준 공유 출력 버퍼형 스위치 구조

남 승 엽*, 성 단 근*, 안 윤 영**
*한국과학기술원 전자전산학과, **한국전자통신연구원

Quasi-Shared Output Buffered Switch

Seung Yeob Nam*, Dan Keun Sung*, Yoon-Young An**
*Dept. of Electrical Engineering and Computer Science, KAIST
**Electronics and Telecommunications Research Institute
E-mail: synam@cnr.kaist.ac.kr

Abstract

One major drawback of conventional output buffered switches is that the speed of writing cells into output buffer should be N times faster than input link speed. This paper proposes a new output buffer switch that divides one output buffer into several buffers and virtually shares the divided buffers by using a distributor. The proposed switch makes it possible to reduce the memory speed. The proposed switch is evaluated in terms of the average cell latency compared with the input buffered switches which use the arbitration alogorithms, i.e., iSLIP or wrapped wave front arbiter(WWFA).

I. 서론

지난 10여년 동안 통합 서비스 망에서 다양한 서비스를 제공하기 위한 많은 연구가 진행되어 오고 있다. 서비스 계약을 지키는 사용자에게 서비스 계약을 어기는 flow에 영향을 받지 않고 대역과 지연을 보장해주기 위해서 많은 스케줄링 방식이 제안되어 왔다. 이러한 연구는 주로 출력 버퍼형 스위치에 대해서 이루어져 왔는데 출력 버퍼형 스위치는 QoS 제공에 많은 장점을 가지고 있지만, 메모리 속도가 전체 입력속도의 합만큼 빨라야 한다는 한계가 있다. 기술적으로 메모리 속도의 증가율이 프로세서 속도의 증가율보다 낮기 때문에 앞으로는 메모리 속도가 고속 스위치를 만드는데 bottleneck이 될 것으로 예상된다. 반면 입력 버퍼형 스위치에서는 메모리 속도가 한 채널의 속도보다 많이 빠를 필요가 없다는 장점이 있다. 입력 버퍼형 스위치의 가장 큰 단점은 HOL(head-of-line) 블로킹으로 인해 수율이 58.6%밖에 나오지 않는다는 것이었는데 입력 버퍼에서 VOQ(virtual output queueing)방식으로 셀을 저장하면 수율이 100%까지 증가할 수 있음이 알려지면서 최근에 입력 버퍼형 스위치가 증가하고 있다[1,2].

그렇지만 입력 버퍼형 스위치에서 VOQ를 사용한다 하더라도 여전히 스위치의 입력단과 출력단에서 셀이 충돌할 수 있다는 문제점이 남아있다. 따라서, 일반적으로 입력단과 출력단에서 발생하는 충돌을 해결하면서 수율을 높이기 위해 중재(arbitration) 알고리즘이 사용된다. 점차적으로 서비스 품질에 대한 요구가 높아지기 때문에 중재 알고리즘은 앞으로 QoS도 고려하게 된다면 중재 알고리즘의 복잡성이 매우 높아지게 될 것이라 예상할 수 있다. 알고리즘의 복잡성은 계산 시간이 길어질 수 있음을 의미하므로 스위치 고속화에 장애물이 될 수 있다.

출력 버퍼형 스위치에서는 스위치 내부 속도 상승(speed-up)으로 인해 입력단과 출력단에서 충돌이 발생하지 않게 되고 따라서 출력단에서 스케줄링 알고리즘은 QoS만 고려하면 된다는 장점이 있다.

본 논문은 지금까지 설명한 입력 버퍼형 스위치의 장점과 출력 버퍼형 스위치의 장점을 모두 고려해서 서비스 품질 보장에 유리한 출력 버퍼형 스위치의 구조를 기본으로 해서 입력 버퍼형 스위치처럼 내부 속도 상승을 많이 하지 않고 동작할 수 있는 새로운 스위치 구조를 제안한다. 그리고, 그 스위치의 성능을 입력 버퍼형 스위치에서 100%의 수율이 가능하게 한 iSLIP[3], WWFA[4] 등의 중재 알고리즘을 사용하는 입력 버퍼형 스위치와 비교 평가한다.

II. 준 공유 출력 버퍼형 스위치의 구조

2.1 준 공유 출력 버퍼형 스위치의 기본 구조

그림 1은 NxM 크기로 내부 속도 상승을 하지 않으면서 출력 버퍼형 스위치처럼 동작하는 스위치 구조를 보

여준다. 일단 입력 포트별로 라우팅 블록이 할당되어 있고, 각 출력 포트별로 스케줄러가 할당되어 있다. 스케줄러 앞에는 셀이 들어온 입력 포트에 따라 구별되어 저장될 수 있도록 버퍼가 나누어져 있고, 각 라우팅 블록은 대응되는 버퍼들과 버스로 연결되어 있다. 구체적인 동작은 일단 셀이 입력 포트를 통해 들어오면 처음으로 거치는 라우팅 블록에서는 헤더 정보를 보고 실제 스위치 내부에서 사용할 라우팅 태그를 셀의 앞에 달아주는 역할을 한다. 라우팅 태그는 멀티캐스팅을 고려해 i번째 포트로 셀을 보내기 원하는 경우에 아래에서 i번째 bit가 1로 세팅되고 그렇지 않은 경우 0인 Mbit의 워드로 이루어지고, 버퍼에서 QoS를 고려해 우선 순위별로 큐를 나누어서 관리하는 경우에는 우선 순위 정보도 추가가 될 수 있다. 라우팅 블록에서 라우팅 태그를 달아주면 버스를 통해 모든 출력 포트의 해당되는 버퍼쪽으로 보내진다. 버퍼 앞에는 필터가 셀의 라우팅 태그를 보고 자신에게 해당되는 셀만 받아들이게 된다. 각 출력 포트는 입력 포트별로 분리된 셀버퍼를 갖는다. 이는 출력버퍼형 스위치에서 출력 버퍼를 입력 포트의 개수만큼 나누고 내부적인 speed-up이 없도록 병렬로 동작시키는 구조라고 볼 수도 있다.

이 스위치는 여러 가지 장점을 가진다. 스위치 내부적으로 speed-up이 필요하지 않다는 점과 출력버퍼 스위치처럼 동작하기 때문에 스위치 입력단과 출력단에서 contention이 발생하지 않는다는 장점이 있다. 따라서, 출력단에서는 오직 QoS만을 고려해 스케줄링할 수 있다는 장점이 있다.

한가지 단점은 아래 스위치를 구현하는데 필요한 버퍼의 개수가 NM으로 많다는 것이다. 이 단점을 개선한 스위치 구조가 그림 2와 같다. 기본적인 동작은 그림 1의 버퍼 분리형 스위치와 유사하고, 버퍼 분리형 스위치에서 필터와 버퍼 사이에 분배기를 두어 N개의 입력단에서 오는 셀을 N보다 작은 L개의 버퍼에 균등하게 나누어주는 점이 다르다. L값을 줄임으로써 필요한 전체 버퍼의 개수를 줄일 수 있으나,

출력 포트별 버퍼의 개수(L) ×RAM(버퍼) write 속도

= N ×입력 링크 속도

와 같이 버퍼의 수와 메모리 속도 사이에 반비례 관계가 있기 때문에 메모리 속도를 고려해 버퍼의 개수를 정해야 한다.

분배기가 도착하는 셀을 round-robin방식으로 L개의 버퍼에 분배하고 스케줄러가 round-robin 방식으로 L개의 버퍼를 서비스하게 되면 각 버퍼에 있는 셀의 수는 2 이상 차이가 나지 않는다. 버퍼의 크기 면에서 이와 같이 L개의 버퍼를 사용하면 L개 버퍼 크기의 공유버퍼를 사용하는 것과 차이가 나지 않는다. 따라서, 버퍼가 공유되는 것처럼 동작한다고 할 수 있고 모든 버퍼를 효과적으로 이용할 수가 있다. L개 버퍼 크기의 공유 버퍼를

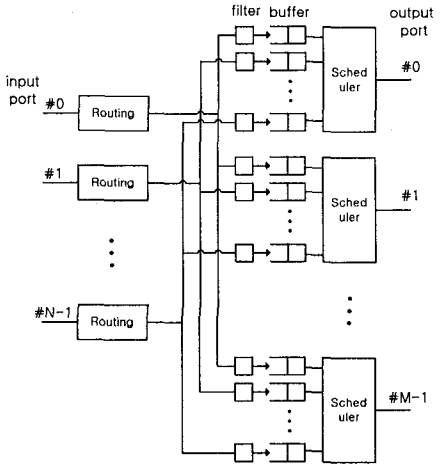

그림 1. 버퍼 분리형 스위치

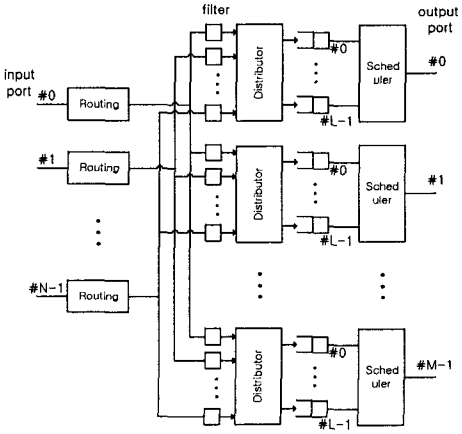

그림 2. 준 공유 출력 버퍼형 스위치

사용하는 경우에 비해 메모리 속도를 낮게 동작시킬 수 있기 때문에 단순한 출력 버퍼형 스위치에 비해 고속화에 적합하다는 장점이 있다.

2.2 QoS를 고려한 준 공유 출력 버퍼형 스위치 구조

그림 2의 스위치에서 분배기가 round-robin으로 셀을 L개의 버퍼에 차례대로 분배하고 스케줄러가 round-robin으로 서비스를 하게되면 준 공유 출력 버퍼형 스위치는 논리적으로 출력 포트당 버퍼가 하나 있고 각 버퍼는 FIFO 방식으로 서비스되는 출력 버퍼형 스위치와 동일하게 동작하게 된다.

여러 사용자가 서로 다른 품질을 요구할 수 있다는 점을 고려하지 않고 모두 동등하게 생각한다면 이와 같이 출력 버퍼에서 연결 또는 class를 구별하지 않고 FIFO

방식으로 서비스를 해도 상관없지만, 실제로는 서로 다른 종류의 트래픽이 서로 다른 품질을 요구하게 되고 그러한 품질을 제공하기 위해서는 출력단에서 FIFO가 아닌 스케줄링 방식도 지원해야 한다.

그런데 그림 2와 같은 스위치 구조에서는 같은 연결에 속하는 셀이라 하더라도 서로 다른 버퍼에 들어갈 수 있기 때문에 연결별로 셀을 관리하기가 힘들고 따라서 연결별 또는 클래스별 스케줄링이 쉽지 않다. 그림 2의 스위치 구조를 QoS 보장을 고려해 개선한 구조가 그림 3과 같다.

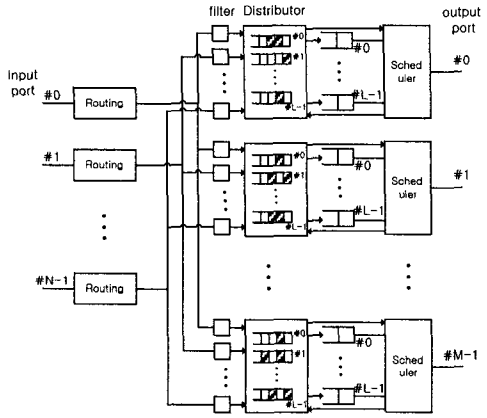

그림 3. QoS를 고려한 준 공유 버퍼형 스위치 구조

그림 2의 준 공유 버퍼형 스위치 구조와 그림 3에서 QoS를 고려한 구조의 차이점은 먼저 분배기의 경우 QoS를 고려하지 않은 구조에서는 단지 입력되는 여러 셀을 집중시킨 round-robin 방식으로 하나씩 버퍼에 전달하는 역할을 했는데, QoS를 고려하는 구조에서는 스케줄러가 반드시 round-robin으로 서비스를 하는 것이 아니기 때문에 각 버퍼의 길이가 서로 차이가 날 수 있고 분배기는 가능한 효율적인 버퍼의 사용을 위해 각 버퍼의 사용상황을 체크해서 점유율이 낮은 버퍼부터 셀을 채워나가는 점이 다르다.

셀이 라우팅블록을 거쳐서 필터에 전달되기까지의 과정은 차이가 없기 때문에 그림 3에서 셀이 필터를 거쳐 분배기에 입력되는 순간부터 스케줄러를 통해 서비스를 받기까지의 과정만 좀 더 자세히 살펴보면 다음과 같다.

① 그림 3에서 보는바와 같이 각 분배기는 각 버퍼별로 셀이 몇 개 있는지와 어떤 어드레스가 idle인지에 관한 정보를 기억하고 있다.

② 셀이 분배기(distributor)에 입력되면 분배기는 각 버퍼별 셀 점유율(저장된 셀의 개수)을 보고 가장 낮은 점유율의 버퍼를 선택한다. 분배기에 입력된 셀이 1 개보다 많으면 그 개수만큼 낮은 점유율의 버퍼를 선택한다.

③ 선택된 버퍼에서 idle 어드레스 가운데 셀을 저장할 주소를 얻는다.

④ 셀을 그 주소에 저장하고, 셀의 헤더와 셀이 저장된 주소를 스위치 내부 제어 셀에 실어 스케줄러로 보낸다.

⑤ 스케줄러는 제어 셀을 받아 헤더와 메모리 주소를 보고 헤더로부터 그 셀의 서비스 class 또는 해당되는 연결(virtual connection:VC)을 판단해 per-class queue 또는 per-VC queue 형태로 셀의 어드레스를 저장한다.

⑥ 스케줄러는 QoS를 보장할 수 있는 스케줄링 알고리즘(WFQ, SCFQ, etc.)을 이용하여 서비스를 받을 queue를 선택한다.

⑦ 선택된 queue의 맨 앞에 있는 셀의 실제 저장 위치를 per-class 또는 per-VC형태의 셀 address queue에서 얻은 다음 그 셀을 메모리의 해당되는 위치에서 가져와 서비스한다.

⑧ 스케줄러가 방금 서비스된 셀이 저장되었던 위치를 분배기에 알려주면 분배기는 각 버퍼별 점유율과 idle 어드레스의 리스트를 갱신한다.

⑨ 다음 셀 타임에 새로운 셀이 분배기에 도착하면 ①번부터 다시 차례로 반복한다.

즉, 분배기는 RAM으로 구현되는 각 버퍼의 idle 주소와 각 버퍼의 점유율을 관리하고 스케줄러 블록에서는 per-VC 또는 per-class 큐를 관리하기 때문에 스케줄링 알고리즘의 적용이 가능하고, 따라서, QoS 지원이 가능한 스위치 구조임을 알 수 있다.

III. 성능 평가

3.1 시뮬레이션 환경

대상 스위치의 크기는 8 x 8로 했고, 각 입력 포트마다 하나의 트래픽원이 연결되어 있고 각 트래픽원은 모든 출력포트에 대해 동등하게 셀을 발생시키는 uniform traffic 환경하에서 시뮬레이션을 했다.

입력 트래픽으로는 Bernoulli 트래픽과 버스티 트래픽으로 on-off 트래픽을 사용했다. Bernoulli 트래픽의 각 셀타임당 셀의 발생확률을 바꾸어서 입력 부하를 조절하였다. on-off 트래픽에서 ON과 OFF 구간의 길이는 지수 분포를 갖도록 했고, 온 구간 내에서는 일정한 간격으로 셀이 발생하도록 하였으며 같은 ON 구간 내에서 발생한 셀들의 출력포트는 모두 동일하게 두었다. 이때 ON 구간동안 발생한 셀의 수는 기하분포를 갖게 되고, OFF 구간동안에는 셀이 발생하지 않는다. 평균 버스트 길이는 16셀로 두었다.

3.2 시뮬레이션 결과

위와 같은 환경하에서 이 논문에서 제안한 준 공유 출력 버퍼형 스위치와 iSLIP 또는 WWFA으로 arbitration

을 한 입력 버퍼형 스위치의 성능을 비교 평가해 보았다.

그림 4는 앞에서 설명한 베르누이 트래픽이 가해졌을 때 준 공유 버퍼형 스위치에서 발생하는 지연의 평균과 iSLIP 또는 WWFA로 중재한 입력 버퍼형 스위치에서 발생한 지연의 평균을 가해진 부하에 따라 비교하고 있다. iSLIP으로 중재한 입력 버퍼형 스위치의 성능이 가장 나쁘고 준 공유 버퍼형 스위치의 성능이 가장 좋게 나타남을 확인할 수 있다. uniform한 베르누이 트래픽이 가해지는 경우 3 가지 경우 모두 100%의 수율이 나옴을 확인할 수 있다.

그림 5는 on-off 트래픽이 가해지는 경우 준 공유 버퍼형 스위치와 iSLIP/WWFA 방식으로 중재한 입력버퍼형 스위치의 성능을 비교해서 보여주고 있다. 그림 4와 마찬가지로 준 공유 버퍼형 스위치의 성능이 가장 좋게 나오고 있고, iSLIP으로 중재한 입력 버퍼형 스위치의 성능이 가장 좋지 않게 나오고 있다. 이와 같이 어떠한 입력 트래픽에 대해서도 준 공유 버퍼형 스위치의 성능이 좋게 나오는 이유는 2 장에서 설명했던 것과 같이 준 공유 버퍼는 논리적으로 출력포트에서 FIFO로 스케줄링하는 출력 버퍼형 스위치와 동일하게 동작하고 있기 때문이다. 입력 버퍼형 스위치는 아무리 좋은 알고리즘을 사용한다 하더라도 기본적으로 스위치의 입력포트와 출력포트에서 충돌이 발생하는 문제가 있기 때문에 출력 버퍼형 스위치만큼 성능이 나오기 어렵다.

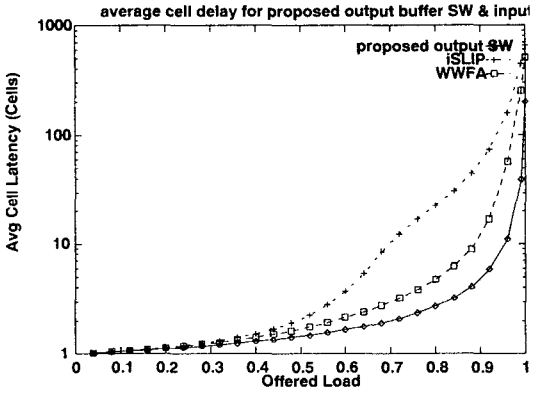

그림 4. 베르누이 입력 트래픽이 가해졌을 때 제안된 스위치와 입력 버퍼형 스위치의 성능 비교

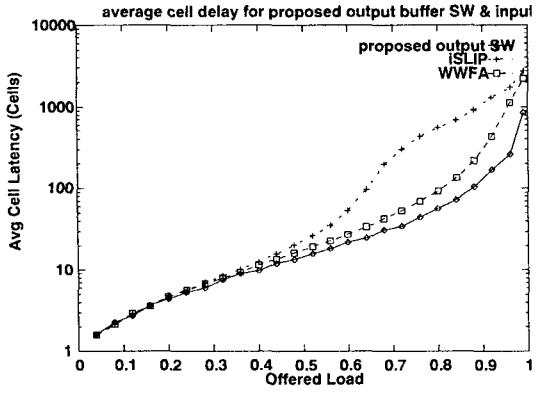

그림 5. 버스티 트래픽이 가해졌을 때 제안된 스위치와 입력 버퍼형 스위치의 성능 비교

IV. 결론

점차적으로 메모리 접속 속도가 고속 스위치를 구현하는데 있어서 bottleneck으로 나타남으로써 같은 입출력 링크 속도에서 높은 메모리 접속 속도를 요구하는 출력 버퍼형 스위치보다 입력 버퍼형 스위치가 선호되고 있지만 입력 버퍼형 스위치는 출력 버퍼형 스위치에 비해 성능이 낮고, QoS를 지원하기가 쉽지 않다는 단점이 있다.

이 논문에서는 논리적으로 출력 버퍼형 스위치와 동일하게 동작하면서 스위치 내부 속도 상승과 메모리 접속 속도를 줄일 수 있는 준 공유 출력 버퍼형 스위치 구조를 제안했다. 메모리 접속 속도를 일반 출력 버퍼형 스위치보다 많이 줄일 수 있기 때문에 고속 스위칭에 적용이 가능하다.

시뮬레이션 결과 좋은 성능으로 알려진 중재 알고리즘을 사용한 입력 버퍼형 스위치에 비해 좋은 성능을 나타냄을 확인할 수 있었다.

참고문헌

[1] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *Proc. IEEE INFOCOM'96*, San Francisco, CA, pp. 296-302.

[2] A. Mekkittikul and N. McKeown, "A practical scheduling algorithm for achieving 100% throughput in input-queued switches," in *Proc. INFOCOM'98*, San Francisco, CA, vol. 2, pp. 792-799.

[3] N. McKeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACE Trans. on Networking*, vol. 7, no. 2, pp. 188-201, Apr. 1999.

[4] Hsin-Chou Chi and Yuval Tamir, "Decomposed Arbiters for Large Crossbars with Multi-Queue Input Buffers," *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 233-238, 1991.