를 기재하고 이를 중간에 수정할 수 있게 하여 플로우 상태를 패킷 상태만으로 도출하도록 하는 방안이 제시되었다[16]. 하지만 기존 연구는 매 노드에서 연속된 패킷 간 서비스 정보를 계속 업데이트하려고 하여, 같은 플로우 내 패킷들의 서비스 순서가 뒤바뀔 가능성을 내포한다. 이를 방지하기 위해 패킷의 서비스 시작 가능 시간(Eligible time)이라는 개념을 도입하여 패킷 간 서비스 순서를 보장한다. 이에 따라 비 작업보존방식(non work conserving)으로 동작한다. 따라서 지연 시간 상한은 보장하지만, 평균 지연 시간이 많이 늘어난다는 단점이 있어서 이에 대한 해결책이 필요하다.

한편, 이러한 패킷 관련 정보를 패킷 헤더에 기재하고 이를 매 노드에서 수정하는 방안은 복잡성으로 인해 그동안 인터넷 표준으로 받아들여지지 않았다. 하지만 최근 Segment routing[19]과 DetNet[20] 등의 IETF WG에서 그 필요성이 강조되고 있어서 그 가능성과 효과에 관한 연구가 필요한 상황이다.

본 논문에서는 이러한 패킷 상태를 도출하는 간단하고 효율적인 방법을 제시한다. 완료 시간(Finish time)으로 대표되는 패킷 상태 정보를 네트워크의 최초 노드에서 생성하여 이를 패킷에 기록하며, 이후 노드들에서는 해당 노드의 상태를 바탕으로 패킷 상태를 업데이트하여 작업보존방식으로 서비스하는 방안을 제시한다. 이렇게 최초 노드에서 생성하여 네트워크 전반에 걸쳐서 사용되는 완료 시간을 전역 완료 시간이라고 부른다. 제시한 방안에서는 같은 경로를 지나가는 모든 플로우의 패킷 간 서비스 순서가 경로 중간에 바뀌지 않는다는 특성을 가진다. 이에 따라 간단한 input port 별 FIFO 큐를 이용해서 해당 큐의 헤드(Head of Queue, HoQ)만을 비교함으로써 최소 완료 시간을 가진 패킷을 손쉽게 찾아낼 수 있다는 장점이 있다. 더 나아가 제시한 스케줄러의 일례가 packetized rate proportional server(PRPS)임을 증명하였다.

본 논문의 구성은 다음과 같다. 2장에서는 확정적 네트워크 기술 관련 기존 연구들과 표준화 추세에 대해 알아본다. 3장에서는 전역 완료 시간에 기반하여 플로우 별 상태 관리 없이 간단하게 각 노드에서 완료 시간을 도출하는 알고리즘을 제시한다. 이를 바탕으로 구체적인 패킷 수준의 스케줄링 알고리즘을 제시하고 이 스케줄러가 Packetized rate proportional server (PRPS) 임을 증명한다. 간단한 토폴로지의 네트워크에서 기존 표준 기술과의 성능을 비교한다. 제시한 스케줄러는 기존 우선순위 기반의 기술들 대비하여 동

등한 수준의 확장성을 확보하면서도 월등한 지연 시간 성능을 보인다. 4장에서는 결론과 향후 연구 방향을 서술한다.

Ⅱ. 관련 연구

2.1 확정적 네트워크 기술

스마트 팩토리, 차량 내 네트워크 등의 환경에서 지연 시간과 지연 시간 변동폭 (지터, jitter)의 상한을 보장해 주는 네트워크 서비스에 대한 요구가 생겨났다. 이에 따라 IEEE time sensitive networking (TSN) task group[5]에서 관련 기술에 대한 표준화가 진행됐으며, 메타버스 등 대규모 네트워크에서도 이러한 요구가 발생하여 IETF에서도 deterministic networking (DetNet) working group[20]을 중심으로 IP/MPLS 등 상위 layer에서의 이슈, 대규모 네트워크에서의 이슈 등이 다루어지고 있다.

최근 IETF DetNet WG에서는 세 가지 방향으로 대규모 네트워크에서의 지연 시간 보장 문제를 해결하고자 하고 있다. 첫 번째는 TSN[5] 동기식 기술[9]의 확장이다. 노드들의 슬롯 길이는 동일하게 유지하지만, 시작과 끝 시점은 동기시키지 않는다. 이렇게 함으로써 전파지연(propagation delay)과 dead time 문제를 해결할 수 있다고 한다. 하지만 인접한 노드 간 tag 정보로 패킷이 서비스되어야 하는 슬롯을 명시해 줄 필요가 있으며, 따라서 이를 Tagged CQF라고 부른다. 이러한 방식의 가장 큰 단점은 슬롯 크기 결정, 슬롯에 플로우들을 배정하는 슬롯 스케줄링 등에 상당한 복잡도가 내포되어 있어 실시간으로 동적으로 변하는 네트워크에서 사용하기 힘들다는 점이다. 슬롯 크기가 커지면 자원의 낭비도 심하며 지연 시간 상한도 비례해서 커진다.

두 번째 방식은 TSN 비동기식 기술인 ATS[6]의 확장이다. 네트워크를 적당한 크기의 도메인으로 나누고 도메인 간 interleaved regulator(IR)를 배치하여 버스트 축적을 방지한다. 도메인 내에서는 같은 경로를 가지는 플로우들을 묶어서 처리함으로써 플로우 기반 처리기술의 복잡성을 상당 부분 상쇄하였다. 현재로는 유일하게 지연 시간 상한을 보장하며, 복잡성도 일정 수준 이하로 유지하여 가장 합리적인 방안으로 여겨진다[8,10]. 이러한 기술은 ITU-T에서도 표준으로 승인된 바 있다[7]. 이의 변형된 형태로, 노드 내 경로를 공유하는 플로우들의 묶음을 하나의 플로우로 보고 버스트 크기와 평균 인입 속도의 합을 바탕으로 묶음을 레귤레이트하는 방안도 제시되어 가장 높은 수준의 확장성을 제공한다 [11].

세 번째 방식은 패킷에 다양한 종류의 메타 데이터를 기록하여 이들을 바탕으로 포워딩 정책을 결정하는 기술들을 총칭한다. 메타 데이터의 종류로는 현재까지의 누적 지연 시간, 지연 시간 상한, 혹은 이들을 뺀 값인 지연 시간 budget 등이 언급되고 있다. 하지만 이러한 메타 데이터 기반 기술들은 아직 이론적인 지연 시간 상한을 제시하지 못하고 있으며, proof-of-concept 네트워크의 간단한 구현을 통해 해당 구현물에서 지연 시간 보장이 가능하다고 보고하고 있을 뿐이다. 하지만 이러한 메타 데이터 기반 네트워크 운용기술이 최근 각광을 받고 있어 주목할 필요가 있다. Segment routing은 end-host 혹은 edge router가 플로우의 경로를 결정할 수 있게 해주며, DetNet에서의 표준안들에서 제시하는 바대로 포워딩 기능도 결정할 수 있게 하여, 사용자의 의지를 적극적으로 반영하며, 더욱 상세한 서비스 차별화가 가능하다.

2.2 플루이드 모델 기반의 Fair queuing 기술

발신지와 목적지가 동일한, 같은 응용에 속하는 패킷들의 집합인 플로우를 유체(fluid)로 보고 이를 바탕으로 중계 노드에서의 적절한 스케줄링으로 모든 플로우들에게 정확하게 요구한 만큼의 서비스를 제공해 주는 기술이 1990년대부터 제시됐다. Generalized processor sharing (GPS)은 전송할 데이터를 유체(fluid)로 보고 이상적인 서비스 순서를 결정하는 패러다임을 제시했고, 이를 패킷 환경에서 구현한 PGPS (혹은 Weighted fair queuing)는 이러한 유형의 패킷 기반 스케줄러들의 선구적 역할을 하였다[1]. PGPS는 아래와 같은 수식으로 도출된 완료 시간(Finish time)의 오름 차순으로 패킷의 서비스 순서를 결정한다.

$$F(p) = \max\{F(p-1), V(A(p))\} + L(p)/r. \quad (1)$$

여기서 p는 플로우의 p번째 패킷, A(p)는 패킷 p가 노드에 도착한 시간, L(p)는 p의 길이, r은 p가 속한 플로우에 할당된 서비스 rate을 의미한다. V(t)는 virtual time function이라고 부르며 시간 t에 서비스받는 플로우들의 r의 총합과 링크 용량의 비율을 곱해준 값이다. Virtual time은 플로우가 할당받은 속도보다 빠른 속도로 서비스받는 상황에서, 뒤늦게 시작된 플로우들이 상대적으로 작은 완료 시간을 가짐으로써 기존 플로우들보다 상당 기간 우선 서비스받는 불공평한 상황을 막아준다. 완료 시간 F(p)는 이상적인 플루이드 환경을 가정했을 때 공정하게 산출한 p의 완

료 시간을 나타내며 이 값이 작은 순서대로 노드의 서비스를 받는다. 완료 시간은 패킷이 노드에 도착한 순간 계산할 수 있으며, 따라서 버퍼에 저장하기 이전에 이미 패킷의 메타 데이터 형태로 노드 내에서 기록하여 사용할 수 있다. 일반적으로 플로우 별 큐를 두고, 해당 큐는 FIFO로 관리하며, 스케줄러는 플로우 별 큐의 HoQ를 살펴보아 가장 작은 완료 시간을 가진 큐를 서비스하는 형식으로 구현된다. 혹은 하나의 큐에 모든 패킷을 넣고 완료 시간의 값에 따라 큐의 중간에 끼워 넣는 구현도 가능하다. (1)의 핵심은, 최악의 경우 즉 모든 플로우들이 active 하여 링크가 full로 사용되는 경우, 플로우에 속한 이전 패킷 대비 L(p)/r 만큼의 간격으로 서비스를 받게 한다는 것이다. 그러면서 동시에 작업보존방식 스케줄러를 사용함으로써 사용되지 않고 낭비되는 링크자원이 없도록 방지하고 있다.

(1)을 구하기 위해 플로우의 F(p-1), 즉 이전 패킷의 완료 시간을 기억하고 있어야 한다. 패킷이 인입되면 어떤 플로우에 속하는지 찾아내어 해당 플로우의 최근 패킷의 완료 시간을 알아내어야 한다. 이 최근 패킷의 완료 시간 F(p-1)이 소위 말하는 '플로우 상태'를 대표하는 값이다. 이러한 상태 정보를 기억하고 있다가 읽어야 하는 점이 수백만 개의 플로우를 관리하는 코어 노드 입장에서 상당한 복잡도를 의미하여, 인터넷에서 이러한 PGPS 계열의 스케줄러가 실제로는 사용되지 않는 가장 주요한 이유가 되었다.

한편 V(t)를 계산하는 일도, 현재 서비스받는 플로우들을 실시간으로 추적해서 이들의 r의 총합을 계산해야 하는 복잡성을 내포하고 있다. 플로우들의 시작이 임의의 짧은 시간 동안 무수히 많을 수 있다는 것을 고려하면 상당이 어려운 계산임이 틀림없다. 따라서 V(t)를 정확하게 계산하는 대신 이를 간단한 방법으로 추정하는 방안들이 제시됐다[12,13,14]. 이 중 Virtual clock[12]은 V(t) 대신 현재시간 t를 사용해서 완료 시간을 결정한다. Self-clocked FQ[13]은 V(A(p)) 대신 최근 서비스받은 패킷(다른 플로우 소속일 수 있는)의 완료 시간을 사용한다.

Stiliadis는 이러한 일련의 스케줄러들이 rate-proportional server (RPS) 그룹에 속할 수 있으며, 이의 패킷 기반 구현 기술인 packetized RPS (PRPS)로 구현될 수 있음을 보였다[15]. 더 나아가 이러한 PRPS들이 모두 다음과 같이 플로우 i에 대해서 지연 시간 D_i을 보장한다는 것을 증명하였다.

$$D_i(t) \le \frac{\sigma_i}{r_i} + \sum_{j=1}^{H} \Theta_j^{S_j}. \quad (2)$$

σ와 ρ는 각각 플로우의 최대 버스트 크기와 평균 인입 속도(할당된 서비스 rate)이다. Sj는 해당 플로우가 지나가는 서버(노드)를 의미하며 그러한 서버가 H개 있다는 것이다. 여기서 Θ는 플로우가 최초로 backlog 되었을 때부터 서비스 시작까지의 지연 시간을 의미하며 모든 PRPS에 대해서 동일하게 (3)과 같이 적용된다. L_{max}는 모든 플로우들에서의 최대 패킷 길이, Li는 플로우 i의 최대 패킷 길이, C는 서버 Si의 링크 용량이다.

$$\Theta_i^{S_i} = \frac{L_{\max}}{C} + \frac{L_i}{r_i}. \quad (3)$$

2.3 플로우 상태 정보 관리가 필요 없는 Fair queuing 기술

이러한 완료 시간 기반 스케줄링 기법은 플로우 별 상태 정보를 기억하고 있다가 해당 플로우의 패킷이 들어오면 이 정보를 추출해서 패킷의 완료 시간을 계산하고, 이를 다시 저장하는 작업이 필요하다. (1)의 F(p-1)이 상태 정보를 대표한다면, 새로 계산한 F(p)가 해당 플로우의 새로운 상태가 되는 것이다. 이러한 복잡성이 수백만 개의 플로우가 동시에 존재하는 코어 노드에서 완료 시간 기반 스케줄링을 구현하기 힘든 이유이다. 따라서 $F_h(p)$를 h번째 노드에서의 p의 완료 시간이라고 한다면, $F_h(p-1)$ 대신 $F_0(p)$으로 $F_h(p)$를 추정하는 방법이 제안되었다. 이러한 방식을 core-stateless fair queuing (CSFQ)이라고 한다[16,17]. 특히 이러한 연구들의 시초인 Stoica의 연구[16]에서는 다음과 같은 방식으로 $F_h(p)$를 도출한다. 먼저 $E_h(p)$를 서비스를 시작할 수 있는 eligible time으로 정의한다. 각 패킷은 eligible time이 지난 후에 서비스를 받을 수 있는 후보가 된다. 이들 후보 중 최저의 $F_h(p)$ 값을 가지는 패킷이 서비스를 받게 된다.

$$\begin{aligned} E_0(p) &= A_0(p). \\ E_h(p) &= A_h(p) + G_{h-1}(p) + \delta_h(p). \\ F_h(p) &= E_h(p) + L(p)/r. \end{aligned} \quad (4)$$

여기서 $G_h(p)$는 h에서의 지연 시간 여유분('완료 시간'과 실제 서비스가 완료된 시간과의 차이)이며 $\delta_h(p)$는 $E_h(p)$가 $E_{h-1}(p)$보다 항상 크도록 강제해주는

지연 시간이다. 이 항목으로 플로우 내 패킷 간의 서비스 순서를 보장해 준다. 결과적으로 $E_h(p)$는 $E_0(p)$의 함수로 나타낼 수 있다. 즉, $E_0(p)$, $G_h(p)$, r 등을 패킷의 헤더에 저장해서 다음 노드로 보내면 플로우 상태 정보의 관리 없이 (4)의 eligible time과 완료 시간을 도출할 수 있다.

(4)의 핵심 아이디어는 (1)의 $F_h(p-1)$대신 $F_0(p)$를 사용한다는 것이다. 예를 들어 어떤 플로우가 계속해서 패킷을 보내 첫 번째 노드에서 계속해서 backlog되어 있다고 하자. 이 경우 $F_0(p)$는 $F_0(p-1)+L(p)/r$로 계속 계산된다. 즉 완료 시간 F들의 간격이 L(p)/r로 유지된다. 이러한 특성은 다른 플로우의 상태와는 무관하다. 즉, 플로우의 완료 시간은 backlog 기간에는 L(p)/r로 패킷 길이와 플로우의 할당받은 서비스 rate만으로 유추할 수 있다. 하위노드들에서도 backlog 기간에는 $F_0(p)$와 유사한 간격으로 유지할 수 있다. (4)로 계산되는 $F_h(p)$는 전역 완료 시간을 바탕으로 다양한 지연 시간 항목들을 노드마다 추가하여 수정된 값으로 볼 수 있다.

(4)를 (1)과 비교하면 CSFQ에 대해서 다음과 같은 사실을 알 수 있다.

Eligible time이 될 때까지 패킷을 서비스할 수 없다. 즉, 비 작업보존형 (non work-conserving) 방식이다.

$G_h(p)$로 패킷 간의 지연 시간 변동 폭이 최소화되도록 한다.

플로우 내의 패킷 간 서비스 순서는 유지되지만, 노드가 지남에 따라, 같은 경로로 진행하는 다른 플로우 간의 작은 패킷이 큰 패킷을 따라잡는 현상이 발생한다.

특히, [16]은 비 작업보존형으로 동작하여 평균 지연 시간이 많이 늘어난다는 단점과, 자세히 서술하지는 않았으나, $E_h(p)$의 계산이 복잡하다는 단점이 있다.

Ⅲ. 작업보존 방식 stateless fair queuing

3.1 코어 노드에서의 전역 완료 시간 도출

본 논문에서는 플로우의 네트워크 인입 후 최초 노드에서의 완료 시간에 이미 패킷 간의 공정한 서비스 시간 간격 정보가 포함되어 있다는 점을 주목한다. 매 노드에서 새롭게 완료 시간을 도출하는 대신, 최초 노드에서 계산된 완료 시간 정보를 활용하여 하위노드에서의 완료 시간을 도출하는 방안을 제시한다.

먼저 플로우가 최초로 인입하는 노드 0에서는 Virtual clock과 같은 방식으로 완료 시간을 계산한다.

$$F_0(p) = \max\{F_0(p-1), A_0(p)\} + L(p)/r. \quad (5)$$

여기서 $F_0(p)$는 패킷 p가 노드 0에 인입하는 시점에서 계산한 완료 시간이며, $A_0(p)$는 패킷 p가 노드 0에 인입하는 시점의 시간이다. $F_0(0)=0$으로 정의된다. 본 논문에서는 $F_0(p)$를 패킷의 전역 완료 시간(global finish time)이라고 명명한다.

한편, core 노드 h (h>0) 에서의 완료 시간은 (6)과 같이 계산한다.

$$F_h(p) = F_{h-1}(p) + d_{h-1}(p). \quad (6)$$

여기서 $d_h(p)$는 노드와 패킷의 함수이며, 대략 노드 h에서 패킷 서비스 처리에 걸리는 시간으로 설명할 수 있다. 다만 이때 $d_h(p)$는 non-decreasing 함수여야 한다. 이는 패킷 간 서비스 순서가 바뀌지 않도록 하기 위함이다. 예를 들어 $d_h(p)$ 값을, 노드의 busy period의 시작 시점부터 p의 인입 시점까지 측정된 최대지연 시간으로 구현할 수 있다.

(5)와 (6)을 사용하면 코어 노드에서 완료 시간을 계산할 때, $F_{h-1}(p)$와 $d_{h-1}(p)$ 값만이 필요하며, 따라서 코어 노드에서의 플로우 별 상태 정보의 저장 및 관리가 필요하지 않다. 이들은 패킷의 헤더에 메타 데이터 형태로 저장해서 전달될 수 있다. 모든 노드에서 $F_h(p)$와 $d_h(p)$로의 업데이트가 필요하며, $d_h(p)$의 경우 패킷의 이탈 (departure) 시점에 업데이트하여야 한다.

(6)의 특별한 예로 다음과 같은 구현도 가능하다.

$$F_h(p) = F_{h-1}(p) + d_{h-1}. \quad (7)$$

d_h는 노드만의 함수이며 경우에 따라 노드 간 사전 통신을 통해 정보를 전달받을 수도 있다.

(4) 혹은 (5)와 비교했을 때 (7)은 L(p)/r 만큼을 매 노드마다 증가시켜 주지 않는다는 것을 알 수 있다. 이는 전체 네트워크를 하나의 노드로 보고, 최초 설정한 패킷 간 거리를 유지하는 것이 공평할 수 있다는 점을 반영한다. 네트워크의 구조와 플로우의 홉 수를 모르는 사용자로서는 이러한 편이 합리적이다.

본 연구에서는 같은 경로의 모든 패킷 간 서비스 순서가 경로 중간에 바뀌는 일을 방지한다. 이 방식을 사용하면 기존의 복잡한 정렬 알고리즘이 필요했던 완료 시간 기준 스케줄링[18]이; 입력 port 별 FIFO queue를 유지하며, 해당 queue들의 HoQ들의 완료 시간만을 비교하는 것으로 전체 저장된 패킷 중 최소 완료 시간을 가진 패킷을 알아내는 것으로 구현할 수 있

다. 이러한 간단한 스케줄러의 구성으로 코어 노드에서의 확장성을 확보할 수 있다. 그림 1에 이러한 구현 예를 도시하였다. 모든 트래픽을 높은 우선순위와 낮은 우선순위의 트래픽으로로 구분하고, preemption이 가능하도록 설정하였다. 이렇게 함으로써, 높은 우선순위 트래픽은 낮은 우선순위 트래픽의 존재 여부와 상관없이 처리될 수 있다. 패킷이 도착하면 입력 포트별 큐에 저장하면서 FT를 계산하여 기입한다. 이때 패킷의 메타 데이터 $F_{h-1}(p)$와 $d_{h-1}(p)$를 읽어 (6)으로 $F_h(p)$를 도출한다. 노드 내에서 도출한 $d_h(p)$와 함께 새로운 메타 데이터로 패킷에 기재한다.

한편, (7)의 식으로 구한 $F_h(p)$는 d_h의 값에 따라 다른 경로로 core 노드에 도착한 플로우들 간의 형평성(fairness)에 영향을 줄 수 있으며, 따라서 이를 해소할 방안이 필요하다. (5)가 $F_0(p) \geq A_0(p) + L(p)/r$를 만족해야 하는 이유가 이와 같다. 즉 플로우들의 완료 시간 F를 현재 시간 t에 연동하여 플로우 간 형평성을 보장한다. 본 연구는 아래의 간단한 방법으로 이를 달성한다.

$$d_h = U_h. \tag{8}$$

여기서 U_h는 노드 h에서 모든 플로우의 패킷에 대해서 보장하는 지연 시간 상한이다. 즉,

$$A_h(p) + U_h \geq A_{h+1}(p). \tag{9}$$

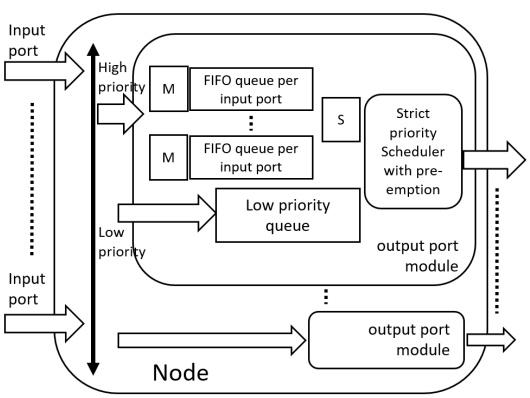

그림 1. 순서 보존형 전역 완료 시간 기반 기술을 적용한 노드 구조의 예. M은 완료 시간 계산 및 기입, S는 FIFO queue의 head를 살펴보고 완료 시간 최소값을 가지는 패킷을 찾아서 서비스하며, metadata $d_h(p)$를 update 함.
Fig. 1. Example node architecture having schedulers with the preserved service order between packets from the same input. M obtains FT and writes it on packet header. S examines the HoQ of FIFO queues, updates metadata $d_h(p)$, and services the packet having minimum FT.

을 만족하는 값이다.

Theorem 1. (5), (6)과 (8)의 방식으로 구현한 $F_h(p)$는 $F_h(p) \geq A_h(p) + L(p)/r$를 만족한다.

Proof: 귀납법을 사용하여 증명한다. 먼저 $F_0(p)$에 대해서 만족함을 보이고, h>0인 $F_h(p)$에 대해서 보인다. (5)에 따라

$$F_0(p) = \max\{F_0(p-1), A_0(p)\} + L(p)/r \\ \geq A_0(p) + L(p)/r.$$

h>0인 h에 대해서, $F_{h-1}(p) >= A_{h-1}(p)+L(p)/r$ 를 만족한다고 하자. (6)과 (8)에 의해

$$F_h(p) = F_{h-1}(p) + U_{h-1}.$$

따라서 (9)에 의해,

$$F_h(p) - A_h(p) = F_{h-1}(p) + U_{h-1} - A_h(p) \\ \geq F_{h-1}(p) - A_{h-1}(p) \\ \geq L(p)/r.$$

이렇게 모든 h, h>=0 에 대해 $F_h(p) >= A_h(p)+L(p)/r$ 임을 증명하였다. ■

(8) 대신 택할 수 있는 또 다른 구현 예는 $d_h = U$ 이다. 여기서 $U = \max_h\{U_h\}$, h는 네트워크의 모든 노드이다. 이렇게 모든 노드에서 같은 값의 delay를 더하게 되어 경로가 다른 플로우 간 FT 증가폭이 동일하게 되어 더 공정한 서비스를 기대할 수 있다.

3.2 도출된 전역 완료 시간 기반 fair queuing 스케줄러의 특성

본 연구에서 차용하는 [12]의 Virtual clock은 V(t)를 따로 구하지 않고 현재 시간을 사용한다. 즉, V(t)=t이다. 이러한 단순한 V(t) 함수가 [15]에서 'system potential'임이 검증되었다. Potential은 fluid 모델인 RPS상에서의 virtual time function의 다른 표현이다. 시스템 potential, P(t), 는 backlog된 플로우들의 플로우 별 potential, $P_i(t)$, 보다 항상 작거나 같도록 정의되어야 한다. 즉,

$$P_i(t) \geq P(t), i \in B(t). \tag{10}$$

여기서 B(t)는 시간 t에 backlog된 플로우의 집합이

다. 본 연구에서 제안하는 스케줄러의 플로우 별 potential은 (5)와 (7)로 정의된다. [15]에서 다음 조건을 만족하는 플로우 별 potential을 가진 스케줄러는 PRPS이며 따라서 (2)와 (3)으로 지연 시간 상한이 보장된다는 것이 증명되었다.

$$P_i(t) - P(t) \le \Delta, i \in B(t). \quad (11)$$

여기서 Δ는 음수가 아닌 임의의 유한한 상수 (finite constant)이다. 즉, 플로우 별 포텐셜이 시스템 포텐셜보다 항상 크면서, 그 둘의 차이가 유한한 값으로 한정(bounded)되면 Pi(t)를 사용하는 스케줄러는 PRPS이다. 지금부터는 (12)과 (13) 두 수식으로 정의되는 스케줄러가 PRPS임을 보인다.

Theorem 2. 유한한 U_h를 가지면서 아래와 같이 정의되는 스케줄러는 PRPS이다.

$$F_0(p) = \max\{F_0(p-1), A_0(p)\} + L(p)/r. \quad (12)$$

$$F_h(p) = F_{h-1}(p) + U_h, \text{ for } h > 0. \quad (13)$$

증명: 위 스케줄러의 시스템 포텐샬은 P(t)=t로 정의된다. 한편 $P_i(t)$는 piecewise linear function이며, $A_h(1)$, $A_h(2)$, $A_h(3)$,... 에서 h=0인 경우에 $P_i(t)=F_0(p)$; h≠0인 경우에는 $P_i(t)=F_h(p)$의 값을 가진다. $A_h(p)$가 아닌 t에서는, $A_h(k-1) < t < A_h(k)$를 만족하는 $A_h(k)$와 $A_h(k-1)$에서의 $P_i(t)$값을 연결한 선 위의 값을 가진다. 우리는 (12)와 (13)으로 정의된 스케줄러가 (10)과 (11)을 만족함을 보인다.

증명은 h=0인 경우와 그 밖의 경우로 나누어 진행한다. 본 증명에서 Δ는 특정 상수가 아니라 임의의 유한한 양의 상수를 의미하며, 각 수식에서 값이 달라질 수 있음에 유의하자.

Part 1: h=0인 경우. 즉 최초 인입 노드에서 (12)의 스케줄러가 PRSP인 것을 보이는 방법은 아래 Part 2와 상당히 유사하며, (12)이 [12]의 Virtual clock 스케줄러와 동일하여 [15]에서 이미 PRSP임이 증명되었기 때문에 생략한다.

Part 2: h≠0인 경우의 증명은 귀납적인 방법으로 증명한다. $t=A_h(1)$, $A_h(2)$, $A_h(3)$,...에서만 증명하는 것으로 충분하다. 먼저 h=1인 경우를 고려하자. 이 경우,

$$\begin{aligned} & P_i(t) - P(t) = F_1(p) - A_1(p) \\ & = F_0(p) + U_0 - A_1(p) \\ & \ge F_0(p) - A_0(p) \\ & = \max\{F_0(p-1) - A_0(p), 0\} + L(p)/r \\ & \ge L(p)/r > 0. \end{aligned} \quad (14)$$

한편, U_0이 유한한 값을 가지며, $A_1(p)$가 양수이기 때문에,

$$\begin{aligned} & P_i(t) - P(t) = F_1(p) - A_1(p) \\ & = F_0(p) + U_0 - A_1(p) \\ & \le \Delta. \end{aligned} \quad (15)$$

이제 h>1인 경우를 생각하자. 먼저 h-1인 경우에 (14)와 (15)가 성립한다고 가정한다. 그렇다면,

$$\begin{aligned} & P_i(t) - P(t) = F_h(p) - A_h(p) \\ & = F_{h-1}(p) + U_{h-1} - A_h(p) \\ & \ge F_{h-1}(p) - A_{h-1}(p) \\ & \ge L(p)/r > 0. \end{aligned}$$

이며, $U_{h-1} < \Delta$ 이므로,

$$\begin{aligned} & P_i(t) - P(t) = F_h(p) - A_h(p) \\ & = F_{h-1}(p) + U_{h-1} - A_h(p) \\ & = F_{h-2}(p) + U_{h-2} - A_{h-1}(p) + U_{h-1} \\ & \quad - A_h(p) + A_{h-1}(p) \\ & \le \Delta + U_{h-1} - (A_h(p) - A_{h-1}(p)) \\ & \le \Delta. \end{aligned}$$

따라서 모든 h와 p에서 (10)와 (11)을 만족한다. ■

3.3 성능 비교

서비스 순서 보존형 전역 완료 시간을 적용한 네트워크 단대단 지연 시간은, Theorem 2에서 증명된 바와 같이, 다른 PRPS 스케줄러를 적용했을 때와 마찬가지로 (2)와 (3)으로 주어진다. 이들 PRPS들의 성능을 기존 IEEE와 IETF에서 제시한 ATS, FAIR와 비교하면 아래와 같다. 먼저 그림 2와 같이 9개의 노드가 격자 형태로 네트워크를 구성한다. 이 네트워크는 네 개의 내부 격자가 있는데 각각은 그림 2와 같이 한 방향으로 플로우들이 순환한다.

모두 8가지 종류의 플로우 경로가 아래 표와 같이 존재한다.

모든 플로우들은 동일한 파라미터, requested rate 480Kbps, 최대 버스트 크기 = 최대 패킷 길이 = 2400bit를 가진다. 링크의 용량은 100Mbps이다. 그림

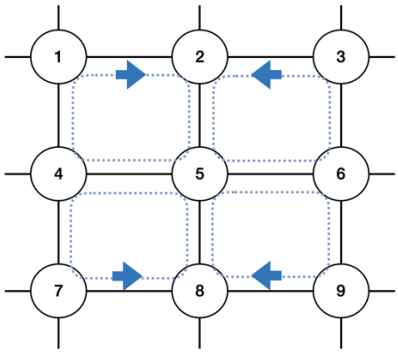

그림 2. 성능 비교를 위한 네트워크 토폴로지
Fig. 2. Network Topology for performance comparison

표 1. 그림 2 네트워크에서 플로우의 경로 종류
Table 1. The flow routes type in the network of Fig. 2

Flow type	Route
0	1-2-5-6-9
1	4-1-2
2	4-7-8
3	7-8-5-6-3
4	3-2-5-4-7
5	6-3-2
6	6-9-8
7	9-8-5-4-1

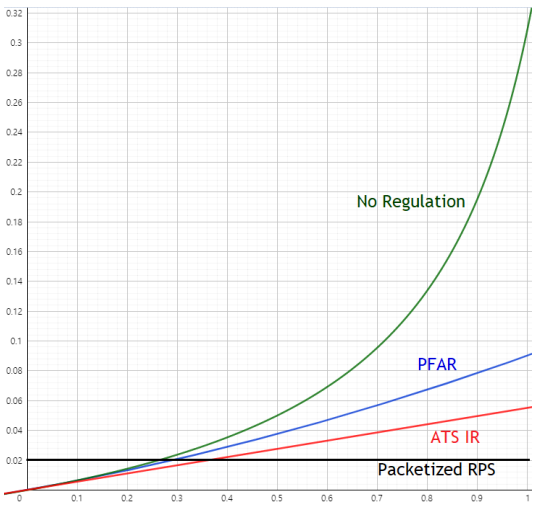

그림 3. 링크 사용률에 따른, regulation 없는 일반 스케줄러, PFAR, ATS IR과 본 논문에서 제안한 PRPS의 단대단 네트워크 지연 시간 상한 비교 (단위: 초)
Fig. 3. End-to-end latency bounds, with varying link utilization, of FIFO scheduler with no regulation, PFAR, ATS IR, and the proposed PRPS schedulers (in second).

3에서 사용률은 해당 플로우가 얼마나 많은가 하는 것을 나타낸다.

그림 3에서와 같이 플로우 보호(protection) 특성을 가지고 있는 PRPS는 사용률과 상관없이 일정한 단대단 지연 시간 상한을 제공한다. PFAR, ATS 시스템과 regulation없이 FIFO service를 제공하는 경우의 성능에 대한 자세한 내용은 [11]을 참고하라.

Ⅳ. 결 론

본 논문에서는 플로우의 네트워크 인입 후 최초 노드에서 규정하는 패킷의 완료 시간(Finish time)에 이미 패킷 간의 공정한 서비스 시간 간격 정보가 포함되어 있다는 점을 주목하였다. 최초 노드에서 도출한 완료 시간을 패킷에 메타 데이터 형태로 기록하여 이를 바탕으로 하위노드들에서 플로우 별 상태 정보 관리 없이 간단한 방법으로 완료 시간을 업데이트하여 이를 바탕으로 패킷 스케줄 순서를 결정하는 방안을 제시하였다. 특히, 제시한 스케줄러는 최초 노드에서의 완료 시간에 노드 고유의 상태 정보만을 더해서 진행해 나가는 특성으로 인해, 같은 경로를 지나가는 모든 플로우의 패킷 간 서비스 순서가 바뀌지 않을 수 있는 특성을 가진다. 이에 따라 간단한 input port 별 FIFO 큐를 이용해서 해당 큐의 HoQ만을 비교함으로써 최소 완료 시간을 가진 패킷을 손쉽게 찾아 낼 수 있다는 장점을 가진다. 더 나아가 제시한 스케줄러의 일례가 PRPS임을 증명하였다. 따라서 플로우 별 보호가 가능하여 기존 표준들에서 제시한 지연 시간 상한의 수준을 훨씬 뛰어넘는 성능을 보인다. 동시에 확장성도 확보하여 기존 기술 대비 획기적인 발전을 이루었다.

향후 virtual time function을 고도화하여, 제시한 완료 시간 도출 알고리즘을 일반화해서 플로우 간 fairness를 향상하도록 한다. 한편 IETF DetNet WG과 ITU-T SG13에 표준안으로 제안하여 기술의 보편화를 추구한다. Segment routing 등 패킷에 메타 데이터를 저장하여 사용자별 서비스 다변화를 추구하는 방향으로 인터넷이 진화하고 있다는 점을 감안하면, 본 연구의 결과물도 무리 없이 표준화할 수 있을 것으로 예상한다.

References

[1] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node

case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344-357, 1993.

[2] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," *IEEE/ACM Trans. Netw.*, vol. 4, no. 3, pp. 375-385, Jun. 1996. (https://doi.org/10.1109/90.502236)

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An architecture for differentiated services,* No. rfc2475, 1998.

[4] L. Thomas, J.-Y. Le Boudec, and A. Mifdaoui, "On cyclic dependencies and regulators in time-sensitive networks," in *Proc. IEEE RTSS*, pp. 299-311, York, U.K., Dec. 2019.

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

[6] E. Mohammadpour, E. Stai, M. Mohiuddin, and J. Y. Le Boudec, "Latency and backlog bounds in time-sensitive networking with credit based shapers and asynchronous traffic shaping," in *IEEE 2018 30th ITC 30*, vol. 2, pp. 1-6, Sep. 2018.

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

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

[9] J. L. Messenger, "Time-sensitive networking: An introduction," *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 29-33, 2018.

[10] J. Joung, J-D. Ryoo, T-S. Cheung, Y. Li, and P. Liu, "Asynchronous deterministic networking framework for large-scale networks," *IETF DetNet WG*, Internet draft, draft-joung-detnet-asynch-detnet-framework-00, Jul. 2022.

[11] J. Joung, J. Kwon, J.-D. Ryoo, and T. Cheung, "Asynchronous deterministic network based on the DiffServ architecture," *IEEE Access*, vol. 10, pp. 15068-15083, 2022.

[12] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," in *Proc. ACM Symp. Commun. Architectures & Protocols*, pp. 19-29, Aug. 1990.

[13] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. INFOCOM'94 Conf. Comput. Commun.*, pp. 636-646, 1994.

[14] D. Stiliadis and A. Varma, "Efficient fair queueing algorithms for packet-switched networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 2, pp. 175-185, 1998.

[15] D. Stiliadis and A. Varma, "Rate-proportional servers: A design methodology for fair queueing algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 2, pp. 164-174, 1998.

[16] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 81-94, 1999.

[17] Z.-L. Zhang, Z. Duan, and Y. T. Hou, "Virtual time reference system: A unifying scheduling framework for scalable support of guaranteed services," *IEEE J. Sel. Areas in Commun.*, vol. 18, no. 12, pp. 2684-2695, 2000.

[18] R. Bhagwan and B. Lin, "Design of a high-speed packet switch with fine-grained quality-of-service guarantees," in *2000 IEEE ICC 2000, Global Convergence Through Commun. Conf. Record*, vol. 3, pp. 1430-1434, 2000.

[19] *IETF Source Packet Routing in Networking (spring) Working Group Home page*, https://datatracker.ietf.org/wg/spring/

[20] *IETF Deterministic Networking (detnet) Working Group Home page*, https://datatracker.ietf.org/wg/detnet/about/

정 진 우 (Jinoo Joung)

1992년 2월 : KAIST 전자공학과 학사
1994 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, 임베디드시스템
[ORCID:0000-0003-3053-9691]