논문 19-44-06-11 The Journal of Korean Institute of Communications and Information Sciences '19-06 Vol.44 No.06
https://doi.org/10.7840/kics.2019.44.6.1105

네트워크 지연시간 보장을 위한 개선된 레귤레이팅 스케줄러

정 진 우◆, 권 주 혁*

Improved Regulating Scheduler for Network Delay Guarantee

Jinoo Joung◆, Juhyeok Kwon*

요 약

단대단 네트워크 지연시간을 수 밀리 초에서 수 초까지 엄격하게 제한하는 응용분야가 늘어나고 있다. 현재 대부분 네트워크는 트래픽의 클래스(class)에 따라 큐를 할당하고 서비스하는 스케줄러를 채택하고 있으나, 이런 방안으로는 사이클(Cycle)이 형성된 네트워크에서 사이클을 따라서 최대 버스트(burst)가 무한히 증가하기 때문에 지연시간을 제한할 수 없다. 이러한 최대 버스트를 강제로 제한하는 트래픽 레귤레이터를 클래스 기반 스케줄러와 연속해서 배치하여 버스트의 크기를 줄이는 방안이 IEEE 802.1 등의 국제 표준에 반영되었다. 더 나아가 스케줄러와 레귤레이터 기능을 동시에 수행하는 레귤레이팅 스케줄러(regulating scheduler, RSC)가 제안되었다. RSC를 입력포트별로 구분된 큐에 적용하면 수 밀리 초 수준의 단대단 지연시간 보장이 가능하다. 하지만 레귤레이터 기능을 포함하는 비작업보존형 스케줄러인 RSC는 필연적으로 작업보존형 스케줄러에 비해 평균 지연시간 등의 확률적 성능이 떨어진다. 본 연구에서는 작업보장형 스케줄러에 좀 더 가까운 확률적 성능을 보이면서도 RSC의 최대성능보장 기능을 유지하는 개선된 RSC를 제안하고 이의 성능을 시뮬레이션을 통해 검증하였다.

Key Words : End-to-end delay guarantee, TSN, DetNet, scheduler, regulator

ABSTRACT

Numerous applications require strict bound on the end-to-end network delay, which is ranged from a few msec to a few seconds. The class-based systems adopted in the actual deployments, however, cannot provides a bounded delay in networks with cycle, since the maximum burst grows infinitely along the cycled path. Regulators implemented next to a scheduler, which limit the maximum burst are adopted as a viable solution. Furthermore, non-work conserving scheduler that also performs regulating function, called Regulating Scheduler (RSC), is suggested. RSC guarantees a few millisecond's end-to-end delay when applied to input-port based queues. The statistical performance of non-work conserving RSC is inevitably worse when compared to work conserving schedulers. This paper proposes an improved RSC. It is shown by simulation the improved RSC gives a closer statistical performance to work-conserving scheduler, while still guarantees the maximum delay just like the RSC.

※ 이 성과는 2018년 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2016R1D1A1B03932066).
◆ First Author : Sangmyung University, Department of Human Intelligence Information Engineering, jjoung@smu.ac.kr, 정회원
* Sangmyung University, Department of Human Intelligence Information Engineering, juhuk98@naver.com
논문번호 : 201904-041-B-RN, Received April 6, 2019; Revised April 24, 2019; Accepted April 25, 2019

Ⅰ. 서 론

스마트 팩토리, 차량 간 통신, 차량 내 통신, 전문 음향 네트워크, 대규모 전력 제어 망 등 다양한 응용 분야에서 수 밀리 초에서 수초까지의 엄격한 단대단 네트워크 지연시간 (end-to-end network delay) 제한(bound)을 요구하고 있다. IEEE 802.1 time sensitive network (TSN)[1]과 IETF deterministic network (DetNet)이 대표적인 관련 국제 표준이다. TSN의 전신인 residential Ethernet과 이의 후속 기술인 AVB는 A/V 응용을 고려하여 비교적 높은 대역폭을 가지는 소수의 플로우(flow; 같은 발신지/목적지, 같은 application을 가지는, 시간적으로 근접한 패킷들의 집합)가 기술 적용대상인 데 반해서[2], TSN은 다수의 저대역폭 산업용 sensor-actuator 네트워크에서의 플로우까지 고려하고 있다.

단대단 지연시간을 보장하는 잘 알려진 해결책으로 integrated services(IntServ) 프레임워크를 들 수 있다. IntServ 프레임워크에서의 플로우 기반 스케줄러는 플로우의 수가 N일 때 O(N)혹은 O(logN)의 복잡도를 가진다. 이러한 복잡도가 구현의 제한요소가 되므로 현재의 네트워크는 트래픽의 우선순위(priority) 등으로 구분된 클래스(class)별로 큐를 운용하는 스케줄러를 채택하고 있으나, 해당 스케줄러는 사이클(cycle)이 형성된 네트워크에서는 사이클을 따라서 최대 버스트가 계속 증가하기 때문에 지연시간을 제한할 수 없다. 이러한 최대 버스트를 강제로 제한하는 token bucket 방식의 트래픽 레귤레이터를 클래스 기반 스케줄러의 앞뒤에 배치하여 버스트의 크기를 줄이면서 동시에 스케줄러의 복잡도를 낮추려는 방안이 제시되었다. TSN과 DetNet은 이러한 기술을 표준으로 채택하고 있다. 하지만 레귤레이터가 플로우의 상태 정보를 필요로 하므로 다시 복잡도가 증가하는 모순이 존재한다. 최근 이의 대안으로 스케줄러가 레귤레이션 기능을 동시에 수행하는 regulating scheduler(RSC)가 제안되었다[3]. RSC는 특정 큐가 일시적으로 링크의 서비스를 독점하지 못하도록 비 작업보존 방식으로 동작하는 것을 특징으로 한다. 더 나아가 이러한 비 작업보존 방식의 스케줄러를 비어 있는 큐에 가상의 패킷을 생성하여 이를 서비스하게 함으로써 간단하게 구현하였다. 하지만 모든 레귤러이터가 그렇듯이 필연적으로 평균 지연시간이 늘어나게 된다. 본 논문에서는 RSC 큐의 서비스 순서를 backlog 기간마다 임의로 재배열하여 실제 패킷을 가상 패킷보다 가능한 한 먼저 서비스하게 함으로써 확률적인 성능 특성을 개선하였다. 본 논문은 다음과 같이 구성된다. 다음 장에서 관련 연구 및 표준화 동향을 살펴보고, 3장에서 상기한 개선된 RSC의 알고리즘을 제시하고 분석한다. 4장에서는 시뮬레이션을 통해 개선된 RSC의 성능을 평가하고 기존 스케줄러들과 비교한다. 마지막으로 5장에서 본 논문에 대한 결론과 향후 방향을 제시한다.

Ⅱ. 관련 연구

IntServ에서 품질보장을 위한 핵심모듈인 스케줄러는 각각의 플로우에게 공정하게 일정 속도 이상의 서비스를 제공해야 한다. 이렇게 각각의 플로우에게 일정 수준 이상의 서비스속도(service rate)를 정해진 시간(이를 latency라고 하며 delay와는 다름) 이전에 제공하는 스케줄러들을 latency-rate(LR) 서버라고 한다[4]. 하나의 플로우 i가 네트워크를 지나면서 LR 서버들만을 통과한다면, 이 플로우 i의 패킷들이 겪는 단대단 지연시간의 최대치는 다음과 같은 식으로 표현된다.

$$D_i \le \frac{\sigma_i - L_i}{\rho_i} + \sum_{j=1}^{k} \Theta_i^{S_j} \tag{1}$$

아래에서 자주 사용하는 수학 기호에 대해서 표로 정리하였다.

(1)의 의미는 다음과 같다. 경로 상에서 여러 개의 LR 서버를 지나더라도 플로우 지연시간의 최대치는 각각의 latency의 합에 최초 한 번의 최대 버스트에 의한 지연시간만을 더한 것이다. 이러한 LR 서버의 특성을 "Pay burst only once"라고 한다. 또한, LR 서버에 인입되는 플로우가 인입커브 (ρ_i, σ_i)를 따른다면, LR 서버를 통과한 후에는 인입커브 $(\rho_i, \sigma_i + \Theta_i^{S}\rho_i)$를 따

표 1. 수학 기호와 그 의미
Table 1. Notations and their meaning

Notation	Meaning
L_i	Max packet length of flow i
r	Link capacity
σ_i	Max burst size of flow i
ρ_i	Input data rate of flow i
ϕ_i	Quantum value assigned for flow i
$\Theta_i^{S_j}$	Latency of flow i at sever S_j
D_i	Delay experienced by packets of flow i

른다. 즉 최대 버스트 크기가 $\rho_i \Theta_i^S$ 만큼 증가하게 된다[4].

간단한 라운드 로빈 기반의 deficit round robin(DRR)[5]과 weighted round robin이 LR 서버에 포함된다. 퀀텀(quantum) 크기가 패킷의 최대길이보다 작은, 일반적인 DRR의 latency는 다음과 같이 주어진다[6].

$$\Theta_i^{DRR} = \frac{1}{r}\left[(F-\phi_i)(1+\frac{L_i}{\phi_i}) + \sum_{n=1}^{N} L_n\right]. \qquad (2)$$

여기서 F는 모든 플로우 ϕ_i들의 합이다. F를 프레임의 크기라고도 한다. DRR의 동작에 대한 자세한 설명은 [5]를 참조하라.

이런 라운드로빈 스케줄러도 플로우의 수가 많아지면 고속으로 동작하도록 구현하기 힘들다. 따라서 다양한 형태의 클래스 기반 스케줄러[7]들이 제시되었지만 1장에서 언급한 바와 같이 사이클이 네트워크 내부에 존재하면 지연시간 최대치를 보장하지 못한다[8]. 이에 따라 트래픽 레귤레이터를 모든 노드에 스케줄러와 같이 구현하는 방안이 제안되었다[8,9]. 여기서 제안된 레귤레이터는 그림 1과 같이 플로우별로 동작한다. 해당 시스템은 스케줄러의 복잡도를 낮추고, 병렬로 배치된 레귤레이터의 복잡도를 높이는 방법으로 구현 가능성을 높였으나 역시 실현되지는 않았다.

TSN TG에서 최근 제시된 asynchronous traffic shaping(ATS)은 상기한 시스템의 개선안이라 할 수 있다. AST 기술을 채용한 방식을 TSN 비동기식 방안이라고 부른다[10]. ATS의 핵심 아이디어는 입력포트별, 클래스별 트래픽 레귤레이션 기능을 출력 포트의

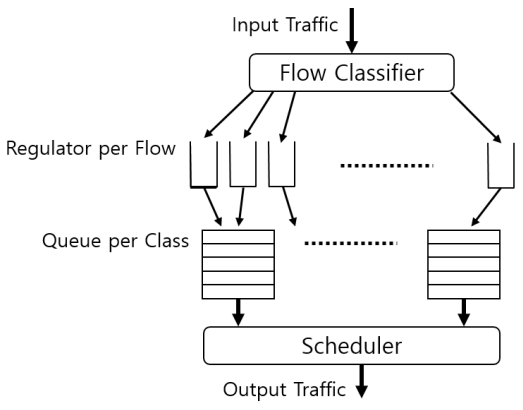


그림 1. [8]에서 제안된 레귤레이터-스케줄러 시스템 구조
Fig 1. Architecture for regulator-scheduler system in [8]

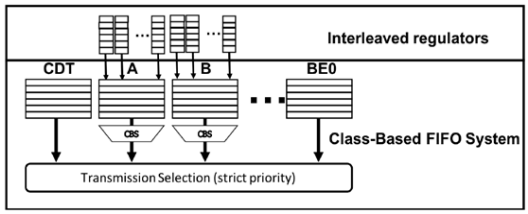


그림 2. ATS(Interleaved regulators)와 기존 Class 기반 FIFO 시스템이 공존하는 TSN의 비동기식 방안[10]
Fig. 2. TSN Asynchronous approach, where ATS(Interleaved regulator) and Class based FIFO system coexist[10]

시작점에서 구현하는 것이다. 다만 입력포트별 레귤레이션이라 하더라도 모든 플로우의 상태정보를 기억하고 있다가 큐의 가장 앞에 있는 패킷이 속한 플로우를 파악하고 해당 플로우의 상태정보에 따라 레귤레이션 여부를 결정해야 하는 복잡성이 내포되어 있다. 그림 2는 이 비동기식 방안을 도시한 것이다. Interleaved regulator와 클래스별 스케줄러가 구현되어 있다.

ATS에 비해 RSC는 플로우별 상태 정보를 필요로 하지 않는다[3]. RSC 중 DRR기반의 nw-DRR은 다음과 같은 algorithm으로 동작하는 비작업보존형 DRR 스케줄러이다[3].

1. 명시된 인입 속도 없이 인입하는 낮은 우선순위 트래픽 전체를 수용하는 플로우(v)를 상정한다. 해당 플로우는 실제 높은 우선순위 플로우들의 인입 속도의 총합과 서버의 capacity와의 차이만큼의 인입 속도를 가진 것처럼 취급된다 ($\rho_v = r - \sum_i \rho_i$).

2. 낮은 우선순위 플로우를 포함하여 모든 플로우의 큐가 항상 backlog 되도록, 큐의 서비스 차례 직전에 플로우의 큐가 비어 있으면 deficit 값을 0으로 만든 후 가상의 패킷을 생성한다. 생성되는 가상 패킷의 크기는 해당 플로우의 quantum 크기로 한다. 즉, 한번의 round에 서비스될 수 있는 최대 크기의 패킷을 생성한다.

3. 가상의 패킷을 서비스하는 동안 해당 큐에 실제 패킷이 도착하면 그 즉시 가상 패킷의 서비스를 멈추고, deficit값을 0으로 만든 후 다음 차례의 큐를 서비스한다.

4. 가상 패킷의 서비스가 완료되어도 실제로 링크로 전송하지는 않는다.

RSC는 결과적으로 모든 플로우가 비작업보존 방식으로 처리된다. 비작업보존 스케줄러인 RSC는 항상 모든 큐가 가상으로 backlog 되어 있으며, 플로우들의 가상의 인입 속도의 총합이 서버의 capacity와 같다.

Ⅲ. 개선된 RSC 알고리즘

먼저 nw-DRR의 성능을 분석해보자. 일반적인 DRR 스케줄러에 대해서 Theorem 1이 성립한다.

Theorem 1. 플로우 i가 시간구간 (a,b] 동안 계속 backlog되어 있다고 가정하자. (a,b] 동안 DRR turn이 플로우 i에게 k번 서비스를 제공한다고 하자. 이 기간 플로우 i가 받은 서비스의 총량 $W_i(a,b)$는 다음과 같이 제한된다.

$$k\phi_i - \delta_i^k \le W_i(a,b) \le k\phi_i + \delta_i^0 \quad (3)$$

여기서 δ_i^k는 i의 (a,b]기간의 첫 번째 라운드부터 세어 k번째 라운드의 끝 시점에서의 deficit값이다.

증명. [5]의 Theorem 4.2의 증명과 동일함. ■

Theorem 2는 DRR 분석과정의 핵심이며 [3]에서 제시되고 증명되었지만, 증명과정에 약간의 오류가 있어 본 논문에서 아래와 같이 정정한다.

Theorem 2. Backlog 기간 중의 임의의 기간 (a,b] 동안 DRR 서버가 플로우 i에 제공하는 서비스의 총량은 다음과 같이 제한된다.

$$W_i(t_0,t_k) \le \rho_i \frac{F_{\max}}{f}(t_k - t_0) + \frac{NL}{f}\phi_i + L_i \quad (4)$$

여기서 f는 이 기간 계속 backlog된 플로우들의 quantum값들의 총합이며 $F_{\max}$는 서버의 capacity와 플로우들의 인입 속도가 같은 가상의 상황에서의 quantum값들의 총합이다. N은 플로우의 수, L은 최대 패킷 길이이다.

증명. 여기에서 backlog 기간이란 하나 이상의 실제 패킷이 서비스를 기다리거나 서비스를 받고 있으면서 큐에 있는 연속된 시간 구간을 의미한다. a에서 시작해서 k번째 라운드가 끝나는 시점을 t_k라고 하자. $t_0 = a$이다. 하나의 라운드 기간, $(t_{k-1},t_k]$의 길이 $t_k - t_{k-1} = \frac{1}{r}\sum_{j\in B_k}(\phi_j + \delta_j^{k-1} - \delta_j^k)$이다. 여기서 B_k는 t_{k-1} 시점에서 backlog 되어 있어 서비스를 받는 플로우들의 집합이다. $(t_{k-1},t_k]$동안 B_k의 원소는 바뀌지 않는다. 여기서 B를 $(t_0,t_k]$동안 계속 backlog되어 있는 플로우들의 집합으로 정의하자. 모든 k에 대해서 $B \subset B_k$이다. 여기서 다음의 부등식이 성립한다. $t_k - t_{k-1} \ge \frac{1}{r}\sum_{j\in B}(\phi_j + \delta_j^{k-1} - \delta_j^k)$. 이 부등식을 k에 대해서 모두 합하면 다음이 성립한다.

$$\begin{aligned} t_k - t_0 &\ge k\frac{f}{r} + \frac{1}{r}\sum_{j\in B}(\delta_j^0 - \delta_j^k) \\ &\ge k\frac{f}{r} - \frac{\sum_B L_j}{r} \ge \frac{k}{r}f - \frac{N_B L}{r}. \end{aligned}$$

왜냐하면 $\Sigma_{j\in B}\delta_j^0 \ge 0$이고 $\Sigma_{j\in B}\delta_j^k \le N_B L$이기 때문이다 ($N_B$는 B에 속한 flow의 수, L은 최대 패킷 길이). 따라서 $k \le \frac{r}{f}(t_k - t_0 + \frac{N_B L}{r})$이다. Theorem 1로부터 $(t_0,t_k]$ 기간 동안 다음 부등식이 성립한다.

$$\begin{aligned} W_i(t_0,t_k) &\le k\phi_i + \delta_i^0 \le \frac{r}{f}(t_k - t_0 + \frac{N_B L}{r})\phi_i + \delta_i^0 \\ &= \frac{r}{f}(t_k - t_0)\phi_i + \frac{N_B L}{f}\phi_i + \delta_i^0 \\ &\le \rho_i \frac{F_{\max}}{f}(t_k - t_0) + \frac{NL}{f}\phi_i + L_i. \end{aligned}$$

왜냐하면 $L_i \ge \delta_i^0$이며 $\rho_i / r = \phi_i / F_{\max}$이기 때문이다. 즉, t_0를 포함하여 언제나 deficit 값은 최대 패킷 길이 L_i보다 클 수 없다. 플로우의 인입 속도와 quantum 값은 비례하며, $F_{\max}$는 서버의 capacity와 플로우들의 인입 속도가 같은 경우의 frame 크기이므로 r에 비례한다. 따라서 t_k와 t_{k+1} 사이의 임의의 시간 b에 대해서

$$\begin{aligned} W_i(a,b) = W_i(t_0,t_k) &\le \rho_i \frac{F_{\max}}{f}(t_k - t_0) + \frac{NL}{f}\phi_i + L_i \\ &\le \rho_i \frac{F_{\max}}{f}(b - a) + \frac{NL}{f}\phi_i + L_i \end{aligned}$$

이며 Theorem이 성립한다. ■

Theorem 2의 의미는, DRR 스케줄러가 큐에 제공하는 서비스의 총량을 제한할 수 있다는 것이며, 이를 적절히 이용하면 레귤레이터가 서비스 총량을 제한하는 기능을 스케줄러 자체에서 구현할 수 있다는 것이다. 구체적으로는 DRR이 nw-DRR로 동작하여 모든 큐가 항상 backlog 되어 있어서 $f = F_{\max}$이면, $W_i(a,b) \le \rho_i(b-a) + \frac{NL}{r}\rho_i + L_i$이 성립하며 시스

템 backlog 기간 중의 임의의 기간 (a,b)에서 token 생성률 ρ_i, bucket 크기 $\frac{NL}{r}\rho_i + L_i$의 token bucket 레귤레이터로 서비스를 제한하는 효과를 가진다. 모든 플로우의 인입 속도 ρ_i가 같은 경우 ρ_i/r가 1/N와 같다는 점을 감안하면 nw-DRR에 대해서 다음과 같은 부등식으로 근사할 수 있다.

$$W_i(a,b) \le \rho_i(b-a) + L + L_i \quad (5)$$

이렇게 레귤레이터 기능을 동시에 수행하는 nw-DRR은 최대 burst 크기를 제한하며, 따라서 최대 지연시간을 보장하지만, 필연적으로 평균 지연시간을 증가시킨다. 이를 아래와 같이 개선한 스케줄러를 제안하고 qnw-DRR(quick non-work conserving DRR)이라 부른다.

1. 명시된 인입 속도 없이 인입하는 낮은 우선순위 트래픽 전체를 수용하는 플로우(v)를 상정한다. 해당 플로우는 실제 높은 우선순위 플로우들의 인입 속도의 총합과 서버의 capacity와의 차이만큼의 인입 속도를 가진 것처럼 취급된다 ($\rho_v = r - \sum_i \rho_i$).

2. 전체 시스템에 실제 패킷이 저장되어 서비스를 기다리거나 서비스를 받는 기간을 “backlog 기간”이라고 정의한다.

3. Backlog 기간이 시작되는 순간 모든 queue의 deficit counter값을 최대 패킷 크기로 바꾼다.

4. Backlog 기간마다 큐들의 서비스 순서는 다를 수 있으며, backlog 기간 초반에, 실제 패킷이 들어온 순서로 큐의 서비스 순서를 결정한다. 한번 결정된 순서는 backlog 기간 동안 바꾸지 않는다.

5. Backlog 기간에만 가상 패킷을 생성해서 처리한다. 가상 패킷의 크기는 quantum 크기와 같다.

6. 가상 패킷을 서비스하는 동안 해당 큐에 실제 패킷이 도착하면 그 즉시 가상 패킷의 서비스를 멈추고, 실제 패킷의 크기를 deficit값과 비교하여 서비스 여부를 결정한다.

7. 전체 시스템에서 모든 실제 패킷이 서비스되어 사라지면 backlog 기간이 끝난다.

8. 가상 패킷의 서비스가 완료되어도 실제로 링크로 전송하지는 않는다.

이 중 4번, 큐의 서비스 순서를 결정하는 알고리즘은 아래와 같다.

Algorithm: Service order in qnw-DRR

```
const integer N          // Number of queues
integer array ServiceOrder[1..N] <- {0..0}
  // Service order of queues,
ServiceOrder {3..} means 3rd queue's order is 1st.
Upon actual packet arrival
  if (any item in ServiceOrder) not=
packet.queue_index
    for i in 1..N
      if ServiceOrder[i] is Null
        ServiceOrder[i] ← packet.queue_index,
        break,  end if,  end for, end if
Upon (virtual or actual) packet departure
 if still system_backlog
   for j in 1..N-1
     if (any item in ServiceOrder) not=
(packet.queue_index+j)%N
       for i in 1..N
        if ServiceOrder[i] is Null
          ServiceOrder[i] ← (packet.queue_index+j)%N,
       //Place nearest next queue in ServiceOrder list
          break,  end if,  end for,  end if,  end for
  else ServiceOrder[i] <- Null for all i,  end if
```

예를 들어 오랜만에 패킷이 하나 들어오고 더 들어오지 않으면 해당 패킷은 즉시 서비스되고 가상 패킷도 생성하지 않는다.

다른 예로, 비어 있는 시스템에 패킷 세 개가 연속으로 들어오는데 각각 순서대로 큐 3, 2, 7번으로 들어왔다면 서비스 순서는 3-2-7이다.

만약 1번부터 10번까지 열 개의 큐를 가진 시스템에 패킷 네 개가 연속으로 들어오는데 각각 큐 3, 3, 2, 7번으로 들어왔으며, 3번 큐가 한 번의 라운드에 두 개 패킷을 모두 처리하지 못하면, 서비스 순서는 3-2-7-8-9-10-1-4-5-6-3이다. 이때 3, 2, 7을 제외한 큐는 가상 패킷을 만들어 서비스한다.

만약 패킷 세 개가 연속으로 들어오는데 각각 큐 3, 3, 2에서 들어와서 3-2-4-5까지 서비스했는데 7번에서 실제 패킷이 들어오면 바로 7번으로 건너뛴다. 3-2-4-5-7-8-9-10-1-6-3의 순서로 진행한다.

만약 패킷 세 개가 연속으로 들어오는데 각각 큐 3, 3, 2에서 들어와서 3-2-4-5-6-7-8까지 서비스했는데 7번에서 실제 패킷이 들어오면, 7번 큐는 이미 서비스한 것이므로 계속 다음 큐로 진행하여 3-2-4-5-6-7-8-9-10-1-3-2-4-5-6-7의 순서로 진행한다.

이와 같이 동작하는 qnw-DRR은 nw-DRR과 비교해 최대 지연시간은 동일하게 유지하면서, backlog 기

간 초기에 인입하는 소수의 패킷들에 대해서 좀 더 빠르게 서비스를 제공한다. Backlog 기간이 짧을수록 이러한 패킷의 비율이 높으므로 평균 지연시간이 더욱 줄어든다. 따라서 전체 트래픽의 인입 속도가 낮을수록 확률적인 성능의 향상에 효과적이다.

Ⅳ. 성능 비교

본 장에서는 제안하는 qnw-DRR의 성능을 DRR, nw-DRR, TDMA 스케줄러와 비교 평가한다. 모든 스케줄러는 10개의 큐를 가지고 있으며 10~90Mbps 중 특정 속도로 스케줄러로 인입된 패킷은 1/10의 확률로 특정 큐로 분배된다. 패킷의 길이는 500~2000bit이며 이 사이에서 균일하게 임의의 (uniformly random) 값을 가진다. 큐가 저장할 수 있는 패킷의 수는 무한하다. 출력 링크는 100Mbps의 용량을 가지고 있다. TDMA 스케줄러에서는 최대 크기의 2000bit 패킷을 100Mbps로 서비스할 수 있도록 정해진 20μs의 slot 10개로 이루어진 frame이 반복된다. 큐에 대기하는 패킷 들 중 앞에서부터 하나의 slot에 수용 가능한 패킷들을 모두 처리한다. 다음과 같은 파라미터 값을 시뮬레이션에서 사용하였다.

그림 3은 트래픽 인입 속도를 변화시키면서 log scale로 관찰한 네 종류 스케줄러에서의 평균 패킷 지연시간이다. 네 종류 스케줄러 중 DRR을 제외하면 모두 최대 지연시간을 보장할 수 있으며, 따라서 DRR의 우월한 확률적 성능 특성을 나머지 세 스케줄러가 얼마나 따라갈 수 있느냐가 중요하다.

nw-DRR은 TDMA와 DRR의 중간 수준의 평균 지연시간을 보인다. 인입 속도가 낮을수록 TDMA와, 인입 속도가 높을수록 DRR과 비슷한 성능을 보인다. nw-DRR이 트래픽의 인입 속도가 아주 낮은 경우에는 가상패킷을 모든 큐에 항상 만들어야 하는 상황이 만들어지는데 이것은 TDMA가 고정된 slot을 할당하는 것과 비슷하다. 반대로 인입 속도가 아주 높은 경우에는 모든 큐에 실제 패킷이 존재할 가능성이 크므로 가

표 2. 시뮬레이션에서 사용한 파라미터 값
Table 2. Parameter values used in the simulation

Parameter	Value
Max packet length	2000bit
Number of queues	10
Output Link capacity	100Mbps
Input data rate to the system	10M~90Mbps
Quantum value assigned for a queue	400bit

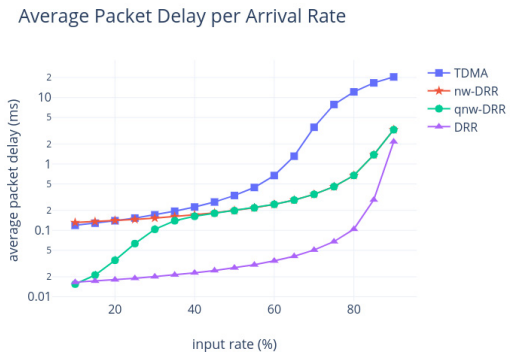


그림 3. 트래픽 인입 속도에 따른 네 종류 스케줄러의 평균 지연시간 변화
Fig. 3. Average packet delay of four schedulers, with varying traffic arrival rate

상패킷을 만들 필요가 거의 없다. 이 상황은 DRR과 유사하다. 본 연구에서 제안하는 qnw-DRR은 인입 속도가 낮은 경우에 DRR과 유사한 성능 특성을 보이다가 인입 속도가 일정 수준 이상일 때는 nw-DRR과 같은 성능을 보인다. 기존의 nw-DRR대비 인입 속도가 낮은 경우 월등한 확률적 성능 특성을 보인다. 인입 속도가 10%인 경우에는 DRR과 거의 같은 성능을 보이다가 40% 이상에서는 nw-DRR과 거의 같은 성능을 보인다. 90%의 아주 높은 인입 속도의 경우 DRR은 2ms, nw-DRR과 qnw-DRR은 3.2ms의 평균 지연시간을 보인다.

그림 4는 같은 환경에서의 평균 큐 길이의 변화를 log scale로 보여준다. 그림 3에서와 마찬가지의 결과를 도출할 수 있다. 트래픽의 인입 속도가 낮은 경우 제안하는 qnw-DRR이 비교 대상인 DRR의 성능에 근

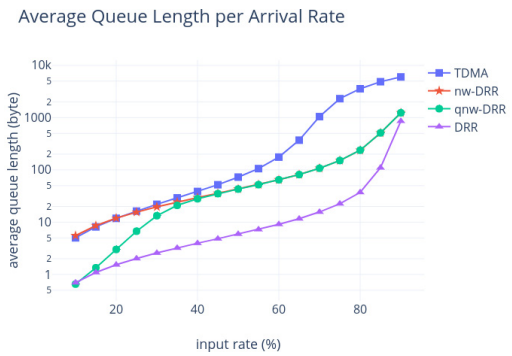


그림 4. 트래픽 인입 속도에 따른 네 종류 스케줄러의 평균 큐길이 변화
Fig. 4. Average queuelength of four schedulers, with varying traffic arrival rate

접하여 가장 우수하다.

위 시뮬레이션에서 트래픽 인입 속도가 큰 경우 시스템의 안정성(stability)을 확인할 필요가 있다.

그림 5와 같이 인입 속도가 출력 링크용량 (Link capacity)의 80%인 경우에도 nw-DRR은 안정적인 지연시간 확률분포를 보인다.

이에 반해 그림 6에서와같이 TDMA는 인입 속도가 출력 링크용량 (Link capacity)의 80%인 경우에 불안정한 (unstable) 모습을 보인다. 이에 비추어 그림 3과 4에서 인입 속도가 60%를 초과하면서 TDMA의 지연시간 증가가 선형적인 특성을 보이는 것은 시스템이 불안정해짐에 따라 정확하지 않은 값이 도출된 것으로 보인다.

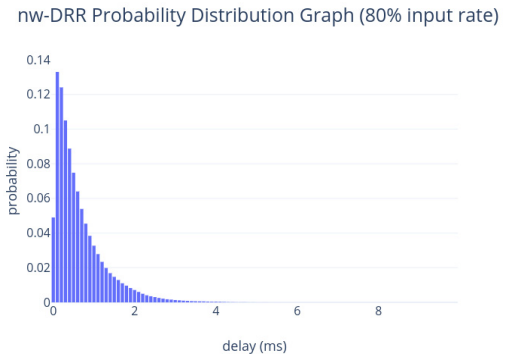


그림 5. 80%의 인입 속도일때 nw-DRR의 지연시간 확률분포
Fig. 5. Probability density distribution of delay of nw-DRR at 80% input rate

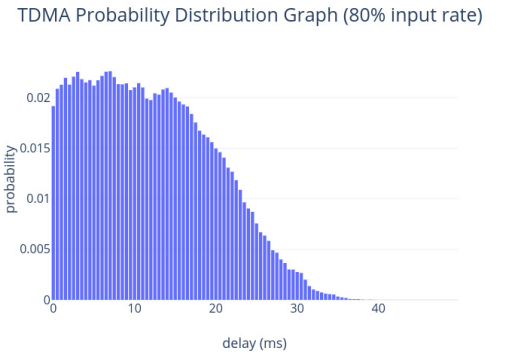


그림 6. 80%의 인입 속도일 때 TDMA의 지연시간 확률분포
Fig. 6. Probability density distribution of delay of TDMA

V. 결 론

기존 연구에서 비작업보존 방식으로 동작하여 스케줄러와 레귤레이터 기능을 동시에 수행하는 레귤레이팅 스케줄러(RSC)가 제안되었다. RSC를 입력포트별로 구분된 큐에 적용하면 수 밀리초 수준의 단대단 지연시간 보장이 가능하다. 하지만 레귤레이터 기능을 포함하는 RSC는 일반적인 작업보존형 스케줄러에 비해 평균지연시간 등의 성능이 떨어진다. 본 연구에서는 작업보장형 스케줄러에 가까운 확률적 성능을 보이면서도 RSC의 최대성능보장 기능을 유지하는 개선된 RSC를 제안하고 이의 성능을 시뮬레이션을 통해 검증하였다.

RSC를 DRR에 적용한 nw-DRR은 TDMA와 DRR의 중간 수준의 평균 지연시간을 보였다. 인입 속도가 낮을수록 TDMA와, 인입 속도가 높을수록 DRR과 비슷한 성능을 보였다. nw-DRR에서 트래픽의 인입 속도가 아주 낮은 경우에는 가상패킷을 모든 큐에 항상 만들어야 하는 상황이 만들어지는데 이것은 TDMA가 고정된 slot을 할당하는 것과 비슷하다. 반대로 인입 속도가 아주 높은 경우에는 모든 큐에 실제 패킷이 존재할 가능성이 크므로 가상패킷을 만들 필요가 거의 없다. 이 상황은 DRR과 유사하다.

본 연구에서 제안하는, 개선된 RSC를 DRR에 적용한 qnw-DRR은 비교 대상인, 최대 지연시간을 보장하지 못하지만 확률적 특성이 뛰어난, 일반적인 DRR과 가장 유사한 성능 특성을 보인다. 인입 속도가 출력 링크 용량의 10%인 경우에는 DRR과 거의 같은 성능을 보이다가 40% 이상의 과부하 상태에서 nw-DRR과 거의 같은 성능을 보인다. 따라서 보통의 네트워크의 동작 환경에서, 제안하는 qnw-DRR은 최대지연시간을 보장하면서도 DRR과 유사한 확률적 성능을 보인다고 할 수 있다.

향 후 RSC가 연속적으로 채택된 네트워크에서의 nw-DRR와 qnw-DRR의 성능을 시뮬레이션을 통해 연구할 예정이다. 앞 단의 RSC에 의해 레귤레이트된 인입 트래픽을 받는 경우 본 연구에서 상정한 임의의 인입 트래픽에 비해 우수한 성능을 보일 것으로 예상한다.

References

[1] *IEEE 802.1 Time-Sensitive Networking Task Group Home Page*, Retrieved December, 14, 2018 from http://www.ieee802.org/1/pages/tsn.html

[2] *Residential Ethernet Tutorial*, Retrieved December, 14, 2018 fromwww.ieee802.org/802_tutorials/05-March/tutorial_1_0305.pdf

[3] J. Joung, "Regulating scheduler (RSC): A novel solution for IEEE 802.1 time sensitive network (TSN)," *Electronics J.*, MDPI, Jan. 2019.

[4] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 5, Oct. 1998.

[5] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," *IEEE/ACM Trans. Netw.*, vol. 4, no. 3, pp. 375-385, Jun. 1996.

[6] L. Lenzini, E. Mingozzi, and G. Stea, "Tradeoffs between low complexity, low latency, and fairness with deficit round-robin schedulers," *IEEE/ACM Trans. Netw.*, vol. 12, no. 4, Aug. 2004.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "*An architecture for differentiated service*," RFC 2475, 1998.

[8] H. Zhang and D. Ferrari, "Rate-controlled service disciplines," *J. High Speed Netw.*, 1994.

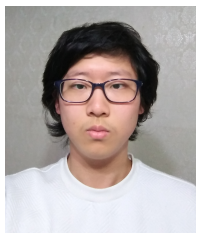
[9] L. Georgiadis, R. Guerin, V. Peris, and K. N. Sivarajan, "The effect of traffic shaping in efficiently providing end-to-end performance guarantees," *Telecommun. Syst.*, vol. 5, no. 1, pp. 71-83, Mar. 1996.

[10] 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," *30th ITC*, 2018.

정 진 우 (Jinoo Joung)

1992년 2월 : KAIST 전자공학과 졸업
1994년 8월 : NYU 전기전자공학과 Master
1997년 8월 : NYU 전기전자공학과 Ph.D.
1997년 10월~2005년 2월 : 삼성전자 종합기술원
2005년 2월~현재 : 상명대학교 휴먼지능정보공학과 교수
<관심분야> 유무선통신, 네트워크, 임베디드 시스템

권 주 혁 (Juhyeok Kwon)



2017년 3월~현재 : 상명대학교 휴먼지능정보공학과 재학 중
<관심분야> 유무선통신, 네트워크, 임베디드 시스템