네트워크 자원 가격정책을 위한 사용자 유틸리티 함수 추정법

(Estimating User Utility Functions for Network-Resource Pricing)

박 선 주 †

(Sunju Park)

요 약 인터넷 트래픽 관리를 위한 IETF 차별화 서비스에서는 우선순위 방식의 가격정책이 널리 채택되고 있으며, 이러한 우선순위 방식의 서비스에서 최적가격을 정하기 위해 균형분석을 이용한 연구가 활발히 진행되고 있다. 그러나 '사용자들의 유틸리티 함수를 정확히 알고 있다'는 균형분석의 가정의 타당성에 대한 비판 또한 끊이지 않는다. 따라서 이 논문은 최적가격을 정하는 문제에 있어서, 현존하는 이론적인 연구의 기본 가정과 사용자 유틸리티 함수를 알 수 없는 현실적인 네트워크 환경과의 차이를 좁힐 수 있는 해결책을 제시하고자 한다. 기본 아이디어는 네트워크 서비스 제공자가 서비스 레벨의 가격을 조정하여 사용자들의 레벨 선택 결정을 바꾸도록 유도하고 그를 통해 더 정확한 유틸리티 정보를 알아내도록 하자는 것이다. 이 연구의 공헌은 크게 두 가지로 볼 수 있다. 첫째, 사용자 유틸리티 함수를 추정해나가는 일반적인 원리를 제시하였다. 둘째, 유틸리티 함수에 대한 정보를 최대한으로 끌어낼 수 있도록 하는 가격을 정하는 방법을 개발하였다. 우리는 실험을 통해 제안된 추정법의 효과적인 성능을 보여준다.

키워드 : 유틸리티 추정, 네트워크 서비스 가격정책, 우선순위 방식 서비스 네트워크, 학습

Abstract Priority-based network service has been widely adopted for the Internet traffic management in the context of IETF differentiated services, and computing optimal prices for such priority-based service is the key topic in many pricing literature. While the equilibrium analysis has been commonly used to this end, many have criticized the validity of the underlying assumption of equilibrium analysis that user utility functions are precisely known. In this paper, we propose a solution for bridging the gap between the existing theoretical work on optimal pricing and the unavailability of precise user utility information in real networks. In the proposed method, the service provider obtains more and more accurate estimates of user utility functions from the initial imprecise knowledge by iteratively changing the price of service levels and observing the users' decisions under the changed price. Our contribution is two-fold. First, we have developed a general principle for estimating the user utility functions. Second, we have developed a novel method for setting the prices that can optimize the extraction of the knowledge about user utility functions. The extensive simulation results demonstrate the effectiveness of our method.

Key words : utility estimation, network service pricing, priority service networks, learning

1. 서 론

현재의 인터넷 서비스 제공자들은 대부분 네트워크 링크의 용량을 기준으로 한 일률가격정책을 쓰고 있다. 예를 들어, T3 링크는 T1 링크보다 비싸고, VDSL 링크는 ADSL 링크보다 비싸다. 즉, 네트워크 자원의 실제 사용량이 아닌 자원의 최대 사용 가능량에 의해 가격이 정해진다. 한편, 이와는 다른 방식의 가격정책들이 사용되는 영역들도 있다. 예를 들어, Boingo 등등과 같은 무선 LAN 핫스팟 (hotspot) 제공자들은 접속시간별로 사용자에게 돈을 받거나, 일정한 접속시간을 예상하여 정해진 플랜을 사용자들에게 팔기도 한다. Verizon이나 Sprint 등등의 셀방식의(cellular) 데이타 서비스 제공자들은 데이타 전송량에 한계를 둔 (예를 들어, 한 달에 20 Mbytes 등의) 플랜을 제공하고 있다.

† 종신회원 : 연세대학교 경영학과 교수
boxenju@yonsei.ac.kr
논문접수 : 2005년 5월 16일
심사완료 : 2005년 10월 18일

이러한 무선 인터넷 액세스나 셀방식 데이타 서비스를 위한 가격정책들은 네트워크 자원이 부족하거나 비싼 환경에 적합한 혼잡가격정책(congestion pricing) 또는 한계가격정책(marginal-cost pricing)의 초기 단계로 볼 수 있다. (1) 대역폭을 많이 요구하는 멀티미디어 서비스가 빠른 추세로 증가하고 있고, (2) 인터넷 사용의 bursty한 속성 때문에, 유선 네트워크에서도 혼잡도 조절과 QoS를 제공하기 위해서는 종량제 가격정책(usage-based pricing) 에 대한 시도가 계속될 전망이다.

가격정책은 이러한 문제들을 푸는 방법으로 사용될 수 있다. 사용자들은 가격에 민감하여, 가격이 사용자원의 가치를 정당화할 수 있을 때에만 값을 지불하고 자원을 소비한다. 따라서 적절한 가격정책은 사용자의 행동양식에 영향을 미쳐 결과적으로 네트워크 전체의 경제적 효율성을 이룰 수 있다. 네트워크 서비스 가격정책에 대해서는 이제까지 많은 연구가 이루어져 왔다[1-16]. 이 논문에서는 현재 인터넷 QoS 아키텍처로 선호되고 있는 IETF 차별화 서비스[1]에 가장 적합한 우선순위 가격정책 (priority pricing)을 그 대상으로 잡고 있다.

우선순위 가격정책하에서 서비스 제공자는 사용자에게 {서비스 레벨, 가격}의 쌍들을 제공하고, 자신의 트래픽의 값어치를 가장 잘 알고 있는 사용자들이 그 트래픽을 수행하기 위해 필요한 서비스 레벨과 가격을 선택한다. 따라서 우선순위 가격정책에 관한 연구에서는 각 서비스 레벨의 가격을 최적화하는 것이 주 관심사이다. 하지만 현존하는 우선순위 서비스를 위한 최적 가격정책은 균형분석을 통해 경쟁균형(competitive equilibrium)을 계산하는 것에 의존하며, 이를 위해 사용자들의 유틸리티 함수를 정확하게 알고 있다고 가정한다. Shenker는 전통적인 경제환경에서는 사용자들의 유틸리티들이 고정되어 있으므로 오랜 시간을 거친 관찰을 통해 알아낼 수 있으나, 네트워크 환경에서는 사용자의 유틸리티가 네트워크의 상태에 따라 짧은 시간내에 변할 수 있기 때문에 장기적인 수요함수로 표현할 수 없다고 주장한다[17]. 즉, 그는 최적가격에 대한 현존하는 연구들에 공통되는 "사용자들의 유틸리티 함수에 대한 정확한 정보를 가지고 있다."라는 기본 가정의 타당성을 비판하고 있다.

Shenker의 주장에 더해, 우리는 유틸리티 함수의 부정확성은 사용자들에게 자신의 유틸리티 함수를 네트워크에 보고하도록 함으로써 해결되지 않는다고 주장한다. 이는 다음과 같은 이유 때문이다. 첫째, 현존하는 인터넷 구조와 프로토콜은 사용자들이 그들의 유틸리티 정보를 가격을 책정하는 쪽에 알려줄 수 있는 표준화된 방법을 제공하고 있지 않다. 둘째, 만일 그러한 방법이 존재한다고 해도 사용자들의 유틸리티가 빠른 시간내에 변하므로 유틸리티 정보를 자주 보고하는 것 자체가 너무 비싼 (비효율적인) 방법이 될 수 있다. 셋째, 사용자들은 자신의 기호도를 솔직하게 보고할 인센티브가 없을 수도 있다. 따라서 보고된 유틸리티 값은 실제의 값과 다를 수 있다.

이 논문에서 우리는 현존하는 최적가격정책에 관한 이론적인 연구의 기본 가정과 현실과의 차이를 메울 수 있는 해법을 제공하고자 한다. 기본 아이디어는 불확실한 초기 유틸리티 정보 하에서 네트워크가 사용자에게 유틸리티 정보를 요구하지 않고서도 정확한 유틸리티 함수를 효율적으로 추정해나가도록 하자는 것이다. 초기의 네트워크가 사용자 유틸리티 함수에 대한 부정확한 정보를 가지고 있다고 가정해 보자. (이러한 불확실한 초기 정보는 사용자들의 수요와 네트워크 서비스 제공을 관찰함으로써 얻어질 수 있다.) 유틸리티 추정은, 사용자 유틸리티 함수에 대한 정보를 얻어내기 위해 고의적으로 선택된 현재의 가격에 대한 사용자들의 반응을 지켜봄으로써 얻어진다. 즉, 네트워크는 각 라운드마다 다른 가격을 제시하며, 여러 라운드를 통한 학습으로 사용자 유틸리티 함수에 대한 추정을 점차적으로 개량해 나간다. 라운드의 횟수와 가격조정의 간격은 사용자 유틸리티가 얼마나 빨리 변하는지와 추정값이 얼마나 정확해야 하는지에 의해 결정된다. 이때 추정값의 정확성의 정도는 유틸리티 함수에 대한 추정이 끝난 후에 사용될 가격책정 방식이 유틸리티 정보의 정확도에 얼마나 민감하게 영향을 받는지에 따라 정한다. 한번 정해진 가격이 계속 쓰이지 않고 실시간에 가격이 계속 변하는 다이내믹 가격책정방식은 이전에도 제안된 적이 있다[8]. 가격변화의 현실적인 어려움(예를 들어, 가격조정 정보교환을 위한 트래픽의 증가로 인한 오버헤드 등)과 이에 대한 보완책은 이 논문이 다루는 범위 밖의 문제이다.

이 연구는 크게 두 가지의 공헌을 한다. 첫째, 주어진 가격에 대한 사용자의 의사결정을 관찰하여 불확실한 사용자 유틸리티 함수를 추정해 나가는 일반적인 방법을 개발하였다. 둘째, 유틸리티 함수에 관한 정보를 최대한으로 끌어낼 수 있는 다이내믹 가격책정 방법을 개발하였다.

이 논문은 다음과 같이 구성되어있다. 2장은 이 논문에서 사용하는 우선순위 서비스 모델을 소개한다. 3장에서는 사용자들의 유틸리티 함수를 추정하는 방법을 설명한다. 4장에서는 제안된 유틸리티 추정방법을 실험을

1) IETF 차별화 서비스는 사용자들이 각 패킷의 중요도를 IP 패킷 헤더에 트래픽 클래스 정보로 제공하도록 하며, 네트워크는 우선순위방식의 스케줄링을 통해 패킷을 처리한다.

통해 분석한다. 5장에서 결론을 맺는다.

2. 우선순위 네트워크 서비스 모델

본 논문이 대상으로 하는 우선순위 방식 가격정책의 네트워크 서비스 모델은 다음과 같다. 네트워크가 사용자들에게 I개의 서비스 레벨을 제공하며, 각 서비스 레벨 i를 일정 데이타 단위만큼 (예를 들어 패킷) 사용하는 금액 P_i를 요금으로 청구한다. I개의 가격 벡터는 $\{P_i\}$로 정의한다. 혼동을 피하기 위해, 가장 좋은 (빠른) 서비스 레벨을 레벨 1, 두번째로 좋은 서비스 레벨을 레벨 2, 등등으로 정의한다. 네트워크 서비스 제공자는 이득을 최대화한다든지 또는 전체 시스템의 유용성을 향상시킨다든지 하는 주어진 목적을 만족시키는 가격벡터를 구하는 것을 목표로 삼는다.

N명의 사용자들이 네트워크상에 존재하고, 이들이 트래픽들을 만들어낸다고 가정하자. 예를 들어 이메일 메시지를 보내는 것이나 웹페이지를 액세스하는 것 등등이 하나의 트래픽이라고 볼 수 있다. 이러한 트래픽을 '일' (job)이라고 부르며, J 종류의 일들이 있다고 가정한다. 사용자와 일 사이의 불필요한 혼동을 피하기 위해 이 논문에서는 각 사용자들이 한 종류의 한 개의 일을 가지고 있다고 가정한다. (이러한 가정은 기호사용의 편리를 위한 것일 뿐이며, 각 사용자들이 반드시 하나의 한 종류의 일을 가져야 한다는 것은 아니다. 즉, 일반적인 모델에서 한 명의 사용자는 여러 개의 다른 종류의 일들을 가질 수 있다.)

사용자는 자신의 일에 적합한 서비스 레벨을 선택하고 그에 해당하는 가격을 지불한다. 사용자는 일을 전송하는 것 자체를 포기할 수는 있으나, 서비스 레벨에 대한 자신의 결정을 번복할 수는 없다. 일의 종료지연시간을 t라 할 때 그 일의 유틸리티 함수를 $U_j(t)$로 정의한다. 이때 $U_j(T)$는 t에 따른 단조감소 함수이다.

서비스 레벨 i에서의 일도착률을 λ_i라 하면 $\{\lambda_i\}$는 I 차원의 일도착률 벡터를 이룬다. 이때 λ_0는 시스템에 도착하지 않은 일들의 도착률을 나타낸다. 즉, λ_0는 사용자가 전송을 포기한 트래픽들을 표현한다. 총가능 일도착률(total *potential* job-arrival rate)은 λ^* ($\lambda^* = \sum_{i=0}^{I} \lambda_i$)로 표현하고, 유효 일도착률(*effective* job-arrival rate)은 $\lambda(\lambda = \lambda^* - \lambda_0)$로 표현한다. 네트워크의 일처리율은 μ로 표현하며, 유효 일도착률 λ는 항상 μ보다 작다.

네트워크 서비스는Non-Preemptive Strict Priority Scheduling (NP-SPS) 방식을 사용한다. SPS란 높은 레벨의 서비스를 기다리는 일들이 없는 경우에만 낮은 레벨의 일들을 처리하는 것을 말한다. 즉, 낮은 레벨의 일은 높은 레벨의 일을 다 처리한 후에야 시작된다. 같은 레벨의 일들은 FIFO 방식으로 처리된다. NP는 새롭게 도착하는 높은 레벨의 일이 현재 처리되고 있는 낮은 레벨의 일을 중단시키지 않는 것을 의미한다.

3. 사용자들의 유틸리티 함수 추정

이장에서는 사용자들의 유틸리티 함수에 대한 정보를 알아내는 방법을 소개한다. 유틸리티 추정의 시작단계에서 네트워크는 유틸리티 함수에 대한 약간의 초기지식을 가지고 있을 수 있다. 우리는 유틸리티 함수의 모형과 매개변수(parameter)들이 가질 수 있는 값의 범위가 상한값과 하한값의 형태로 알려져 있다고 가정한다. 따라서 추정법의 목표는 최대한으로 매개변수들의 불확실성을 줄여나가는 것 (즉 상한값과 하한값의 간격을 최소화하는 것)이라고 말할 수 있다.

네트워크는 주어진 가격하에서 사용자들이 어떤 서비스 레벨을 선택하는지를 관찰함으로써 사용자 유틸리티 함수를 추정해 나간다. 이러한 추정은 다음과 같은 세 단계를 반복적으로 사용하여 이루어진다. 첫째, 네트워크는 가격벡터 $\{P_i\}^1$ $(1 \le i \le I)$를 책정한다. 둘째, 사용자들은 현재 책정된 가격에서 자신의 일들에 가장 적합한 서비스 레벨을 선택한다. 사용자의 결정은 사용자 의사결정벡터 (*user decision vector*) $\{L_n\}^1$ $(1 \le n \le N)$로 표현하며, 이때 L_n은 n번째 사용자가 선택하는 서비스 레벨을 말한다. 셋째, 네트워크는 사용자의 선택을 바탕으로 유틸리티 추정벡터(*utility estimation vector*) $\{U_n\}^1$ $(1 \le n \le N)$를 변경한다. 이러한 세 단계 사이클이 끝나게 되면 네트워크는 보다 자세한 유틸리티 정보를 알게되며, 이 정보를 이용하여 다음 번 반복 사이클에서 사용할 새로운 가격벡터 $\{P_i\}^2$를 계산한다. 이러한 세 단계 추정 사이클은 (1) 미리 정한 신뢰수준(confidence level)내로 유틸리티 함수를 알아낼 때까지 (즉 매개변수들의 추정범위가 일정 이하로 작아졌을 때), 또는 (2) 미리 정해놓은 시간 제한에 도달했을 때까지 계속된다. 각 반복 사이의 실제 간격은 네트워크가 얼마나 자주 가격벡터를 변경하고 싶어하는지에 따라 결정된다.

그림 1은 세 단계 반복을 보여준다. 여기에서 $\{U_n\}^0$는 초기 유틸리티 추정치를, $\{U_n\}^F$는 최후의 유틸리티 추정치를 말한다. 첫째, 사용자들은 주어진 가격벡터 하에서 자신의 일에 적합한 서비스 레벨을 선택한다 ($\{P_i\} \rightarrow \{L_n\}$). 둘째, 네트워크는 사용자의 결정을 바탕으로 유틸리티 함수에 대한 추정치를 변경한다 ($\{L_n\} \rightarrow \{U_n\}$). 셋째, 네트워크는 새로운 유틸리티 함수 추정치를 이용하여 새 가격벡터를 구한다 ($\{U_n\} \rightarrow \{P_i\}$).

아래에서 우리는 추정법의 각 단계를 자세히 설명한

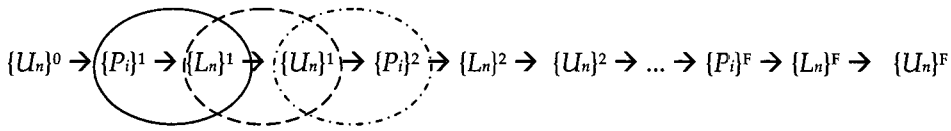

그림 1 사용자 유틸리티 함수 추정을 위한 반복 과정

다. 다시 한 번 강조하지만, 모든 추정작업이 끝나고 $\{U_n\}^F$이 구해진 후에는, 최적 가격벡터를 구하기 위해 현존하는 어떤 가격정책을 쓰더라도 상관이 없다. 즉, 사용자 유틸리티 함수를 추정하는 과정은 얻어진 $\{U_n\}^F$를 사용하는 과정과는 무관하다.

3.1 사용자의 의사결정 모델링: $\{P_i\} \rightarrow \{L_n\}$

사용자들이 이기적이며(self-interested), 비전략적이고(non-strategic), 이성적이라고(rational) 가정한다면 사용자들의 의사결정은 다음과 같이 모델할 수 있다. j-타입의 일을 가진 사용자는 자신의 유틸리티 값에서 원가를 뺀 값을 최대화하는 서비스 레벨 i를 선택할 것이다. 즉 j-타입 일을 가진 사용자의 의사결정은 다음의 식으로 표현할 수 있다.

$$Argmax_i\ [U_j(T_{ij}) - P_i \times C_j]. \quad (1)$$

이때 $U_j(\cdot)$는 사용자 j의 유틸리티 함수를, P_i 는 i-레벨의 가격을, C_j 는 j-타입 트래픽의 평균 크기를, T_{ij} 는 j-타입의 일이 i-레벨에서 서비스가 종료될 때까지 걸리는 기대시간을 말한다. 만약 $[U_j(T_{ij}) - P_i \times C_j]$가 모든 i $(1 \le i \le I)$에 대해 음수이면, 그 일은 시스템에 제출되지 않는다. NP-SPS 스케쥴링 방식에서는 종료시간을 보장할 수 없으므로 "기대" 종료시간을 쓴다.

예를 들어 각 서비스 레벨의 일도착률이 포아송 분포를 따르고 각 일들의 크기가 지수분포를 가진다고 가정하면, j-타입의 일이 k-레벨에서 종료까지 걸리는 기대시간 T_{kj} 는 (2)의 식과 같이 $M/M/1$ 우선순위 대기행렬 모델로 구할 수 있다[18]. 이때 C는 모든 일들의 평균 크기를 나타낸다.

$$T_{kj} = \frac{C_j}{\mu} + \frac{C^2/\mu^2 \cdot \sum_{i=1}^{I} \lambda_i}{(1 - C/\mu \cdot \sum_{i=1}^{k-1} \lambda_i)(1 - C/\mu \cdot \sum_{i=1}^{k} \lambda_i)} \quad (2)$$

3.2 사용자의 선택을 바탕으로 한 유틸리티 추정값 변경: $\{L_n\} \rightarrow \{U_n\}$

네트워크가 각 사용자의 의사결정으로부터 유틸리티 함수에 대한 정보를 얻어내는 방법은 다음과 같다. 사용자 j가 일을 전송하는 것을 포기했다면, 그 결정은 $U_j(\cdot)$에 관한 다음과 같은 정보를 드러낸다.

$$U_j(T_{ij}) - P_i \times C_j \le 0 \ for \ \forall i \ (1 \le i \le I). \quad (3)$$

만일 사용자 j가 레벨 L_k $(k \neq 0)$를 선택했다면 $U_j(\cdot)$에 대해 다음과 같은 정보를 알 수 있다.

$$U_j(T_{kj}) - P_k \times C_j > 0, \quad (4)$$

$$U_j(T_{ij}) - P_i \times C_j < U_j(T_{kj}) - P_k \times C_j \ for \ \forall i (i \neq k). \quad (5)$$

위의 정보를 이용하면 네트워크는 각 사용자들의 유틸리티 함수의 추정치의 범위를 줄일 수 있다. 앞에서 언급한 바와 같이, 우리는 유틸리티 함수의 모양은 알려져 있고 각 함수의 매개변수의 값들이 알려져 있지 않다고 가정하고 있다. 따라서 (3), (4), (5)의 식을 유틸리티 함수 모형에 적용함으로써, 네트워크는 각 매개변수의 상한값과 하한값의 범위를 줄일 수 있다.

이러한 과정을 보다 구체적으로 설명하기 위해 우리는 현존하는 유틸리티 함수 모형중의 하나를 사용한다[2-3]. 물론 이 함수모형은 설명의 목적으로 사용된 것일 뿐으로, 우리의 추정법은 이 특별한 함수 모형에만 적용이 국한된 것은 아니다.

Mendelson은 오목(strictly concave)하고, 두번 미분가능한(twice-differentiable) 가치함수(value function) $V_j(\lambda_j)$를 가정한다. 가치함수 $V_j(\lambda_j)$는 모든 j-타입 일들의 단위시간당 총 기대가치를 나타낸다. 여기서 λ_j는 j-타입 일들의 총도착률을 표현하고 있다. (λ_j는 i-레벨의 일도착률인 λ_i와 다르다는 것에 주의하자.) 한계가치함수 (marginal value function), $V_j'(\lambda_j)$, 는 하나의 j-타입 일의 가치를 나타낸다. 이 논문은 참고문헌에서처럼 일의 종류에 따라 일정한 상수 K_j를 가진 등탄력성 한계가치함수 (iso-elastic marginal value function) $V'(\lambda^*) = K_j / \sqrt{\lambda^*}$ 를 사용한다[3]. 이때 유틸리티 함수 모형은 일의 종료지연에 따라 일의 가치가 선형으로 일정하게 감소하는 경우를 가정한다. 지연비용(delay cost)이라 불리는 1차 선형 유틸리티 함수의 기울기는 각 일의 타입에 따라 다르며, 사용자 j의 지연비용은 α_j로 표현한다(물론 α_j는 음수이다). 다시 말해 사용자 j의 유틸리티 함수는 다음과 같다.

$$U_j(T_{ij}) = V_j'(\lambda_j) + \alpha_j \times T_{ij} = K_j / \sqrt{\lambda} + \alpha_j \times T_{ij}. \quad (6)$$

식 (6)에서 보여진 것처럼 각 사용자 j의 유틸리티 함수는 α_j와 K_j라는 두 매개변수를 가지고 있다. 따라서 우리의 문제는 이들의 값이 알려져 있지 않을 때 어떻게 유틸리티 함수를 추정해나가느냐 하는 것이다.

우선 $V_j'(\lambda_j)$을 알고 있다고(즉, K_j를 알고 있다고) 가정을 하고 α_j만을 추정해보자. 만약 사용자 j가 일을 전

송하기를 포기했다면, 식 (3)과 (6)을 이용하여 α_j의 상한값에 대한 다음과 같은 조건을 구할 수 있다.

$$\alpha_j \le \{P_i \times C_j - V_j'(\lambda_j)\} / T_{ij} \text{ for } \forall\ i(1 \le i \le I).$$

만약 사용자 j가 L_k $(0<k<I)$를 선택했다면, 식 (4), (5), (6)을 이용하여 다음과 같은 조건들을 얻어낼 수 있다.

$$\alpha_j > \{P_k \times C_j - V_j'(\lambda_j)\} / T_{kj},$$

$$\alpha_j > \{(P_k - P_i) \times C_j\} / (T_{kj} - T_{ij}) \text{ for } \forall i(i<k),$$

$$\alpha_j < \{(P_i - P_k) \times C_j\} / (T_{ij} - T_{kj}) \text{ for } \forall i(i>k).$$

식 (5)는 부등식을 푸는 과정에서 $i < k$일 때는 하한값의 조건을, $i > k$일 때는 상한값의 조건을 제공한다. 만약 사용자 j가 가장 낮은 서비스 레벨 L_I를 선택했다면 식 (4), (5), (6)을 이용하여 다음과 같은 조건들을 얻어낼 수 있다. 여기서 식 (5)는 오직 하한값의 조건만을 제공한다.

$$\alpha_j > \{P_I \times C_j - V_j'(\lambda_j)\} / T_{Ij},$$

$$\alpha_j > \{(P_I - P_i) \times C_j\} / (T_{Ij} - T_{ij}) \text{ for } \forall\ i(i < I).$$

한편, 지연비용 α_j는 알려져 있고 K_j를 모르는 경우를 고려해 보자. 만약 사용자 j가 일의 전송을 포기했다면, 다음과 같은 조건을 구할 수 있다.

$$K_j \le \{P_i \times C_j - \alpha_j \times T_{ij}\} \times \sqrt{\lambda_j} \text{ for } \forall\ i(1 \le i \le I).$$

만약 사용자 j가 L_i $(1 \le i \le I)$를 선택했다면, 식 (4)와 (6)을 이용하여 다음과 같은 K_j의 하한값의 조건을 구할 수 있다.

$$K_j > \{P_i \times C_j - \alpha_j \times T_{ij}\} \times \sqrt{\lambda_j}.$$

이때 식 (5)에 식 (6)을 적용하면 취소가 되어 더 이상의 압축된 범위를 생성하지 못한다.

한편, α_j와 K_j가 모두 알려지지 않는 경우에는 두 매개변수를 동시에 추정해 나간다. 네트워크는 하나의 매개변수의 값을 추정할때 다른 변수의 현재 알려진 추정치를 사용한다. 이때 다른 변수의 추정치가 잘못되어 나머지 하나의 값을 잘못 구하는 경우를 막기 위해, 추정과정 도중 각 매개변수의 추정치를 번갈아 초기값으로 리셋(reset)한다.

3.3 유틸리티 추정치를 이용한 가격벡터 계산: $\{U_n\} \rightarrow \{P_i\}$

추정법의 세 번째 단계는 현 유틸리티 함수 추정치를 이용하여 새로운 가격벡터를 구하는 것이다. 현 단계에서 사용자 유틸리티 함수의 매개변수들은 상한값과 하한값의 범위의 형태로 알려져 있다. 기존의 최적가격선정 방법들은 (부정확한 범위가 아닌) 정확한 유틸리티 함수를 필요로 하기 때문에, 만일 기존 방법들을 이용하려면 현재까지 알려진 유틸리티 함수 추정범위의 중앙치나 평균값을 선택하여 사용하여야 한다. 그러나 이러한 방법은 선택한 값들과 실제값과의 차이 때문에 sub-optimal한 가격을 선택하게 만들 수 있고, 특히 추정범위가 넓을 수 밖에 없는 초기 단계에는 더욱 그러할 것이다. 또한 기존의 가격책정 방법들은 유틸리티 정보를 얻어내는 것이 목적이 아니므로, 사용자 유틸리티 함수에 대한 지식을 향상시키지 못하거나 아주 느리게 향상시킬 수도 있다.

따라서 기존의 최적가격선정 방식을 사용하는 대신, 우리는 추정법을 위한 새로운 가격선정 방식을 개발하였다. 추정단계에서는 가격책정의 목표가 "학습"(즉 사용자 유틸리티 함수에 대한 정보를 알아내는 것)에 있으므로, 시스템 목표(예를 들어 전체 시스템 효율을 올리는 것)를 희생한다. 즉, 추정법에서 사용하는 가격선정 방법은 정확한 사용자 유틸리티 함수의 정보를 얻는 학습의 이득과 추정과정에서의 sub-optimality로 인한 오버헤드를 trade-off한다(정확한 사용자 유틸리티 함수가 알려져 있지 않은 추정단계에서는, 어차피 sub-optimality를 피할 수 없는 것이 사실이다).

무작위로 가격벡터를 정하는 것이 하나의 방법일 수 있다. 하지만 무작위로 선택된 가격벡터가 어떤 사용자들의 의사결정을 바꾸는데 성공한다 하더라도, 유틸리티 함수의 부정확성을 줄여나가는 체계적인 방법이 부족하므로, 무작위 가격선정은 좋은 방법이라고 볼 수 없다. 따라서 우리는 보다 효과적인 학습을 위해, 현재의 가격벡터에 대한 사용자의 반응을 이용하여 새 가격벡터를 정하는 "점진학습"(progressive learning) 방법을 개발하였다.

3.3.1 점진학습

점진학습의 기본 아이디어는 사용자중 일부를 목표로 삼아 그들의 유틸리티 함수에 대한 정보를 더 자세히 알아낼 수 있도록 하는 가격벡터를 설정하자는데 있다. 그리고 이때 목표로 삼는 사용자 그룹과 가격벡터를 추정 사이클의 낭비를 막도록 체계적으로 정하는 방법을 제공한다.

초기 가격벡터가 어떤 간단한 방법을 통해(예를 들어 무작위로) 정해져 있다고 가정하자. 주어진 가격벡터에서 어떤 서비스 레벨을 선택했는지에 따라 최대 I개까지의 그룹으로 사용자들을 나눌 수 있다. 즉, 같은 서비스 레벨을 선택한 사용자들은 같은 그룹에 속하며, 같은 그룹에 속한 사용자들의 유틸리티 함수의 추정범위는 모두 같다. 다음 단계에서 네트워크는 하나의 사용자 그룹을 선택하여 그 그룹내의 일부 또는 전체의 사용자들의 결정을 바꾸도록 하는 새로운 가격벡터를 구한다. 이때 사용자들의 바뀐 결정은 그들의 유틸리티 함수에 대한 더 정확한 정보를 알려준다.

점진학습에서 새 가격벡터를 정할 때에는 오직 하나의 서비스 레벨의 가격만을 바꾼다(여러 레벨의 가격을

동시에 바꾸는 연구는 현재 진행중이다). 레벨 i에서 사용자 j가 얻는 이득 $U_j(T_{ij})$ - $P_i \times C_j$을 $Profit(i,j)$로 정의하고, 레벨 i를 선택한 사용자 그룹을 G_i라고 정의하자. 네트워크는 P_i를 올리거나 내리는 두 가지의 선택이 있다. 사용자는 레벨 i에서의 이득이 다른 어느 레벨의 이득보다 높을 때에만 레벨 i를 선택한다는 것을 상기해 보면 다음이 성립한다. 첫째, P_i를 올리는 것은 $Profit(i,j)$를 낮추므로 그룹 G_i에 속한 사용자가 다른 레벨을 더 선호하게 만들 수 있다. 따라서 G_i에 속한 사용자의 결정을 바꾸기 위해서는 $P_k(k \neq i)$를 고정한 상태에서 P_i를 증가시켜야 한다. 이때 $P_k(k \neq i)$를 고정하는 이유는 P_k를 변화시키면 P_i 증가로 인한 효과를 상쇄할 수도 있기 때문이다. 둘째, 이와는 반대로, P_i를 내리는 것은 $Profit(i,j)$를 높이므로 레벨 i를 더욱 선호하게 만들고, 따라서 그룹 G_i에 속한 사용자의 결정을 바꾸지 못한다. 하지만 P_i를 내리는 것은 다른 레벨을 선택했던 사용자의 결정을 바꾸어 레벨 i를 선택하도록 할 수 있다. 즉, 사용자를 G_i로 들어오게 하기 위해서는 $P_k(k \neq i)$를 고정한 상태에서 P_i를 감소시켜야 한다.

먼저, 첫째의 경우를 살펴보도록 하자. 그룹 G_i에 속한 사용자 j의 결정을 바꾸기 위해서는 다음의 두 조건을 만족하는 레벨 i의 가격(^jP_i)을 구한다.

조건 (A-1): $P_{i-1} > {}^jP_i > P_{i+1}$,

조건 (A-2): $Profit(i,j) <$ Max [$Profit(k,j)$] for user j in group G_i, for any level $k \neq i$, where $Profit(i,j) = U_j(T_{ij}) - {}^jP_i \times C_j$ and $Profit(k,j) = U_j(T_{kj}) - P_k \times C_j$.

조건 (A-1)은 새로 정해진 ^jP_i가 허용 가능한 범위 안에 있는 것을 말한다. 한편, 조건 (A-2)는 ^jP_i가 G_i에 속한 사용자 j의 서비스 레벨 결정을 바꾸도록 설정하는 조건을 제시한다.

원래의 P_i값에서는, G_i에 속한 모든 사용자 j의 $Profit(i,j)$가 가장 높은 값을 가진다. 이때 P_i를 증가시키면 $Profit(k,j)$ $(k \neq i)$는 그대로 있는 채 $Profit(i,j)$가 감소한다. 따라서 각 사용자 j에 대해 조건 (A-2)를 만족시키는 ^jP_i를 구할 수 있다. (사용자 j가 레벨 i가 아닌 다른 레벨을 선택하도록 할 만큼만 ^jP_i를 증가시키면 된다.) 위의 두 조건을 모두 만족시키는 가격이 존재하지 않을 경우에는 무작위 가격 책정을 택한다.

두번째의 경우로 P_i의 값을 내리는 경우를 살펴보자. P_i의 값을 내린다면 G_i내의 사용자들은 결정을 바꾸지 않는다 해도 원래는 레벨 i를 선택하지 않았던 사용자들 중 일부가 레벨 i를 선택하게 만들 수 있다. 그룹 G_i에 속하지 않은 사용자 j의 결정을 바꾸기 위해서는 다음의 두 조건을 만족하는 ^jP_i를 구한다.

조건 (B-1): $P_{i-1} > {}^jP_i > P_{i+1}$,

조건 (B-2): $Profit(i,j) >$ Max [$Profit(k,j)$] for user j in G_k, $k \neq i$, where $Profit(i,j) = U_j(T_{ij}) - {}^jP_i \times C_j$ and $Profit(k,j) = U_j(T_{kj}) - P_k \times C_j$.

조건 (B-1)은 새로 정해진 ^jP_i가 허용 가능한 범위 안에 있도록 한다. 조건 (B-2)는 ^jP_i가 G_k에 속한 사용자 j의 서비스 레벨 결정을 바꾸도록 한다. 원래의 P_i값에서는, G_k에 속한 사용자 j의 $Profit(k,j)$ $(k \neq i)$가 $Profit(i,j)$보다 더 높은 값을 가진다. 이때 P_i를 감소시키면 $Profit(k,j)$ $(k \neq i)$는 그대로 있는 채 $Profit(i,j)$가 증가하여 레벨 i가 더 매력이 있게 된다. 따라서 각 사용자 j에 대해 조건 (B-2)를 만족시키는 ^jP_i를 구할 수 있다. (사용자 j가 레벨 i를 선택하도록 할 만큼만 ^jP_i를 감소시키면 된다.) 새 P_i값에 따라 의사결정을 바꾸는 사용자들의 수를 조절할 수 있다.

결론적으로 말해, P_i 를 증가시키면 G_i 그룹내의 사용자중 일부의 결정을 바꿀 수 있고, P_i 를 감소시키면 G_i에 속하지 않은 사용자중 일부의 결정을 바꿀 수 있다. 두 가지 경우 모두, 결정을 바꾼 사용자들의 유틸리티 함수 추정범위를 좁힐 수 있다. 이때 (1) 어느 사용자를 목표로 할지, (2) P_i를 증가시킬지 감소시킬지, (3) P_i를 어느만큼 증가/감소시킬지 등등의 문제는 5장에서 실험을 통해 조사 분석한다.

3.4 유틸리티 추정 알고리즘

이 장에서는 유틸리티 추정을 위한 알고리즘을 소개한다. 그림 2의 알고리즘은 점진학습을 이용하여 새 가격벡터를 구한다. 각 반복 사이클에서 오직 하나의 가격만을 변화시키므로, 점진학습에서는 가격을 변화시킬 목표 사용자 그룹을 정하는 것이 중요한 문제이다. 무작위로 목표 사용자 그룹을 선택할 수도 있다. 또는 가장 정보가 없는 사용자를 (즉, 상한과 하한의 차이가 가장 큰 사용자를) 포함한 그룹을 목표로 삼을 수도 있다. 또는 가장 많은 사용자를 포함하는 (즉, 가장 높은 일도착률을 가진) 레벨을 선택할 수도 있다. 4장에서 우리는 목표 그룹 선택시 여러가지 다른 기준을 적용하여 각 방식의 차이점을 분석해본다. 만약 목표 그룹의 사용자들의 결정을 바꿀 수 있는 그러한 가격이 존재하지 않는다면, 무작위로 가격을 책정하는 방식을 쓴다.

4. 평 가

이 장에서는 체계적인 시뮬레이션을 통하여 우리가 제안한 유틸리티 추정법을 평가한다. 우리의 주 관심사는 유틸리티 추정법의 효율성으로, 추정법이 (1) 주어진 시간 내에 얼마나 정확하게 사용자 유틸리티를 추정할 수 있는지, (2) 얼마나 빨리 사용자 유틸리티 함수를 알아낼 수 있는지를 측정한다. 이를 위해 점진학습시에 가격을 선정하는 여러가지 방법을 분석하였다.

Algorithm Progressive_Learning_Algorithm
Inputs: I, the number of service levels
N, the number of users in the system
$\{U_n\}^0$, the initial set of utility estimate
Δ_{max}, the time limit
Ω, the estimation-range size limit
Output: $\{U_n\}^F$, the final set of utility estimates

$U_{max} \leftarrow$ maximum possible utility value, given $\{U_n\}^0$.
Randomly choose the initial price vector $\{P_i\}^1$, where $P_i \geq P_{i+1}$ and $P_i \leq U_{max}$, for $\forall\, i$.
$\Delta \leftarrow 1$
Loop
Wait for users' decisions $\{L_n\}^\Delta$ under the price vector $\{P_i\}^\Delta$. // Section 3.1
Update the utility estimation $\{U_n\}^\Delta$. // Section 3.2
Update U_{max} from $\{U_n\}^\Delta$.
$\Delta \leftarrow \Delta + 1$.
If ($\Delta > \Delta_{max}$ **or** $|U_n| < \Omega$, for $\forall\, n$) **then**
Break;
Endif
Choose target level i.
Compute new P_i either by increase or by decrease to generate $\{P_i\}^\Delta$. // Section 3.3
If (P_i *that satisfies the conditions described in Section 3.3 cannot be computed*) **then**
Randomly choose the initial price vector $\{P_i\}^\Delta$, where $P_i \geq P_{i+1}$ and $P_i \leq U_{max}$, $\forall\, i$.
End_if
End_loop
$\{U_n\}^F \leftarrow \{U_n\}^\Delta$
End_algorithm

그림 2 점진학습을 통한 유틸리티 추정 알고리즘

따로 언급이 없는 한, 아래의 모든 실험에서 다음과 같은 환경을 사용한다. 네트워크에는 100명의 사용자가 있으며($N = 100$), 8개의 서비스 레벨이 제공된다($I = 8$). 각 사용자는 한 종류의 일을 가지고 있고, 유틸리티 함수는 (3.2장에 설명된) K와 α의 두 매개변수를 가지는 모형을 취한다[3]. 이때 K는 알려져 있고 α를 모른다고 가정한다. 모든 사용자들의 K는 6이고, 각 유틸리티 함수의 α값은 [-2.0, -0.5] 사이에 있다. 일처리율은 120이고 ($\mu = 120$), 총가능 일도착률은 100이다($\lambda^* = 100$). 일의 크기는 일의 타입에 관계없이 85 Kbits으로 한다. 이러한 설정은 약 10 Mbps의 용량을 갖는 네트워크를 본뜬 것이다(120 job/second × 85 Kbits/job = 10.2 Mbps). 모든 점진학습 실험에서 가격변화의 폭은 1로 한다. 가격의 최소단위는 0.00001로 정한다.[2]

시뮬레이션의 결과는 2 차원의 그래프로 나타내며, 이때 X축은 추정 사이클을 단위로 하며,[3] Y축은 평균추정오차(average estimation error)를 나타낸다. 평균추정오차는 다음과 같이 구해진다.

$$Average\ estimation\ error = \frac{\sum_{i}^{N} |\alpha_i - \alpha_i^*|}{N}.$$

여기서 α^*_i는 α_i의 현재 추정 범위의 중앙치(median)이다. 평균추정오차라는 성능 측정기준은 로그 스케일로 Y축에 그려진다. 통계적 의미를 위해 각 그래프는 500번의 실험결과를 평균하여 얻어진다.

점진학습은 목표 사용자 그룹을 정하여 그 서비스 레벨의 가격을 올리거나 내리는 방법이다. 이 과정에서 (1) 어느 서비스 레벨을 목표로 할지, (2) 목표 레벨의 가격을 올릴지 내릴지, (3) 가격을 얼마나 변화시킬지 등이 점진학습의 성능에 미치는 영향을 조사해 본다.

2) 가격의 최소단위는 소수점 다섯째 자리로 제한한다. 그 이상의 가격단위는 실용적이지도 않고 현실적이지도 않다고 본다.

3) 각 추정 사이클의 실제 시간은 가격벡터를 업데이트하는 빈도에 따라 다를 수 있다. 우리의 실험에서는 한 사이클은 5초로 정해져 있다.

먼저 가격변화의 "폭"이 점진학습의 성능에 미치는 영향을 알아보자. 3.3장에서 설명한 바와 같이, 네트워크는 가격변화의 폭 ($^{j}P_i$)에 따라 의사결정이 바뀌는 사용자의 수를 조절할 수 있다. 그림 3은 최대가능한 의사결정 변화 사용자의 수가 10, 5, 3, 1인 4가지 경우의 시뮬레이션 결과를 보여준다. 모든 실험에서 가장 덜 알려진 사용자 그룹을 목표 서비스 레벨로 선택하고, 목표 서비스 레벨의 가격을 증가시키는 방법을 사용하였다. 3.3장에서 설명된 조건 (A-2)와 (B-2)를 만족시키는 $Uj(\cdot)$를 계산할 때에는 현재 추정범위의 중앙치를 이용하였다.

그림 3은 큰 폭의 가격변화는 단기간에는 효과가 있으나, 적은 폭의 가격변화가 보다 정확한 추정을 가능하게 한다는 것을 보여준다. 큰 폭의 가격변화는 많은 수의 사용자들의 결정을 바꾸므로 초기에는 더 많은 정보를 얻어낼 수 있다. 적은 폭의 가격변화는 적은 수의 사용자들의 결정을 바꾸지만, 더 좁은 상한과 하한의 조건을 제공함으로써 더 정확한 추정을 가능하게 한다.

두번째로, 가격변화의 "방향"이 점진학습에 미치는 영향을 알아보자. 점진학습에서는 '증가', '감소', '증가/감소 섞임'의 세 가지의 가격변화 방법이 가능하다. 시뮬레이션 결과는 그림 4에 나타나 있다. 초기에는 가격을 '감소'시키는 것이 '증가'시키는 것보다 더 좋은 성능을 보이지만, 추정기간이 늘어날수록 그러한 추세는 역전된다. 가격을 감소시키는 것이 초기에 더 효과적인 이유는, 가격을 증가시키는 것보다 더 많은 사용자들에게 영향을 미치기 때문이다. 목표 서비스 레벨의 가격을 감소시키면 다른 서비스 레벨을 선택했던 사용자들이 목표 레벨을 선택하게 되며, 목표 서비스 레벨의 가격을 증가시키면 그 서비스 레벨을 선택했던 사용자들만이 그 영향을 받는다. 가격을 올리는 방법은 추정범위가 가장 큰, 덜 알려진 사용자들을 직접적으로 목표로 삼기 때문에 시간이 갈수록 더 효과적이다. 이에 반해 가격을 내리는 방법은 추정범위가 좁은, 더 알려진 사용자들에게 영향을 미친다. 따라서 만약 언제 가격변화의 방향을 바꿀지를 알 수 있다면, 초기에는 가격을 내리는 방법을 쓰고 후기에는 가격을 올리는 방법을 취하는 것이 가장 낫다. 하지만 언제 가격변화 방향을 바꾸어야 하는지를 아는 것은 쉬운 일이 아니다. 우리는 가격을 올리거나 내리는 결정을 무작위로 섞어쓰는 '증가/감소 섞임'을 사용하였다. 그림 4는 '증가/감소 섞임'이 증가와 감소 방식의 장점들을 잘 활용하는 것을 보여준다. 따라서 앞으로의 실험에서는 '증가/감소 섞임'을 기본으로 사용한다.

일반적으로 점진학습은 짧은 기간내에 빠르게 적당히 정확한 추정치를 제공하고, 긴 시간에 걸쳐 조금씩 추정의 정확도를 높여나간다. 그림 4에서 20 사이클 부근에서 평균추정오차가 0.01로 줄고, 200 사이클 부근에서 0.001로 떨어지며,. 300 사이클에서는 평균추정오차가 0.0005보다 작게 된다. 이 값들의 의미는 다음과 같다. 만약 10종류의 일들이 있고 (J = 10) 그들의 α 값이 [-2.0, -0.5] 사이에 균일하게 분포되어 있다면, 정확한 일의 타입을 알기 위해서는 추정치의 범위가 0.15이면 된다. 이러한 정도의 정확도는 몇 추정 사이클내로 얻어진다. 만일 100종류의 일이 있다면 (J = 100), 요구되는 평균추정오차의 값은 0.015이며 이는 20 사이클내에 이루어진다.

마지막으로, 목표 레벨의 "선택기준"이 점진학습에 미치는 영향을 알아본다. 선택기준은 (1) 무작위로 선택, (2) 사용자들의 유틸리티 함수 추정치의 평균 범위가 가장 큰 레벨을 선택, (3) 유틸리티 함수 추정치의 범위의 합이 가장 큰 레벨을 선택하는 세가지를 실험해 보았다. 그림 5는 (2)와 (3)이 무작위 선택을 능가하는 것을 보여준다. (선택기준 (2)와 (3)은 거의 비슷한 성능을 보여

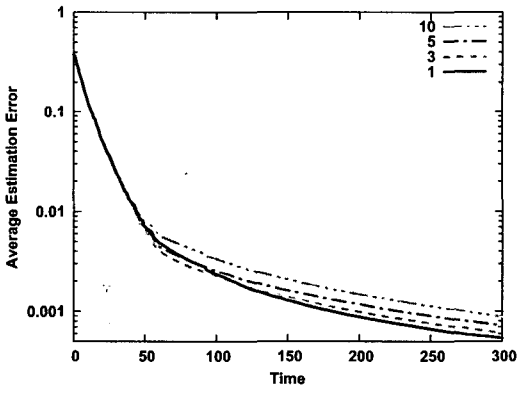

그림 3 가격증가의 폭에 따른 비교

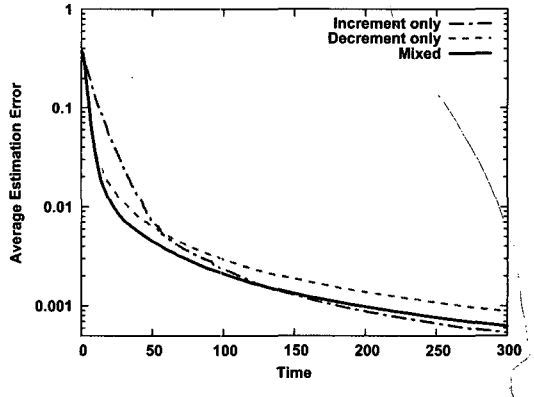

그림 4 가격변화의 방향에 따른 비교

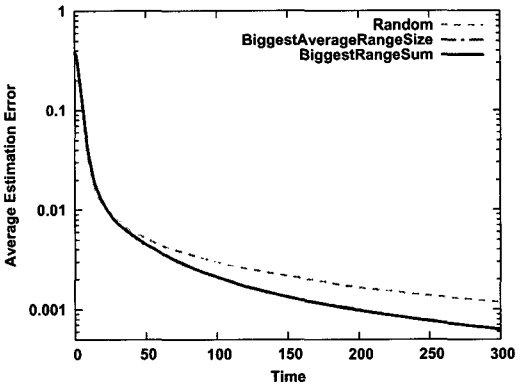

그림 5 목표 레벨 선택의 기준에 따른 비교

그림 5에서는 하나의 라인으로 중첩되어 보인다.) 이는 (2)와 (3)이 모두 덜 알려진 사용자들을 목표로 하기 때문이다.

5. 결 론

이 논문은 사용자 유틸리티 함수에 대한 부정확한 정보가 가지는 문제점을 인식하고, 그에 대한 해결책을 제시한다. 네트워크가 사용자 유틸리티 함수에 대한 정확한 지식을 가지고 있다는 가정은 최적가격정책에 대한 기존 연구의 취약점으로 비판되어왔다. 이러한 약점을 보완하기 위해 이 논문에서는 부정확한 초기 지식으로부터 유틸리티 함수에 대한 보다 정확한 정보를 추정해 나가는 방법을 제안하였으며, 체계적인 실험분석을 통해 긍정적인 결과들을 보여주고 있다.

참 고 문 헌

[1] Cocchi, R., S. Shenker , D. Estrin, and L. Zhang. Pricing in Computer Networks: Motivation, Formulation, and Example. *IEEE/ACM Transactions on Networking*, Vol. 1, pages 614-627. 1993.

[2] Mendelson, H. Pricing Computer Services: Queuing Effects. *Communications of the ACM*, Vol. 28, No. 3, pages 312-321. 1985.

[3] Mendelson, H., and S. Whang. Optimal Incentive-Compatible Priority Pricing for the M/M/1 Queue. Operations Research, Vol. 38, No. 5, pages 870-883. 1990.

[4] Altmann, J., H. Daanen, H. Oliver, A. Suarez. How to Market-Manage a QoS Network, *IEEE INFOCOM*. 2002.

[5] Mandjes, M. Pricing Strategies under Heterogeneous Service Requirements, *IEEE INFOCOM*. 2003.

[6] Mackie-Mason, J., and H. Varian. Pricing Congestible Network Resources. *IEEE JSAC*, Vol. 13, No. 7, pages 1141-1149. 1995.

[7] Odlyzko, A. A Modest Proposal for Preventing Internet Congestion. http://www.research.att.com/~amo. 1997.

[8] Mackie-Mason, J., L. Murphy, and J. Murphy. The Role of Responsive Pricing in the Internet, *Internet Economics*, MIT Press, pages 279-303. 1997.

[9] Kelly, F. On Tariffs, Policing, and Admission Control for Multiservice Networks, *Operations Research Letters*, pages 1-9. 1994.

[10] Kelly, F. Charging and Rate Control for Elastic Traffic, *European Transactions on Telecommunications*, Vol. 8, pages 33-37. 1997.

[11] Kelly, F., A. Maullo, and D. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability, *Journal of the Operation Research Society*, Vol. 49, pages 237-252. 1998.

[12] Courcoubetis, C., F. Kelly, and R. Weber. Measurement based charging in communication networks, *Operations Research*, pages 535-548. 2000.

[13] Jordan, S., and H. Jiang. Connection Establishement in High-Speed Networks, *IEEE JSAC*, Vol. 13, No 7, pages 1150-1161. 1995.

[14] Gupta, A., D. Stahl, and A. Whinston. A Stochastic Equilibrium Model of Internet Pricing. *Journal of Economic Dynamics and Control*, pages 697-722. 1997.

[15] Paschalidis, I., and J. Tsitsiklis. Congestion-Dependent Pricing of Network Services. *IEEE/ACM Transactions on Networking*, Vol. 8, No. 2, pages 171-184. 2000.

[16] Marbach, P. Pricing Differentiated Services Networks: Bursty Traffic, *IEEE INFOCOM*. 2001.

[17] Shenker, S., D. Clark, D. Estrin, and S. Herzog. Pricing in Computer Networks: Reshaping the Research Agenda, *ACM Computer Communication Review*, Vol. 26, pages 19-43. 1996.

[18] Bertsekas, D., and R. Gallager. *Data Networks*. Prentice-Hall. 1992.

부록: 논문에서 사용된 기호들

μ	Job service rate.
λ_i	Job arrival rate at level i.
λ_0	Rate of jobs that are not submitted.
$\{\lambda_i\}$	Job arrival rate vector, where λ_k at the k-th position represents the arrival rate at k-th service level.
λ^*	Total potential job arrival rate ($\sum_{i=0}^{I}\lambda_i$).
λ	Effective job arrival rate ($\lambda^*-\lambda_0$).
C_j	The average size of type-j job.
$U_j(t)$	Utility function of a type-j job.
P_i	Price of level-i service.

$\{P_i\}$ Price vector, where P_k at the k-th position represents the price of k-th service level.

T_{ij} Expected completion time of a type-j job at level i.

N Total number of users in the system.

$\{L_n\}$ User decision vector, where L_k at the k-th position represents the service level choice of k-th user.

$\{U_n\}$ User utility estimation vector, where U_k at the k-th position represents the estimated utility function of k-th user.

박 선 주

1989년 서울대학교 컴퓨터공학과(학사) 1991년 서울대학교 컴퓨터공학과(석사) 1999년 Univ. of Michigan, Ann Arbor, CSE (Ph.D). 1999년~2005년 Rutgers University, MSIS Department(Assistant Professor). 2005년~현재 연세대학교 경영학과(조교수). 관심분야는 에이전트 시스템, 전자상거래, 옥션, SCM, 네트워크 가격정책