고속 네트워크 스위치에서의 QoS보장을 위한 아웃풋 큐 구조

김성원∪ 김종권
서 울 대 학 교 전 기. 컴 퓨 터 공 학 부
{swkim, ckim}@popeye.snu.ac.kr

Advanced Pipelined Heap Architecture for Output Queueing Switches

Sung-Won Kim∪ Chong-Kwon Kim
School of Computer Science and Engineering, Seoul National University

요 약

본 논문에서는 여러 단계의 QoS(Quality of Service)를 지원하면서 빠르고 확장이 용이하며 각종 패킷 폐기(packet drop) 방식을 지원하는 평형 파이프라인 우선순위 아웃풋 큐 구조(balanced pipelined priority output queue architecture)를 제시하고 있다. 본 방안은 기존에 연구된 파이프라인 우선순위 힙(pipelined heap, P-heap)[1]을 기반으로 하고 있다. 파이프라인 우선순위 힙은 우선순위에 따라 패킷을 전송하는 작업을 파이프라인 방식으로 처리하여 처리 성능을 향상시킨 아웃풋 큐 구조이다. 그러나 P-heap은 평형성(balance) 문제를 전혀 고려하고 있지 않으며, 다양한 패킷 폐기 방안을 제공하고 있지 못하다. 본 논문에서는 이런 측면에서 P-heap을 개선한 Advanced P-heap을 제안하고 있다. Advanced P-heap은 평균적인 상황에서 힙에 평형성을 부여하고, 각종 패킷 폐기 정책을 지원할 수 있는 일반적인 우선순위별 차별 패킷 폐기 구조를 제시하고 있다.

1. 서론

다가올 미래의 패킷 네트워크에서는 여러 가지 QoS 요구사항을 가지는 각종 실시간 응용이 보편화될 것으로 예상된다. 초고속 네트워크 환경에서 QoS 요구 사항들을 보장하기 위해서 여러 가지 방안들이 고안되어 있는데, 이들 중 많은 수가 네트워크 자원에 대한 우선순위 사용을 통해 각각의 QoS 요구사항을 충족하도록 하고 있다. 패킷 전송의 측면에서 우선 순위 패킷 전송이 가능하기 위해서는 무엇보다도 네트워크 상의 라우터에서 우선 순위 패킷 전송을 가능하게 해주는 아웃풋 큐의 고안이 절실하다. 우선 순위 패킷 전송은 결국 아웃풋 큐를 힙으로 구성하는 문제로 귀결되는데, 초고속 네트워크 환경에서 작동하기 위해서는 매우 빠른 실시간 정렬 알고리즘이 지원되어야만 한다. 이를 위해서 고안된 방식이 파이프라인 우선순위 큐 구조이고 이 구조의 핵심은 파이프라인 힙(P-heap)에 있다. P-heap의 구조는 이진 힙(binary heap)[2]의 구조와 유사하다. 그러나 파이프라인을 구현하기 위해 복잡한 처리방식이 필요한 이진 힙과는 다르게, P-heap은 enqueue/dequeue시의 힙 처리 순서를 루트-단말 방향으로 통일시킴으로써 복잡한 하드웨어의 도입없이 간단하게 파이프라인이 구현 가능하도록 하였다. 그리고, 이를 통해 빠른 처리 속도와 좋은 확장성을 보장받을 수 있었다.

그러나 P-heap은 enqueue/dequeue시 힙의 균형문제를 전혀 고려하지 않고 있으며, 따라서 우선순위의 종류가 그리 많지 않은 환경에서는 힙의 편향화 문제로 인해 처리속도의 저하를 유발하게 된다. 편향화 문제를 보완하기 위해 본 논문에서는 P-heap에 평형성을 부여한 balanced P-heap을 제시하고 있다. 또한, P-heap 구조에서는 패킷 폐기에 대해 고려하고 있지 않으므로, 이 방식을 그대로 적용할 경우, 우선순위에 상관없이 패킷 수에 의한 후위 폐기(drop tail) 방식만을 적용할 수 밖에 없게 된다. 본 논문에서는 이를 개선하여, 우선순위별로 손쉽게 패킷 폐기을 가능케하는 구조를 제시하여 사용자가 원하는 패킷 폐기 정책에 따라 패킷을 폐기하는 것이 가능하도록 하였다.

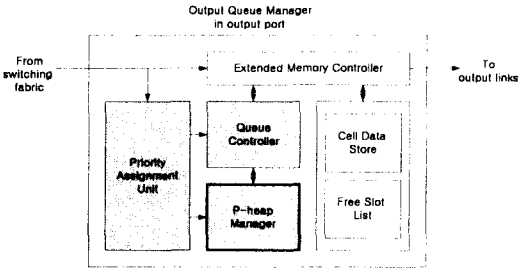
본 논문의 나머지 부분들은 다음과 같이 구성된다. 2장에서는 기존의 파이프라인 우선순위 큐 구조에 대해서 간략하게 설명한다. 3장과 4장에서는 본 논문에서 제안한 방식인 Advanced P-heap 구조의 힙의 평형성 개선 방법과 우선순위별 패킷 폐기 구조에 대해서 설명한다. 5장에서 결론을 짓고 향후 개선 방향을 제시한다.

2. 파이프라인 우선순위 큐 구조

본 장에서는 기존에 제시된 P-heap 구조에 대해서 설명한다.

2.1 P-heap 구조의 전체적인 모델

P-heap을 지원하기 위해서는 스위치의 각 아웃풋 포트별로 동적 큐 관리자(dynamic queue manager)가 필요하다.[5] P-heap 구조는 [그림 2-1]과 같이 우선순위 할당기(Priority Assignment Unit), 큐 제어기(Queue Controller), P-heap 관리자(P-heap Manager)의 크게 3부분으로 나누어볼 수 있다. 우선순위 할당기는 사용되는 스케줄링 기법에 따라 큐 관리자로 들어온 패킷에 우선순위를 부여한다. P-heap 관리자는 우선순위 값으로 구성된 힙인 P-heap 우선순위 큐(P-heap priority queue)를 포함하고 있다. 큐 제어기는 각각의 우선순위에 해당하는 실제 패킷의 리스트에 대한 포인터를 룩업 테이블(lookup table)로 저장하고 있다. 따라서 P-heap 구조는 우선순위별 큐잉을 지원한다. 실제 셀 데이터는 Cell Data Store에 저장되고 실제 저장된 데이터에 대한 접근 제거는 Ext. Memory Controller가 담당한다.



[그림 2-1] P-heap 구조의 전체적인 모델

2.2 P-heap의 자료구조

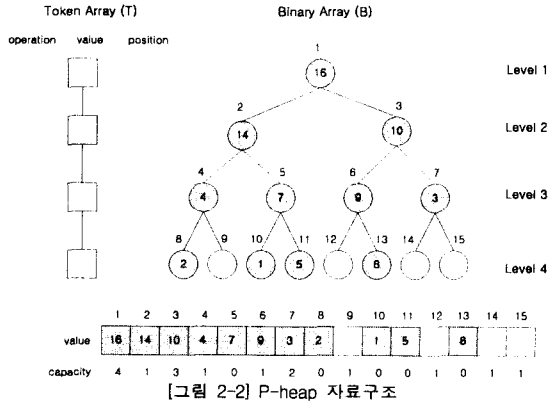
우선순위 큐를 구성하기 위해 일반적으로 이진 힙을 사용한다. 그러나 이진 힙의 경우 일반적인 enqueue/dequeue 작동 방법으로는 파이프라인이 쉽지 않다. P-heap은 이진 힙에서의 간단한 enqueue/dequeue 작동을 가능하게 하고 있으며 이를 위해서 [그림 2-2]와 같은 자료 구조를 사용하고 있다.

P-heap은 B,T의 두개의 배열로 구성된다. B는 실제 우선순위 값이 저장되어 있는 이진 힙을 이루는 배열이다.

B[i].active: 노드에 값이 저장되어있는가를 나타내는 플래그
B[i].value: 노드에 저장되어 있는 우선순위 값
B[i].capacity: B[i]를 루트로 하는 서브 트리의 빈 노드 개수

T는 토큰 배열이라고 부르며 P-heap의 파이프라인 작동을 돕기 위해서 고안된 배열이다.

T[i].operation: 힙의 각 레벨에서 이루어지고 있는 작업 종류
T[i].value: 저장되어야할 우선순위 값
T[i].position: 작업이 이루어지고 있는 P-heap 노드의 인덱스



[그림 2-2] P-heap 자료구조

2.3 P-heap의 작동

P-heap은 enqueue/dequeue/enqueue-dequeue의 세가지 작업을 지원한다. 위의 3가지 작업은 모두 하향 방식으로 실행되도록 일치시키고 백트래킹을 제거함으로써 파이프라이닝을 가능하도록 하고 있다.[1]

3. Advanced P-heap 구조의 balanced P-heap

P-heap은 이진 힙에서 파이프라인을 구현하였으나 이를 구현하기 위해서 필연적으로 이진 힙의 기본 성질 중 몇 가지를 희생하였다. 일반적인 이진 최대 힙의 경우 complete tree이고 따라서 상당한 균형성을 유지한다. 그러나 P-heap은 enqueue/dequeue시에 균형성에 대한 어떠한 고려도 하고 있지 않다. P-heap은 enqueue시에 양쪽 자식 노드 중에서 왼쪽 서브 트리에 남은 공간이 존재하는 경우 왼쪽 우선 순위로 enqueue하기 때문에 상대적으로 왼쪽 노드로 편향되는 현상이 나타난다. 이런 현상은 파이프라인을 위해 균형성을 고려하지 않은 dequeue방식을 사용함으로써 더 두드러지게 나타나게 된다. 특히 최대 허용할 수 있는 우선순위의 개수에 비해 상대적으로 적은 수의 우선순위들이 빠르게 변화하면서 갱신되는 환경에서는 편향성으로 인한 작업 처리 시간의 낭비가 심각해진다.

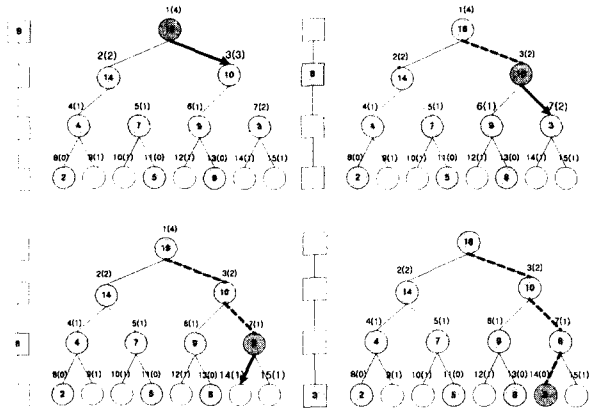
이를 보완하기 위해서 Advanced P-heap 구조에서는 balanced P-heap을 고안하였다. balanced P-heap은 P-heap의 enqueue 과정을 개선한 것으로 P-heap이 일반적인 상황에서 거의 균형을 유지할 수 있도록 해준다. P-heap의 각 노드는 이미 자료구조상으로 capacity라는 필드를 지니고 있는데 이는 2.2절에서 설명한 바와 같이 해당 노드를 루트로 하는 서브트리상의 빈 노드의 개수이다. balanced P-heap은 바로 이 필드를 이용한다. 기존의 P-heap은 새로운 값이 enqueue될 경우 루트 노드로부터 새 값과 비교를 시작하여 필요에 따라 상호교환 후 좌측 자식 노드가 포화되지 않는 한 좌측 노드로 작업을 진행하는데 반해, balanced P-heap은 좌측 자식 노드와 우측 자식 노드의 capacity 필드를 비교하여 더 빈공간이 많은 자식 노드 방향으로 작업을 진행해 간다. 따라서 빈 공간이 많은 쪽으로 새로운 값이 계속 전달되어 위치하게 되므로 전체적으로 결국 평형성을 높이게 된다. 자세한 작동 방식은 [그림 3-1][그림 3-2]에 나와있다.

```
procedure balanced_enqueue(j)
begin
    i <= T[j].position;
    v <= T[j].value;
    if B[i].value = false
        B[i].value <= v;
        B[i].active <= true;
        Decrement B[i].capacity;
        return done;
    elsif T[j].value > B[i].value
        Swap T[j].value, B[i].value;
        Move T[j].value to T[j+1].value;
    end if;
    if B[left(i)].capacity > B[right(i)].capacity && B[left(i)].capacity > 0
        T[j+1].position <= left(i);
    else
        T[j+1].position <= right(i);
    end if;
    return not_done;
end procedure;
```

[그림 3-1] balanced P-heap enqueue 알고리즘

물론 이 경우에도 dequeue시에는 평형성을 위한 처리를 해주지 않으므로, 힙이 완전히 평형 트리가 될 수는 없겠지만, 새로운 우선순위의

도착 비율과 제거 비율이 비슷한 일반적인 네트워크 상태에서는 dequeue작업에 의해 생성되는 빈자리가 enqueue에 의해 채워지는 작업이 반복되므로 평형성을 유지하게 된다.



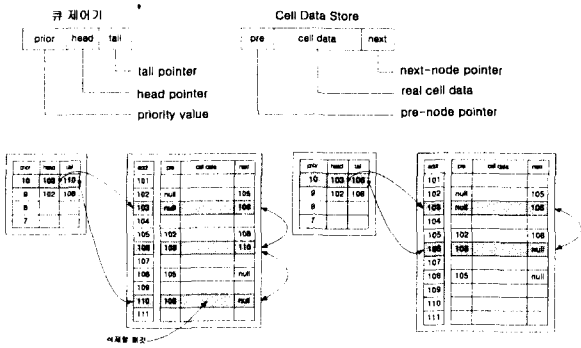
[그림 3-2] balanced P-heap enqueue 작동 예

4. Advanced P-heap 구조의 우선순위별 패킷 폐기 구조

P-heap 구조에서는 패킷을 폐기시키는 것에 대해서는 고려하고 있지 않으므로, 우선순위에 상관없이 전체적으로 후위 폐기 아웃풋 큐와 같은 방식으로 동작한다. 즉, Cell Data Store가 가득찰 때까지는 조건없이 패킷을 받아들이다가 가득찰 경우 새로 들어오는 패킷들을 폐기시키는 방법을 취한다. 그러나 QoS를 제공하기 위한 여러 가지 정책에 따라 후위 폐기 방식 외의 다른 폐기방식을 필요로 하는 경우가 많이 존재한다. 즉, 큐에 남은 공간이 부족할 경우, 우선순위가 높은 패킷이 들어오면 낮은 패킷을 폐기시키고 우선순위가 높은 패킷을 받아들이는 등의 방식을 일례로 들 수 있다.

기존의 P-heap 구조에서는 각 우선순위별 실제 데이터 패킷은 리스트의 형태로 Cell Data Store에 저장되고 Queue Controller가 그 포인터를 가지고 있게 되는데, 이런 구조에서는 이미 들어온 패킷을 폐기시키고 필요 공간을 확보하는 방법이 쉽지 않다. 이 구조에서 패킷을 폐기시키기 위해서는 현재 큐잉되어 있던 패킷중에서 가장 먼저 들어온 패킷을 제거하는 수밖에 없는데, 이는 큐가 혼잡한 순간보다 이전 순간에 들어온 패킷들을 제거하게 되어, 네트워크 혼잡과 패킷 폐기의 시간이 불일치 하는 문제를 낳을 수 있다. 또한, 가장 먼저 들어온 패킷을 폐기함으로써 그 이후의 패킷들이 모두 무효화되어버리는 문제를 낳을 수도 있다. 따라서 패킷 폐기시에는 우선순위별로 가장 최근에 들어온 패킷을 제거하는 것이 타당하다.

이를 위해서 Advanced P-heap 구조에서는 [그림 4-1]과 같은 구조를 고안하였다. 여기서 가장 독특한 점은 Cell Data Store에 저장하고 있는 패킷을 더블 링크트 리스트(doubly linked list)를 사용해서 관리한다는 점이다. 더블 링크트 리스트는 특정 노드에 대한 부모 노드와 자식 노드에 대한 양방향 포인터를 가지고 있어서 양방향으로 자유롭게 리스트를 순회할 수 있다는 특징이 있다. 그리고 큐 제어기는 더블 링크트 리스트를 지원하기 위해서 우선순위별 리스트의 헤드 포인터와 테일 포인터를 보유하게 된다. 이렇게 더블 링크트 리스트를 사용함으로써 특정 우선순위의 패킷을 폐기해야 하는 경우 큐 제어기의 테일 포인터에 해당하는 패킷을 폐기하고 테일 포인터만 갱신해주면 된다.



[그림 4-1] 우선순위별 패킷 폐기를 위한 자료구조 및 작동 방식

여러개의 패킷을 폐기해야 하는 경우에도 리스트를 순회하면서 패킷들을 폐기하고 한번의 테일 포인터 갱신을 해주면 되므로 빠른 속력을 지원할 수 있다. 이와 같은 방식으로 원하는 우선순위의 패킷들을 원하는 만큼 폐기하는 것이 가능하므로, 각종 패킷 폐기 정책에 일반적으로 적용 가능한 구조라 할 수 있다.

5. 결론 및 향후 개선방향

여러 가지 QoS에 대한 요구사항들이 점점 늘어가고 있는 현재 네트워크 상황에 적용 가능한 아웃풋 큐 구조인 P-heap 구조를 개선한 Advanced P-heap 구조를 제안하였다. Advanced P-heap 구조는 균형성에 대한 고려가 없었던 기존의 P-heap의 작동 방식을 개선하여, P-heap에 평균적인 균형성을 부여함으로써 각 패킷에 대한 딜레이를 줄였다. 그리고 우선순위별 서비스의 차별화를 지원하기 위해 우선순위별 패킷 폐기를 지원할 수 있는 방법을 제시하였다. 이 패킷 폐기 방안은 특정 패킷 폐기 정책에 한정되지 않고, 여러 가지 정책에 자유롭게 적용 가능하도록 일반적인 구조로 제안되었다.

향후에는 Advanced P-heap을 여러 가지 구체적인 정책에 실제로 적용하여 어떤 기능들이 적용가능한가를 살펴보고, 성능향상에 대해 분석할 예정이다. 특히 우선순위별 RED나 우선순위별 차등 패킷 폐기 등에 대한 연구를 진행할 예정이다.

6. 참고 문헌

[1] Bhagwan, R., Lin, B., "Fast and scalable priority queue architecture for high-speed network switches", INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE , Volume: 2 , 2000 Page(s): 538 -547 vol.2

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "Introduction to algorithms", McGraw-Hill Book Company, ISBN 0-07-013143-0.

[3] V. N. Rao and V. Kunner, "Concurrent Access to Prioirty Queues", In IEEE *transactions of Computers*, vol.37, Dec 1988.

[4] S. K. Prasad, S. I. Sawant, "Parallel Heap: A Practical Priority Queue for Fine to Medium-Grained Applications on Small Multiprocessors", The Seventh Symposium on Parallel and Distributed Processing, San Antonio, TX 1995.

[5] Y. Chen and J. S. Tunner, "Dynamic Queue Assignment in a VC queue Manager for Gigabit ATM Networks", Proceedings of the IEEE ATM Workshop, 1998.