메모리 자원 사용 효율성 증진을 위한 적응적 네트워크 이중 버퍼 모델

(An Adaptive Network Double Buffer Model for Efficient Memory Resource Usage)

최 창 범 † 이 승 룡 ††
(Daniel Choi) (Sung Young Lee)

요 약 본 논문에서는 네트워크 통신에서 혼잡으로 인한 패킷의 손실을 최소화하기 위하여 새로운 버퍼 모델인 적응적인 이중 버퍼 모델을 제안한다. 이는 제약된 메모리 환경에서 송수신 버퍼가 서로의 여유 공간을 공유하여 패킷의 손실을 최대한 줄일 수 있는 버퍼 모델이다. 또한 리스트와 비슷한 성능을 지니는 본 버퍼 모델은 자유 리스트를 사용한 버퍼와 달리 메모리 누수로 인한 버블(bubbles) 현상을 방지하므로 제한된 환경의 네트워크 버퍼에 적용할 수 있으며 배열을 사용하는 경우와 비교 할 때 최대 100% 성능 향상을 기대할 수 있다.

키워드 : 버퍼, 버퍼 관리, 자료구조, 네트워크 버퍼, 자원 효율성 증진

Abstract This paper proposes an Adaptive Double Buffer Model. As a new FIFO buffer model, this technique minimizes packet losses from network congestion by logically managing buffers. It allocates the spare spaces of non-congested buffers to congested buffers by allowing receive/send buffers to share two queues, and hence it minimizes packet losses. In contrast to the buffer model utilizing a free list, this buffer model can prevent the bubble phenomenon caused by a memory leak and thereby apply to a network buffer in a restricted environment. Also, compared with the model using an array, this model brings maximum 100 percent improvement in accepting packets and compared with the model utilizing a free list, this model has the similar efficiency. Results of the performance test on Adaptive Double Buffer Model, shows that this proposed model decreases packet losses and enhances memory efficiency.

Key words : Buffer, Buffer Management Data Structure, Network Buffer, Resource Efficiency Improvement

1. 서 론

컴퓨터 네트워크 사용이 급속히 확산되고 기술이 발전함에 따라 사용자가 시간과 장소에 구애 받지 않고 자유롭게 네트워크에 접속할 수 있는 환경이 도래하게 되었다. 이 같은 환경에서 사용자들은 보다 좋은 서비스 품질(Quality of Service: QoS)과 서비스를 끊김 없이 실 시간적으로 제공받기를 원하게 되었다. 이같이 처리해야 할 패킷이 폭증하고, 다양한 사용자의 서비스 요구를 지원하기 위해서 제한된 자원을 최대로 사용하기 위한 연구가 활발히 진행 중이다. 자원의 낭비 없이 사용하기 위해 수행된 많은 연구 중 하나로 네트워크 혼잡을 효과적으로 다루는 방법이 있다. 네트워크 혼잡이란 네트워크의 일부 혹은 전 요소의 링크에 유입된 트래픽 양이 링크의 용량을 초과하여 네트워크의 구성 요소가 이를 수용하지 못하고 패킷이 손실되는 현상을 말한다. 이 현상은 데이타의 재전송을 일으켜 서비스의 품질을 저하시키며 네트워크 자원을 낭비하는 결과를 초래한다[1].

이런 혼잡을 해결하기 위해 제시된 방법에는 혼잡 제어, 혼잡 회피 등이 있다. 하지만 네트워크의 불확실성으로 인하여 패킷이 폭주하게 되면 패킷이 손실되므로 라우터, 게이트웨이, 단말에서는 [2]에서 알 수 있듯이 트래픽의 완충 역할을 하는 버퍼를 사용한다. 네트워크에서 사용되는 버퍼는 그 특성상 용량을 크게 하면 패

† 학생회원 : 한국과학기술원 전산학과
asurazero@kaist.ac.kr
†† 종신회원 : 경희대학교 저자정보학부 교수
sylee@oslab.kyunghee.ac.kr
논문접수 : 2003년 11월 27일
심사완료 : 2006년 6월 30일

킷을 저장할 때 할당된 메모리 자원을 효율적으로 사용하지 못하게 되고, 용량을 작게 하면 가능한 네트워크 대역폭을 모두 사용하지 못하게 되므로 최적의 크기로 버퍼의 용량을 정하는 것이 중요하다[3]. 또한 버퍼는 선입후출(FIFO, First In First Out) 구조로 되어 있어 자료구조에 따라 메모리의 용량과 패킷이 저장될 때의 메모리 효율, 그리고 할당된 용량을 효율적으로 관리하는 알고리듬의 복잡도가 결정된다. 이러한 점에 착안하여, 본 논문에서는 제한된 환경에서 송수신 시에 버퍼가 최적의 용량과 동적으로 관리되는 동적 큐(active Queue, 이하 AQ)를 제시하고, 이를 사용한 적응형 이중 버퍼(adaptive Double Buffer, 이하 ADB) 모델을 제안한다.

제안하는 AQ는 빠른 접근 속도를 제공하고 메모리 낭비를 방지하기 위하여 배열을 사용하고, 그 용량을 동적으로 조절할 수 있게 한 자료구조이다. 이 큐는 메모리를 2 블록으로 구분 지어 2종류의 패킷이 존재할 수 있도록 한다. 그리고 ADB 모델은 하나의 버퍼가 두 개의 큐를 사용할 수 있도록 패킷의 입출력을 관리한다. 따라서 ADB 모델은 여유 공간이 없는 버퍼가 여유 공간이 많은 버퍼의 메모리를 사용할 수 있게 하여, 동적으로 큐의 크기를 결정하고 낭비되는 메모리의 비율을 줄이고, 폭주에 효율적으로 대응할 수 있다.

본 논문의 구성은 다음과 같다. 제 2장에서는 여러 알고리듬에서 사용하는 버퍼 모델의 특징을 비교, 분석하며, 제 3장에서는 메모리의 제약과 폭주를 대처할 수 있는 AQ를 사용한 ADB 모델을 제시한다. 제 4장에서는 제안된 모델에 대한 시뮬레이션을 통해 기존의 기법과 비교 분석한다. 마지막으로 제 5장에서는 결론을 다루며, 향후 수행되어야 할 연구 방향을 제시한다.

2. 관련 연구

네트워크 환경 중 통신에서 발생하는 75%의 혼잡은 TCP와 IP를 통하여 제어된다[4]. 이러한 TCP/IP의 혼잡제어는 기본적으로 버퍼를 사용하여 혼잡을 관리하므로 버퍼의 구조, 버퍼의 관리 방법에 따라 성능 향상을 가져올 수 있어 다양한 버퍼 관리기법이 연구되었다. 그 연구 대상 중 하나가 버퍼의 공유, 즉 큐를 공유하는 것이며, 이를 통하여 성능의 향상을 가져올 수 있음이 연구 결과로 발표되었다[5,6]. 따라서 이번 장에서는 본 연구와 관련된 버퍼 공유 기법과 버퍼 구조에 대한 기존 연구에 대해 살펴 보겠다.

기존에 연구된 버퍼 공유 기법은 Complete Partitioning(CP), Complete Sharing(CS), Sharing with a maximum queue lengths(SMXQ), Sharing with a minimum allocation(SMA) 방법이 있다. CS는 메모리 공간 전부를 모든 버퍼들이 공유하는 것을 말한다. CS는 각 포트 별 버퍼의 메모리를 공유하여 패킷의 폭주 현상 시 처리량을 향상시키지만 한 버퍼에 대량의 패킷이 유입된 후 다른 버퍼 역시 대량의 패킷이 유입되면 후자의 버퍼는 할당 받을 메모리 공간이 없어 버퍼의 처리율이 떨어지는 문제가 있다. SMXQ는 각 버퍼 용량의 최대값을 설정하는 기법이지만 버퍼의 최대값을 정확히 예측할 수 없는 문제와 버퍼의 최댓값을 설정하였다고 하더라도, 최대값보다 많은 패킷이 유입되면 여유공간을 사용하지 못한다. SMXQ와는 반대로 SMA는 각 버퍼 마다 최소의 공간을 할당하여 나머지 공간을 공유하는 방법으로 메모리 공간의 낭비를 최소화하며 처리량을 극대화하는 방법이다[7]. ADB 모델은 SMA 공유 기법을 사용하여 최소한의 처리량을 보장하며 메모리 공유를 통해 폭주에 유연히 대처한다.

버퍼 공유를 위한 자료 구조에 관하여 자유 리스트와 배열을 사용한 연구가 있다. 자유 리스트 구조는 버퍼로 사용할 메모리를 일정 단위로 나누어 버퍼가 메모리를 필요로 할 경우 자유 리스트의 노드를 가져다 사용하고, 사용이 끝나면 다시 자유 리스트로 반환하여 버퍼의 용량을 동적으로 관리한다[8]. 하지만 버블(bubbles)라 불리는 메모리 누출 현상으로 필요한 버퍼에 메모리를 할당할 수 없을 수 있으며, 이 현상은 메모리 공간이 부족하여 패킷이 손실되었을 때의 상황과 유사하다[9]. 따라서 패킷을 저장하는 도중 이 현상을 감지하기 어려우며 이를 복구 하는 것도 어렵기 때문에 메모리 누출을 확인하고 복구하는 작업을 하는 과정에서 막대한 부하가 발생한다[10].

배열을 사용한 버퍼 공유 기법은 분산 컴퓨팅 분야에서 연구되었으며 스택과 큐보다 일반적인 구조인 덱(deque: double ended queue)을 배열로 구현함으로써 프로세스간 메모리를 공유한다[deque]. 네트워크 분야에서 버퍼는 연속된 버퍼 공간에 패킷을 저장하므로 이들의 용량을 동적으로 관리하기 위해 [DCAS]과 같이 배열을 사용한 덱을 이용하여 메모리를 공유하는 방식을 바로 적용하기 어렵다. 하지만 배열은 패킷을 저장하는 공간 외에 필요한 다른 정보를 최소한으로 할 수 있어 버퍼를 동적으로 관리할 수 있다고 가정한다면 메모리를 최적으로 사용할 수 있어 자원이 제약된 환경에서 문제없이 적용할 수 있는 장점과 리스트와 같이 메모리 누출 현상이 일어나지 않으므로 메모리 누출에 대한 문제가 없고, 이를 처리해야 하는 루틴 역시 필요 없는 장점이 있다.

비 동기 전송 모드(Asynchronous Transfer Mode: ATM)에서 버퍼 공유에 관한 연구로는 셀(cell)의 과잉을 막는 우선 순위 제어 방법이 있다[13]. 이러한 우선

순위 방법에는 부분 버퍼 공유 기법과 적응적 분할 버퍼 공유 기법이 있으며 특히 분할 버퍼 공유 기법은 각 큐에 경계 값을 설정하여 그 경계 값과 도착한 셀의 우선 순위에 따라 큐에 셀을 저장한다. 이 방법은 고 순위 셀이 손실되지 않도록 보장하기 때문에 버퍼에서 고 순위의 셀을 얼마만큼 보장할 것인가 결정하는 것에 있어 자원이 제약된 상황에 적용하기에 무리가 있다. 그 밖에 이를 개선한 가변 고유 방식에서는 여유 공간을 사용하고 우선 순위가 다른 셀들을 묶어 셀을 저장함으로써 버퍼 공유를 가능하게 한다[14]. 이는 여유 공간을 사용하여 셀을 큐에 저장하는 데 이를 결정하는 과정에서 부하가 발생하며 송 수신 버퍼의 공유를 고려하지 않았기 때문에 상대적으로 메모리의 사용률이 떨어진다. 게다가 라우터, ATM에서 사용되는 버퍼는 제약된 환경을 가정하지 않았기 때문에 구현의 편리성으로 리스트를 사용하여 버퍼를 관리한다. 앞에서 언급하였듯이 리스트로 관리되는 버퍼는 메모리 누수 현상이 발생할 수 있으며 누수 현상이 발생하면 메모리가 낭비되는 비율이 점차 증가하여 패킷이 계속 손실된다. 따라서 자원이 부족한 환경에서 메모리 누수 및 낭비 없이 버퍼간 용량을 조절하며 메모리를 공유할 수 있는 버퍼 모델이 필요하다.

3. 적응적인 이중 버퍼 모델

본 논문에서 제안하는 ADB 모델은 제약된 환경에서 버퍼가 메모리를 최대한 사용하기 위하여 그림 1에서처럼 두 개의 버퍼가 하나의 메모리 블록을 공유할 수 있도록 한다.

따라서 어느 한 버퍼가 폭주하게 되더라도 다른 큐에 여유 공간이 있으면 경계를 이동하고 여유 공간을 사용할 수 있다. ADB 모델은 AQ에 의해서 분리된 영역을 연결하여 버퍼가 일관된 순서가 되도록 유지하고 내부적으로 AQ는 패킷의 유입량에 따라 경계를 이동시키며 버퍼의 용량을 동적으로 관리한다. 이 후 ADB 모델에서 사용되는 알고리듬을 설명하기 위해 표 1의 용어를 정의한다.

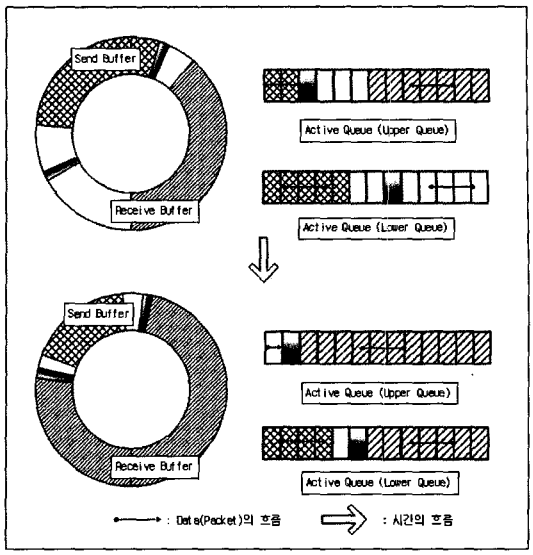

그림 1 기존 모델과 ADB 모델과의 비교

ADB 모델은 그림 2에서처럼 초기화 알고리듬, 데이타 입력 알고리듬, 데이타 처리 알고리듬, 입력 위치 변경 알고리듬, 처리 위치 변경 알고리듬으로 구성된다. 초기화 알고리듬은 통신이 시작되기 전 메모리를 할당

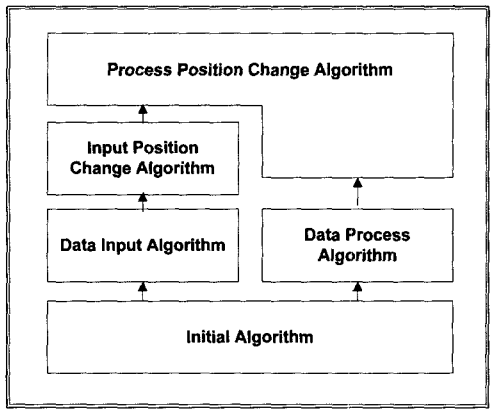

그림 2 ADB 모델의 알고리듬 구성

표 1 ADB 모델을 위해 정의한 용어

이 름	내 용
Upper Queue (UQ)	두 개의 AQ 중 한 개의 큐를 가리킴
Lower Queue (LQ)	두 개의 AQ 중 다른 한 개의 큐를 가리킴
Usa (Upper Queue Start address)	UQ의 송수신 버퍼의 시작 위치
Lsa (Lower Queue Start address)	LQ의 송수신 버퍼의 시작 위치
Ub (Upper Queue Border)	UQ의 송수신 버퍼 간 경계를 나타낸다.
Lb (Lower Queue Border)	LQ의 송수신 버퍼 간 경계를 나타낸다.
Ip(Input position)	패킷의 입력 위치를 나타낸다.
Pp(Process position)	패킷이 처리될 위치를 나타낸다.

할 때 호출되며 이후 통신이 시작되면 데이타 입력 알고리듬과 데이타 처리 알고리듬이 호출되고 각각의 알고리듬은 위치 변경 알고리듬을 통하여 버퍼의 흐름을 제어한다. 또한 송신, 수신 버퍼는 서로 대칭되므로 송신 버퍼를 기준으로 이야기하기로 한다.

3.1 초기화 알고리듬

초기화 알고리듬은 메모리를 할당하여 각 메모리 블록에 AQ를 할당하여 UQ, LQ로 구분하고 각 버퍼의 경계(Ub, Lb), 각 버퍼의 Ip, Pp를 설정하는 과정이다. 이 설정을 통해 송신 버퍼의 Ip, Pp, 수신 버퍼의 Ip, Pp의 위치 포인터가 증가될 것인지, 감소 될 것인지가 결정된다.

3.2 데이타 입력 알고리듬

데이타 입력 알고리듬은 AQ에 패킷이 들어 올 때 이를 처리하는 알고리듬이다. 그림 3은 데이타 입력 알고리듬의 시간 경과에 따른 입력 위치의 변화를 보여주며 그림 4는 데이타 입력 알고리듬의 흐름도이다. 데이타 입력 알고리듬은 큐에 패킷을 저장하므로 (line04)에서처럼 Ip의 위치에 패킷이 없다면 큐에 패킷을 저장하고, 패킷이 들어 있다면 (line06)에서처럼 패킷을 폐기시키고 Ip의 위치를 변화시키지 않는다. 만약 패킷이 정상적으로 저장되면 초기화 알고리듬에서 설정한 결과대로 송수신 버퍼의 Ip를 증가 또는 감소시킨다.

만일 Ip가 Ub/Lb의 위치와 동일한 경우 Ip를 바꾸거나 계속 유지해야 하기 위해 (line 02)에서 위치 변경 알고리듬을 호출하여 다음에 패킷이 입력될 위치를 결정한다.

3.3 데이타 처리 알고리듬

데이타 처리 알고리듬은 그림 5에서처럼 그림 6의 과정에 따라 패킷의 처리를 담당하는 알고리듬이다. 이 알고리듬은 데이타를 처리할 지 결정하는 과정에서 (line03)과 같이 패킷이 있으면 패킷을 처리하고 Pp를 증가/감소시킴으로써 패킷의 흐름이 어긋나지 않게 할 수 있다. 만약 처리할 패킷이 없다면 Pp의 위치를 움직이지 않고 패킷이 입력될 때까지 기다린다. 이 알고리듬 역시 Pp가 Ub/Lb의 위치와 같을 경우 (line02)에서 위치 변경 알고리듬을 호출하여 다음에 소비될 위치를 결정한다.

3.4 위치 변경 알고리듬

위치 변경 알고리듬은 데이타 입력 알고리듬과 데이

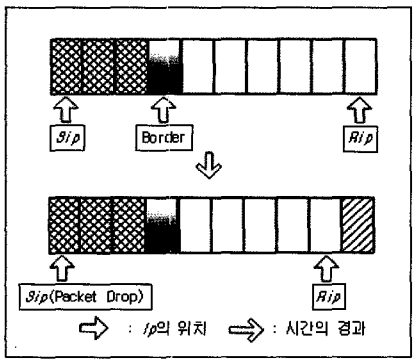

그림 3 데이타 입력 시 포인터 위치 변화

```
Parameter : None
01: If Ip= Ub/Lb then
02:     Call (Position Change Algorithm)
03: If Ip = Empty) then
04:         Input Data at Ip
05:         Move Ip to Next Position
06: Else then
07:         Drop Packet
08: return.
```

그림 4 데이타 입력 알고리듬 의사코드

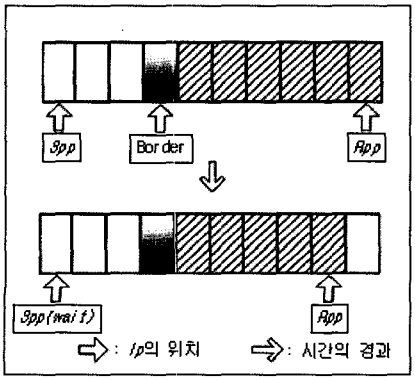

그림 5 데이타 처리 시 포인터 위치 변화

```
Parameter : None
01: If Pp = Ub/Lb then
02:     Call ( Position Change Algorithm)
03: If Pp ≠ Empty then
04:     Process Data at Pp
05:     Move Pp to Next Position
06: Else then
07:     return
```

그림 6 데이타 처리 알고리듬의 의사코드

타 처리 알고리듬에서 버퍼의 Ip 또는 Pp의 위치가 Ub/Lb와 동일한 경우 이를 처리하는 알고리듬이다. 이를 위해 위치 변경 알고리듬은 입력과 처리의 경우를 나누어, 각각 경계를 기준으로 다음 위치에 패킷의 유무를 판단하고 Ub/Lb의 위치를 이동시킬 것인지 Sip/ Rip의 위치를 바꿀 지를 결정한다. 이는 송수신 버퍼 각각에 해당되며 포인터의 위치 변경 시 UQ에서 수행 중이면 LQ를 LQ이면 그 반대를 처리한다. 다음은 데이타 입력과 처리 시의 위치 변경 알고리듬의 흐름의 설명이다.

3.4.1 데이타 입력

데이타 입력 시에 호출되는 위치 변경 알고리듬은 Ip가 Ub/Lb의 위치에 도달했을 때 큐의 용량을 증가시킬 것인지 또는 Ip를 다른 큐의 시작위치로 옮길 지를 결정한다. 따라서 위치 변경 알고리듬은 그림 7의 과정을 거쳐 AQ에서 패킷이 서로 섞이지 않도록 한다. 이 때 위치 변경 알고리듬은 데이타 입력 알고리듬에서 호출될 때의 정보를 이용해 자신이 속한 AQ가 어떤 것인지를 알고 (line02-05)에서처럼 Ub/Lb의 다음 위치가 비어 있을 때 경계 위치에 다른 버퍼의 Ip, Pp가 존재하는 지 확인한다. 만약 경계 위치에 상대방 버퍼의 Ip와 Pp가 존재하면 이를 다른 AQ의 시작위치로 이동시킨다. 그 후 (line06)에서 처리의 위치 변경 알고리듬을 호출하여 현재 수행되고 있는 버퍼의 Pp를 이동할 것인지를 판단하고, Pp의 위치가 결정되면, (line07)에서 Ub/Lb를 이동시키고 자신을 호출한 데이타 입력 알고리듬으로 반환된다. 만약 다음 위치가 비어있지 않으면, (line10)에서처럼 입력 위치를 버퍼의 시작위치로 이동한다. 이와 같은 과정을 통해 데이타 입력 알고리듬은 입력에 대한 위치 변경 알고리듬을 통해 데이타의 입력 시 큐의 용량을 적응적으로 조절한다.

3.4.2 데이타 처리

처리에 대한 위치 변경 알고리듬에서는 패킷의 처리와 입력 순서에 대한 고려가 이루어져야 한다. 그림 8에서 확인할 수 있듯이 처리에 대한 위치 변경 알고리듬은 (line 01)에서 자신이 속한 AQ를 가르쳐 주는 인자 값을 받아 자신이 속한 버퍼의 Usa/Lsa를 확인하여 소비할 패킷이 있는지 확인한다. 버퍼에서의 Pp와 Ip의 관계는 항상 Ip가 Pp보다 앞에 있거나 같은 위치에 존재 하므로, 만약 Pp가 경계 위치에 있을 때 Usa/Lsa가 비어있다면 Ip가 Pp와 같은 위치에 존재하는 지를 검사한다. 만약 같은 위치에 존재하면 속한 AQ의 용량이 커져 패킷을 저장할 수도 있으므로 Pp의 위치를 옮기지 않는다. 하지만 Usa/Lsa가 비어 있는 데 Ip가 Pp와 같은 위치에 존재하지 않으면 입력의 위치 변경 알고리듬에서 Ip를 옮긴 것이기 때문에 (line02~03)와 같이 Pp를 Usa/Lsa로 옮긴다. 만약 Usa/Lsa가 비어있지 않다면, 이는 다음으로 처리해야 할 패킷이 Usa/Lsa에 존재하는 것을 뜻하기 때문에, (line 04~05)에서 Pp의 위치를 Usa/Lsa로 이동시킨다.

```
Parameter :  AQ Type (UQ or LQ)
01:     If Usa/ Lsa = Empty then
02:     If Ub/ Lb = IP
03:         return
04:     Change Pp to different buffer's Usa/Lsa
05:     return
```

그림 8 처리 위치 변경 알고리듬의 의사 코드

```
Parameter : AQ Type (UQ or LQ)
01:   If Next Position of Ip = Empty && Next !=  Usa/Lsa then
02:     If different buffer's Ip = Ub/Lb then
03:       Change different buffer's Ip to Other AQ's Send Start Position
04:         If different buffer's Pp = Ub/Lb then
05:             Call  (different buffer's Process Position Change Algorithm
06:         Call current buffer's Process Position Change Algorithm
07:             Move Border to Next Position.
08:             return
09:     Else then
10:             Change Ip to different buffer's Usa/Lsa
11:     return
```

그림 7 입력 위치 변경 알고리듬의 의사코드

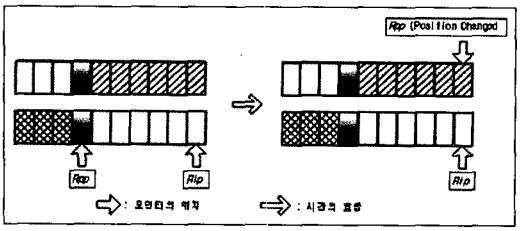

그림 9 처리 위치가 바뀌는 경우

4. Adaptive Double Buffer 모델의 성능 평가

FIFO 버퍼의 구성을 리스트 또는 배열을 사용할 것인지에 따라 버퍼의 용량과 관리 기법이 바뀌기 때문에 제한된 환경에서의 성능평가를 위해 리스트와 배열 구성 요소로 하는 버퍼와 ADB 모델의 성능을 서로 비교한다. 그리고 그림 10과 같이 네트워크가 구성되어 있고 자원이 제약되어 있는 환경이라 가정하고, 리스트를 사용한 버퍼의 경우 버퍼의 용량을 동적으로 결정할 수 있으므로 ADB 모델의 특성과 유사해 이론적 성능 평가에서는 이를 제외하였다.

이 때 노드 A_1, B_1, C_1은 Router B에 연결되어 있는 네트워크의 노드들과 통신하려면 라우터 A를 경유해야 하며, 각 노드 A_1, B_1, C_1은 각각 다른 노드 A_1, B_1, C_1이 보내는 패킷의 양을 모르기 때문에 A_2, B_2, C_2의 요청에 따라 패킷을 전송한다. 따라서 노드 A_1, B_1, C_1는 라우터 A의 용량을 고려하지 않고 패킷을 전송하기 때문에 라우터 A에서 혼잡이 발생하며, 이를 처리하기 위해 라우터 A는 버퍼를 사용하여 도착한 패킷을 임시로 저장한다. 이 때 A_1, B_1, C_1에서 전송하는 패킷의 양을 동일하게 하여 라우터 A역할을 하는 컴퓨터에서 사용되는 버퍼의 용량과 구성요소를 변경할 수 있는 시뮬레이션 프로그램을 작성하였다. 이는 Windows 2000 Professional 운영체제 환경에서 유입되는 패킷의 양을 통제하여 실험을 수행하였다.

4.1 패킷의 손실 관점

그림 11에서 확인할 수 있듯이 리스트와 ADB 모델을 사용한 버퍼가 배열을 사용한 버퍼보다 패킷의 폭주 현상을 효율적으로 처리하였다. 이는 배열로 구성된 버퍼가 패킷을 폐기하고 있는 동안에 ADB 모델과 리스트를 사용한 버퍼가 그 용량을 동적으로 변화시켜 패킷을 수용하였고, 이것이 패킷의 재전송을 줄이게 되었다. 그리고 리스트가 ADB 모델보다 패킷을 많이 수용한 것은 현재의 ADB 모델이 내부적으로 사용하는 경계를 사용하지 않았기 때문에 시간에 지남에 따라 패킷이 손실되었기 때문이다. 특별히 패킷이 폭주할 때 배열을 사용하는 것보다 ADB 모델을 사용하는 것이 패킷이 손실되는

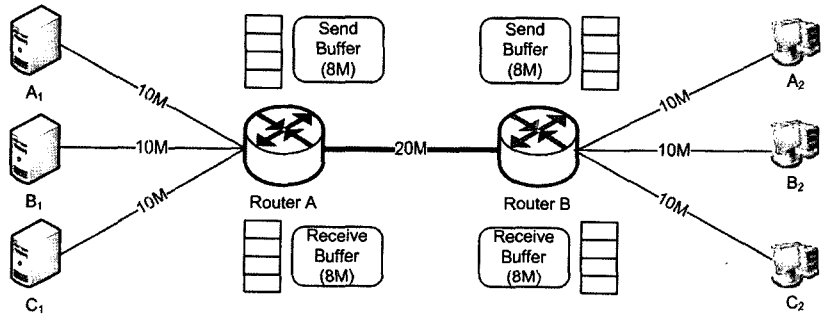

그림 10 성능평가를 위한 네트워크 모델

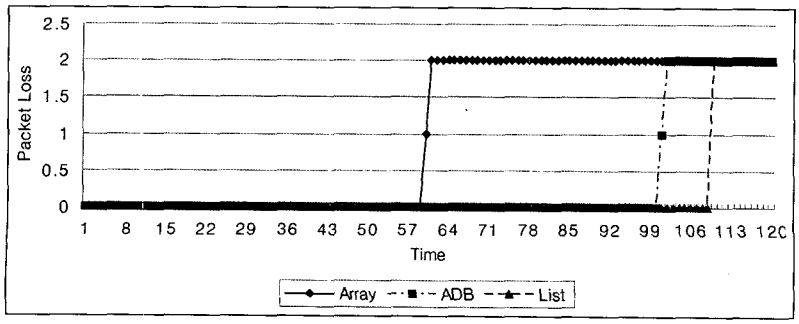

그림 11 큐의 용량이 100인 경우

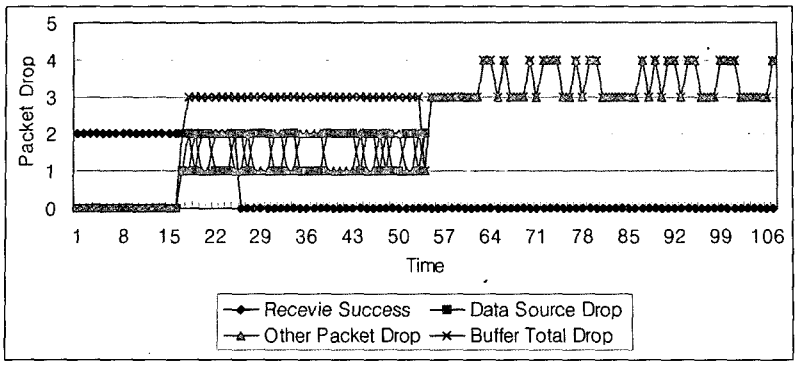

그림 12 일반적인 FIFO Queue의 패킷 폐기

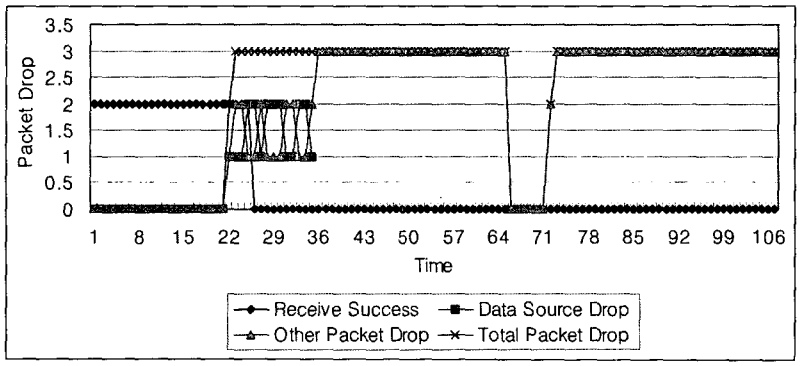

그림 13 ADB를 사용하는 버퍼에서의 패킷 폐기

시간을 배열을 사용한 때보다 2배 정도 늦춘다. 또한 리스트와 ADB 모델간의 패킷 손실이 되는 시간의 차이가 크게 벌어지지 않았다.

위의 그림 12, 그림 13은 패킷의 재전송을 고려했을 때의 시뮬레이션 결과를 나타낸 것이다. 위 그림의 Receive Success는 시간에 따른 데이타 자원으로부터 받은 패킷을 나타낸다. 자료를 분석해 보면 한 단위마다 2개의 패킷을 입력 받아 그것이 수신 버퍼에 저장되며 수신 버퍼의 내용이 처리되어 송신 버퍼로 들어가고, 그 밖의 다른 패킷이 동시에 들어간다. 이를 한 단위에 2번 반복하게 되며, 시간이 경과하게 되면 패킷의 폐기가 발생하게 된다. 또한 Data Resource Drop은 송신 버퍼에서 사용자에게 제공하려는 서비스 패킷의 폐기를 나타내며, Other Packet Drop은 사용자에게 보내는 다른 종류의 패킷을 나타낸다. Total Packet Drop은 패킷의 총 폐기양을 나타낸다. 자료를 분석해 보면 일반적인 FIFO 버퍼의 경우 54초 때 서비스 패킷의 전송이 완료되고, ADB의 경우 36초에 서비스 패킷의 전송이 완료된다. 이는 ADB의 유휴 공간을 사용하는 특성으로 인하여 패킷의 폐기가 일어나지 않아 일반적인 FIFO 버퍼보다 더 빠른 시간 내에 서비스 패킷을 전송한 것을 알 수 있다. 또한 서비스 패킷을 모두 전송한 후 다른 종류의 패킷을 보내는 과정에서 ADB의 경우 패킷의 폐기가 일어나지 않는 구간이 있다(66-71). 이는 ADB에서 수신 버퍼에 패킷이 들어 있을 때 밀려난 영역을 송신 버퍼가 다시 사용하게 되어 패킷을 수용한 것이다.

4.2 패킷 저장 시의 메모리 효율 관점

ADB와 배열, 자유 리스트를 사용한 버퍼의 메모리 효율을 계산하기 위해 패킷 하나당 크기가 53byte로 하였고, 최대 버퍼로 할당되는 최대의 메모리를 10600byte로 하여 패킷을 송수신 버퍼 각각 100개 저장시킬 수 있도록 모의 실험을 수행하였으며 자유 리스트의 경우 발생할 수 있는 메모리 누출은 고려하지 않았다.

그림 14를 분석해 보면 한 버퍼에서 총 100개의 패킷을 저장할 수 있다고 가정하였을 때 배열의 구조로 버퍼를 할당하는 경우 송신 버퍼로 할당한 버퍼의 용량만큼 패킷을 수용할 수 있다. 즉, 53*100인 5,300 byte 의 메모리를 사용하여 패킷을 수용하였고, 자유 리스트의 경우 버퍼 간 메모리 공유를 통해 최대 10545 byte만큼 메모리를 사용하였다. ADB의 경우 메모리를 공유하여 10600 byte 모두 사용할 수 있었으며 자유 리스트와 비슷한 양상으로 메모리 사용률이 증가하였으나 자유 리

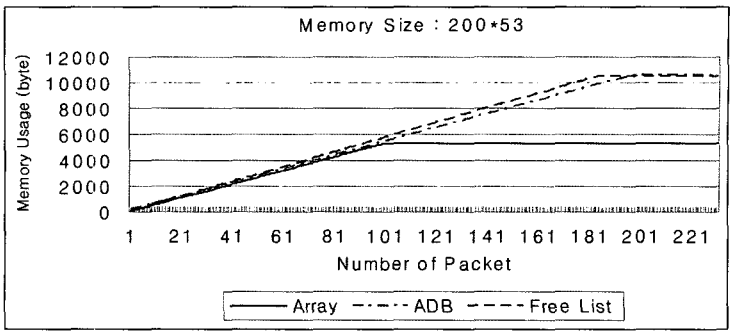

그림 14 버퍼로 할당되는 메모리의 크기

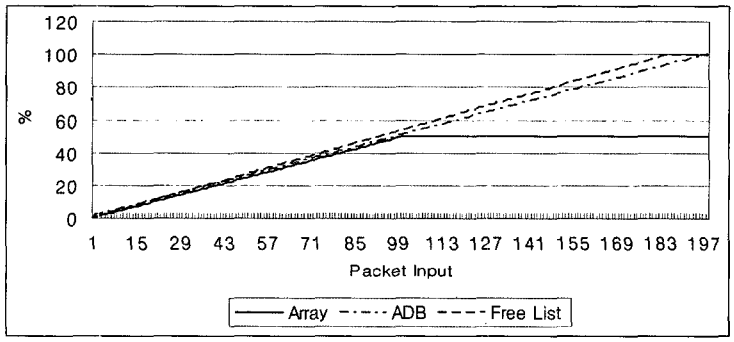

그림 15 메모리의 효율

스트보다 많은 패킷을 수용하여 할당된 10600 byte 메모리 전부를 사용함을 확인할 수 있다.

그림 15는 배열과 ADB 모델, 자유 리스트의 메모리 효율을 나타낸 그래프이다. 배열의 메모리 효율은 버퍼의 정적인 특성으로 인하여 초기에 할당한 메모리 만을 사용하여 50%의 사용률을 보여 주고 있다. 또한 자유 리스트는 리스트 구조를 유지하기 위한 포인터 용량으로 인하여 메모리를 모두 사용하는 시점이 ADB 모델보다 빠르므로 자유 리스트와 배열과 비교하여 ADB 모델이 메모리 효율 측면에서 좋다.

4.3 토의

패킷의 크기가 가변적인 경우 리스트는 노드를 재활용하기 위하여 노드의 크기를 패킷의 크기에 맞게 할당하지 않는다. 따라서 리스트, 배열 모두 패킷의 크기가 가변적일 때는 노드의 용량보다 큰 패킷이 도착하였을 경우에는 cluster라고 하는 extern-buffer를 사용하여 저장한다[8]. 따라서 패킷의 용량이 가변적인 환경에서도 ADB 역시 외부 버퍼에 저장하는 방식으로 구현하면 리스트와 동일한 성능을 나타낼 수 있다.

패킷의 크기가 고정적인 경우 메모리 절약 정도에 대하여 리스트와 ADB를 비교하면 다음과 같은 결론을 얻을 수 있다. 패킷의 크기를 100byte라 가정하면 리스트의 경우 최소한 104byte의 용량이 필요하고, ADB의 경우 100byte의 용량과 추가적으로 경계를 나타내는 용량이 사용된다. 만약 버퍼로 사용할 수 있는 메모리 공간이 5 Mbyte라 한다면 리스트의 경우 총 5096 byte 만큼 사용할 수 있으며 포인터로 사용되는 메모리는 총 196 byte이다. ADB의 경우 5024 byte 만큼의 용량을 사용할 수 있고, 경계를 고려하지 않는다면 총 5000 byte를 패킷을 수용하고, 경계 위치로 사용한 공간까지를 계산하면 리스트의 경우와 동일한 성능을 가진다. 즉, 리스트는 메모리를 총 95.7% 사용하였고, ADB의 경우 경계를 고려하지 않는 경우 98.1%, 고려하는 경우 리스트와 동일한 성능을 가진다. 이는 패킷의 크기와 버퍼의 용량에 따라 결정되며 특히 리스트와 ADB와 비교할 때 메모리의 용량을 고정시키면 패킷의 크기가 작은 경우 메모리 효율이 증가함을 알 수 있다. 게다가 자유 리스트의 경우 노드의 자료구조를 데이타와 포인터 주소를 저장한 공간만 가지고 있는 것으로 가정하였기 때문에 자유리스트를 관리하는 알고리듬에 따라 패킷을 저장할 수 없는 공간이 증가할 수 있다[8]. 또한 배열은 버퍼에 메모리를 할당하면 그 용량을 변화시킬 수 없으므로 ADB는 배열을 사용한 버퍼보다 성능이 뛰어나다.

한편, 복잡도 측면에서 배열을 사용하여 버퍼를 구성하는 경우 버퍼는 환형 배열을 사용하므로 패킷을 저장하거나 처리하는 루틴에서 배열의 마지막 위치를 가리

키고 있을 때 포인터가 배열의 처음 위치로 이동되고 입력 또는 처리할 위치에 패킷이 존재하는 지 확인하는 과정까지 포함하여 총 2번의 검사를 거치며 이는 상수 시간에 검사할 수 있어 시간 복잡도는 O(1)이다. 자유 리스트의 경우 패킷 하나에 대하여 자유 리스트에 노드가 있는 지 확인하는 과정과 자유리스트에 노드가 존재하지 않을 때 메모리에서 자유 리스트 노드를 할당하는 과정이 필요하며 버퍼에 노드를 할당하는 과정, 버퍼에 할당된 노드를 처리하는 필요하다. 이 때 메모리 누수 현상을 탐지하는 과정, 메모리 누수를 복구하는 과정, 메모리를 할당하는 과정을 고려하지 않으면 각 리스트를 확인하고 노드를 할당하는 것은 상수 시간 안에 수행 가능하므로 시간 복잡도는 O(1)이다. ADB 모델을 버퍼로 사용하는 경우는 패킷 하나가 저장되는 데 최악의 경우 8번의 검사를 거친다. 이 역시 상수 시간 안에 수행 가능하므로 시간 복잡도는 최악의 경우 O(8)로 O(1)이다. 즉, 자유 리스트 내의 메모리 할당 및 해제 루틴, 메모리 누수 시의 복구 작업을 고려하면 ADB 모델은 배열보다 더 많은 포인터 검사로 인해 많은 CPU 사이클을 사용하게 되지만 상수 시간 안에 검사가 완료되며 패킷을 저장할 때 메모리 누수 없이 패킷의 손실을 줄일 수 있다.

5. 결 론

일반적으로 네트워크 환경에서 버퍼의 구조를 결정할 때에는 버퍼의 용량을 동적으로 관리할 수 있는지 고려해야 한다. 만약 버퍼의 용량을 동적으로 관리할 수 없다면, 할당한 메모리 보다 많은 패킷이 들어 왔을 때 패킷이 손실되며, 할당한 메모리 보다 적은 패킷이 들어올 경우 메모리 공간이 낭비된다. 이런 사실을 고려하여 버퍼의 자료구조가 결정되어야 하며 본 논문에서는 어느 한 버퍼가 폭주하여도 다른 버퍼의 여유 공간을 폭주한 버퍼에 할당할 수 있는 ADB 모델을 제안했다. 이는 송수신 버퍼를 논리적으로 관리함으로써, 송신, 수신 버퍼가 두 개의 큐를 공유하여 버퍼에서 낭비되는 메모리의 비율을 줄이고 패킷을 저장할 때 메모리 사용 효율성을 증진시키고 패킷의 손실률을 줄인다. 게다가 배열의 특성을 지녀 메모리의 누출 없이 최적의 용량으로 메모리를 사용할 수 있으며, 리스트의 특성을 가지고 있어 동적으로 버퍼의 용량을 조절할 수 있다. 더욱이 버퍼에 유입되는 패킷의 양에 따라 경계가 결정되어 패킷이 많이 도착한 버퍼에 여유 공간을 할당할 수 있게 함으로써 여러 혼잡제어 알고리듬을 쉽게 적용시킬 수 있는 장점을 지닌다. 이러한 장점으로 ADB 모델은 배열보다 최대 100%의 성능향상을 가져올 수 있으며, 리스트를 사용했을 때와의 거의 유사한 성능을 지녀 유비쿼터스와 같은 제한된 환경에 적용하기 적합하다.

한편, 이미 점유한 버퍼에 대하여 다른 버퍼가 이미 점유한 버퍼의 유휴공간을 사용하지 못하는 경우가 발생하여 이를 회복하는 회복시간을 최소화하는 연구와 다른 혼잡제어 기법과 ADB 모델과의 연동에 관한 구체적인 연구가 요구된다.

참 고 문 헌

[1] S. Floyd. et el, "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309, April. 1998.

[2] W. Leland, M. Taqqu, W. Willinger, D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," IEEE/ACM Transactions on Networking, 2(1), pp. 1-15, February 1994.

[3] Amit Cohen, Reuven Cohen, "A Dynamic Approach for Efficient TCP Buffer Allocation," IEEE Transaction on Computers Vol.51. No.3, March 2002.

[4] Besson, E., "Performance of TCP in a wide-area network: influence of successive bottlenecks and exogenous traffic," *GLOBECOM '00*. IEEE, Volume: 3, pp. 1798-1804, December, 2000.

[5] P. Gonet, J. P. Coudreuse, and M. Servel, "Implementing asynchronous transfer mode concepts: Main results of the prelude experiment," in Proc. IEEE GLOBECOM, pp.1871-1875, November, 1987.

[6] M. G. Hluchyj and M. J. Karol, "Queueing in high-performance packet switching," IEEE J. Select. Areas Commun., vol. 6, pp. 1587-1597, December. 1988.

[7] Choudhury, A.K., Hahne, E.L. "Dynamic queue length thresholds for shared-memory packet switches," Networking, IEEE/ACM Transactions on, Volume: 6, Issue: 2, pp. 130-140, April 1998.

[8] G.R. Wright, W.R. Stevens, "TCP/IP Illustrated: The Implementation," vol.2, Addison Wesley, 1995.

[9] Yu-Sheng Lin, Shung C.B., "Queue management for shared buffer and shared multi-buffer ATM switches," INFOCOM '96. Proceedings IEEE, Vol. 2, pp. 688-695, March 1996.

[12] J.J. Bae and T. Suda., "Survey of Traffic Control Schemes and Protocols in ATM Networks," Proceedings of the IEEE. Vol. 79. No.2 pp.170-189, 1991.

[10] Cisco Tech Note, Cisco - Troubleshooting Buffer Leaks, http://www.cisco.com/warp/public/63/bufferleak_troubleshooting.pdf.

[11] 이영교, 안정희, "셀 손실 QoS향상을 위한 큐 구조에 관한 연구", 컴퓨터산업교육기술학회 논문, VOL. 03 NO.01 pp. 0019~0026, 2002. 01.

[12] 이정찬, 최창범, 이승룡, "모바일 인터넷 환경을 위한 적응형 멀티미디어 버퍼관리 기법", 한국통신학회 2003 하계 학술 발표 초록집, p.45, 2003.

최 창 범

2005년 경희대학교 전자정보 공학부 졸업(학사). 2005년~현재 한국 과학 기술원 대학교. 전자 전산학과 전산학 석사 재학 중. 관심분야는 컴퓨터 네트워크, 그래프 이론, 소프트웨어 공학

이 승 룡

1978년 고려 대학교 재료 공학과 공학사 1986년 Illinois Institute of Technology 전산학과 석사. 1991년 Illinois Institute of Technology 전산학과 박사. 1992년~1993년 Governors State University, Illinois 조교수. 1993년~현재 경희대학교 전자 정보학부 교수. 관심분야는 내장형 시스템, 실시간 시스템, 미들웨어 시스템, 멀티미디어 시스템