리눅스 기반의 QoS 지원 기술

QoS Implementation over Linux

박진(J. Park) 통신프로토콜표준연구팀 계약직연구원
강신각(S.G. Kang) 통신프로토콜표준연구팀 책임연구원, 팀장

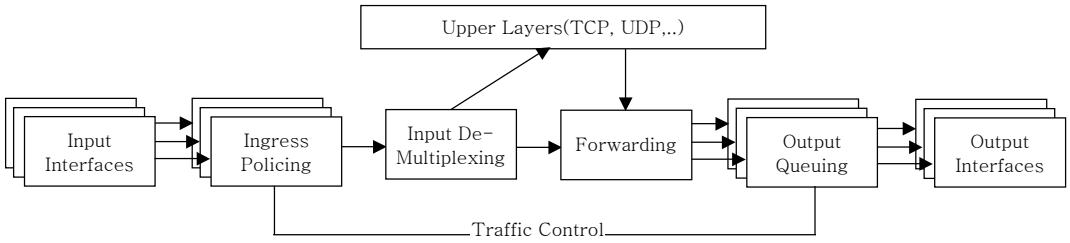
인터넷 QoS 이슈는 차세대 인터넷 핵심기술 중의 하나로서 그 동안 많은 연구가 이루어져 왔다. 실제로 QoS를 구현하고 실험하기 위해, 많은 개발자들이 가장 일반적인 형태의 개발환경으로서 리눅스(Linux)를 사용해 왔다. 특히, 리눅스 커널 버전 2.2 이상부터 TC 기능이 지원되면서 개발이 더욱 용이해졌다. 본 고에서는 QoS 개발자들을 위해 리눅스 커널 2.4 기반 TC 메커니즘과 2.4 커널 메인 스트림에 새로이 통합된 DiffServ 메커니즘에 대해 살펴본다.

I. 서 론

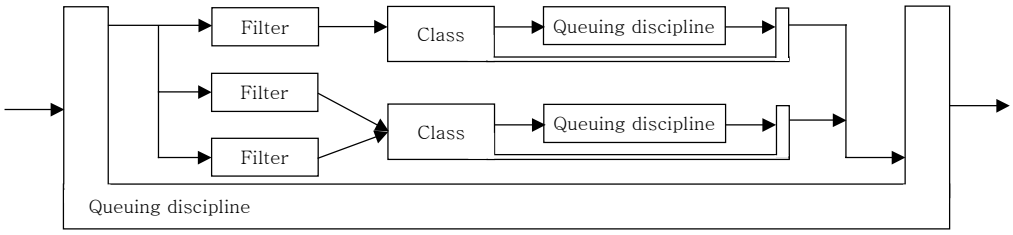
인터넷 QoS 이슈는 차세대 인터넷 핵심기술 중의 하나로서 그 동안 많은 연구가 이루어져 왔다. 리눅스는 QoS를 구현하고 실험하는 많은 개발자들로부터 가장 일반적인 형태의 개발환경으로서 사용되어 왔다. 특히, 커널 버전 2.2 이상부터 TC(Traffic Control) 기능이 지원되면서 개발이 더욱 용이해졌다. 본 고에서는 먼저 QoS 메커니즘 개발에 빌딩블록을 제공하는 리눅스(커널 버전 2.4) 커널의 TC 메커니즘과 이를 통해 구현되어 커널 2.4에 새로이 메인스트림으로 추가된 DiffServ 메커니즘에 대해 알아본다.

II. 리눅스 트래픽 컨트롤

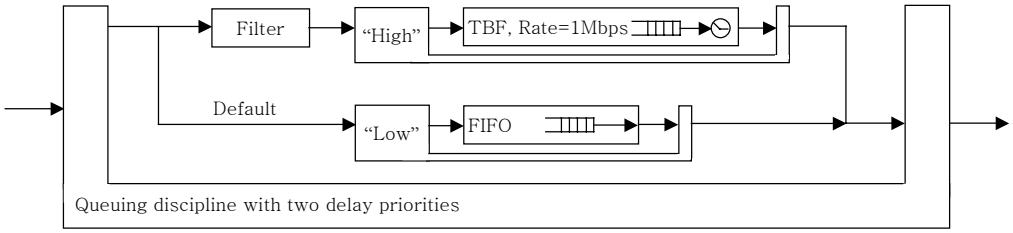
본 절에서 리눅스 기반의 QoS 제공 기술로서 리눅스 커널 버전 2.2(2.1.90) 이상에서 지원되는 TC[1]-[3] 메커니즘에 대해 살펴본다. 라우터로 설정된 리눅스 머신에 패킷이 입력 인터페이스에 도착하면 네트워크 드라이버에 의해 수신된 후 리눅스 내부 버퍼(skbuff)로 복사된다. 패킷에 대한 디멀티플렉싱(demultiplexing)을 통해 TCP 등의 상위 계층 프로토콜 스택으로 전달할 것인지, 아니면 포워딩을 통해 다음 노드로 전달할 것인지를 결정한 후 포워딩할 패킷에 대해서는 출력 인터페이스를 통해 다음 노드로 전달한다. TC가 적용되는 시점은 패킷을 출력 큐에 담는 시점에 적용되나, 커널 2.4부터는 패킷이 입력 큐에 저장되는 시점에도 TC가 적용될 수 있게 개선되었다((그림 1) 참조). 디바이스에 대해 어떤 큐잉 규칙을 적용할 때 해당 큐의 포인터는 디바이스 구조체(include/netdevice.h)에 저장된다. 전송할 데이터가 발생하면 IP 층은 패킷에 필요한 헤더 정보를 추가한 후(net/ipv4/ip_output.c) dev_queue_xmit (net/core/dev.c) 함수호출을 통해 패킷 전송을 시도한다. dev_queue_xmit에서 패킷이 dev→hard_start_xmit(skb, dev)를 통해 실제 전송되기 전에 디바이스가 관리하는 큐에 패킷을 넣은(enqueue) 후 해당 디바이스를 깨운다(qdisc_wakeup(dev)). TC에 사용된 큐잉 규칙은 q=dev→qdisc를 통해 패킷이



(그림 1) Traffic Control 적용 시점



(그림 2) 리눅스 TC 요소들



(그림 3) 리눅스 TC 설정 예제

디바이스 드라이버의 큐에 보내지기 바로 전에 적용된다.

리눅스 커널 2.4의 TC의 요소들로서 크게 큐잉 규칙(queuing discipline), 클래스(class), 필터(filter), 폴리서(policer) 등이 있다((그림 2) 참조). 각 네트워크 디바이스에 설정된 큐잉 규칙은 해당 디바이스의 입력 큐에 도착한 패킷을 어떻게 처리할 것인지를 결정한다. 예를 들어 'sch_fifo'는 FIFO를 구현한 것이다. 보다 정교하게 큐잉 규칙을 적용하기 위해서 필터를 사용할 수 있다. 필터를 통해서 패킷을 서로 다른 클래스로 구분하고 특정 클래스에 대해 우선 순위를 부여하여 차별화 할 수 있다. 각 클래스에 대해서 다시 나름대로의 큐잉 규칙이 적용될 수 있다. 큐잉 규칙으로는 11가지가 있으며 그 종류는 Class Based Queue(CBQ), Token Bucket Flow(TBF), Clark-Shenker-Zhang(CSZ), First In First Out(FIFO), Priority, Traffic Equalizer(TEQL), Stochastic Fair Queuing(SFQ), Asynchronous Transfer Mode(ATM), Random Early Detection(RED), Generalized RED(GRED), DiffServ Marker(DS_ MARK) 등이 있다.

(그림 3)은 TC 설정의 예를 보인 것이다. 한 네트워크 디바이스에 두 개의 지연(delay) 우선 순위를 가지는 큐잉 규칙을 설정한다. 필터에 의해 선택된 패킷은 '높은' 우선 순위 클래스로, 나머지는 '낮은' 우선 순위 클래스로 매핑한다. '높은' 우선 순위 클래

스는 'sch_prio'로 구현하여 버퍼에 패킷이 존재할 때 무조건 '낮은' 우선순위 패킷보다 먼저 전송할 수 있으나, '낮은' 우선 순위 패킷의 기근현상(starvation)을 막기 위해 토큰 버켓(TBF) 필터를 사용한다. '낮은' 우선 순위 패킷은 FIFO 큐잉 규칙을 가지고 '높은' 우선 순위의 패킷이 없거나 버스터 시간이 아닌 동안 전송된다. 이와 같은 설정이 CBQ를 사용해서 역시 구현될 수 있다.

리눅스 커널의 TC 모듈과 통신하여 TC를 사용자 레벨에서 제어하기 위한 응용 프로그램으로서 'IPROUTE2'를 사용할 수 있다. IPROUTE2 패키지는 리눅스 커널 2.2 이상에서 지원되는 향상된 IP 처리 기능을 제어하기 위한 사용자 프로그램으로서 기존의 디바이스 설정(device configuration), 프로토콜 주소(protocol address), 라우팅 테이블(routing table) 제어 프로그램인 'ip', 'rtmom'과 Netlink 소켓을 이용한 TC 프로그램인 'tc'를 하나로 통합한 사용자 프로그램이다. 인터넷 트래픽에 대해 QoS를 적용하기 위해서 제공되는 사용자 레벨 프로그램인 'tc'는 다양한 큐를 적용하고 해당 큐와 관련되는 클래스를 연결시킨다. 또 '라우팅 테이블', 'u32 classifier', 'tcindex classifier', 그리고 'rsvp classifier' 등을 기반으로 필터를 설정하는 데도 사용된다. 'tc'를 통해서 큐잉 규칙, 필터, 클래스를 다양하게 조합하여 패킷 스케줄링 모듈을 만들어 리눅스를 지나는 패킷을 처리할 수 있다. 'IPROUTE2'는 2.4버전부터 따로 인스톨이 필요 없이 리눅스 커널의 메인 스트림에 내장되어 서비스된다. 리눅스 커널의 TC 모듈과 사용자 응용 프로세서간 통신은 'rtnetlink' 소켓을 통해서 이루어지는데 'rtnetlink'는 Netlink 소켓을 기반으로 한 것으로서 응용 프로세스와 리눅스 커널간에 TC 오브젝트를 전달하는데 사용된다. 사용자는 'sockaddr_nl' 주소 구조체를 이용하여 커널과 통신한다. 사용자 공간에서 사용자 응용이 Netlink 소켓을 통해 자신의 요구 메시지를 커널에 전달할 때, 커널 공간의 'net/netlink/af_netlink.c' 내 'netlink_sendmesg' 함수가 호출된다. 사용자 공간의 요구 메시지는 커널 공간에서 'net/core/rtnetlink.c' 내 'rtnetlink_rcv_msg' 함수를 통해 커널로 수신된다. 이 함수에서 메시지의 헤더가 검사되고 메시지의 타입에 따라 커널내 각 처리 루틴으로 분기한다. (그림 4)는 tc를 사용하기 위한 명령어 문법을 보인 것이다.

```
tc[OPTIONS] OBJECT{COMMAND | help}
where OBJECT := {qdisc | class | filter}
OPTIONS := {-s[statistics] | -d[details] | -r[raw]}
```

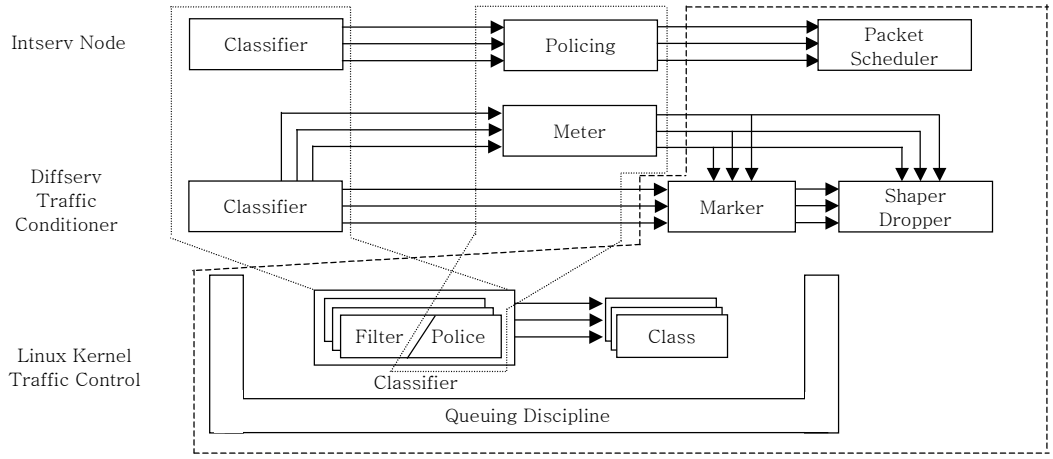
(그림 4) tc command 사용 문법

III. 트래픽 컨트롤을 이용한 QoS 지원

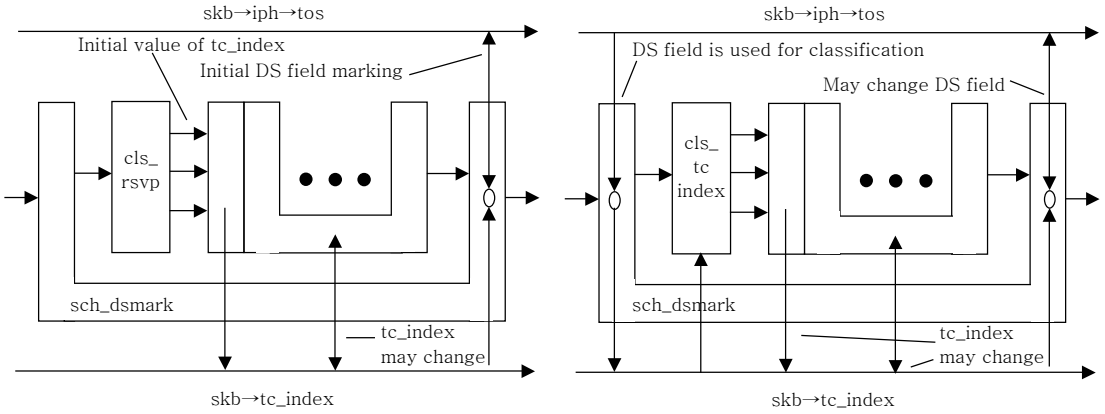
리눅스가 제공하는 TC 메커니즘은 실제 IntServ와 DiffServ를 구현하기 위한 기본적인 프레임워크를 제공한다. (그림 4)와 같이 queuing discipline, classes, 그리고 filters(policers)는 리눅스에서 QoS 메커니즘을 지원하기 위한 빌딩 블록을 제공한다.

IV. 리눅스 기반의 Diffserv

리눅스 커널 2.4에서는 커널 2.2에서 패치를 통해서 지원되었던 DiffServ extension[4],[5]이 리눅스 커널 메인 스트림으로 통합되었다. DiffServ 지원을 위해 추가된 요소들은 'cls_tcindex', 'sch_dsmark', 'sch_gred', 'skb→tc_index' 등이다. 리눅스 커널내 DiffServ의 패킷 처리과정은 다음과 같다. 일단 네트워크 인터페이스 드라이버에 의해 수신된 패킷은 리눅스 커널내 버퍼에 복사된다. 'skb→iph→tos'는 skbuff 내 IP 패킷의 TOS(Type Of Service) 필드를 가리킨다. 'sch_dsmark'는 해당 노드가 DiffServ로 동작하기 위해서 인터페이스 당 설정하는 루트 큐잉 규칙(root queuing discipline)으로서 패킷의 TOS 필드에서 DS 필드만을 마스킹을 통해 가져온 후 'skb→tc→index'에 복사한다. 'cls_tcindex'는 패킷 분류자로서 DS를 이용해서 키를 만든 후 그 키를 이용해서 해시 테이블을 검색하여 해당 패킷이 매핑될 클래스 ID를 가져온다.



(그림 5) 리눅스 TC 기반의 IntServ/DiffServ 빌딩 블록



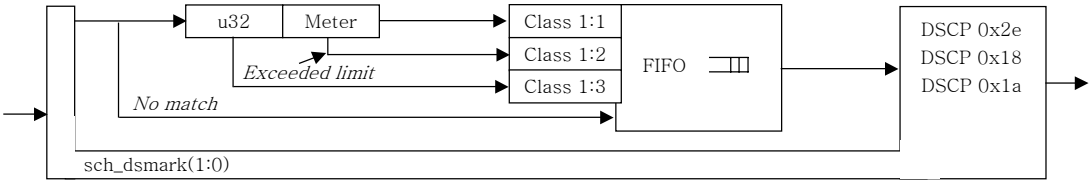
(a) DiffServ 에지 라우터 (b) DiffServ 코어 라우터

(그림 6) 리눅스 DiffServ 블록도

(그림 6) (a)의 'cls_rsvp'는 DiffServ 망의 에지 라우터(Edge Router: ER)가 플로를 구분하여 초기 DS 필드를 표시한 필터로 사용된다. 한 클래스는 자신의 내부 큐잉 규칙을 가진다. 커널 2.4에서 Diff-Serv의 PHB는 지연 우선순위만을 기반으로 구현되었다. 'Expedited Forwarding'은 기존의 CBQ나 sch_prio를 사용해서 구현이 되었으나 'Assured Forwarding'은 세 개의 드롭 우선 순위들을 정하고 있으므로 이를 구현하기 위해서 RED를 확장시킨 GRED, 'sch_gred'를 내부 큐잉 규칙으로 사용하였다.

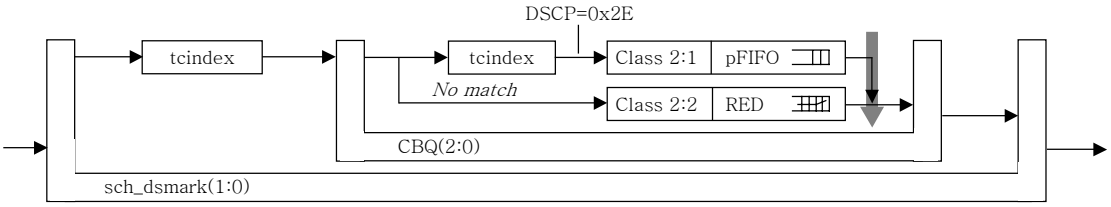
(그림 7)은 리눅스 박스를 DiffServ로 설정하고, DiffServ의 에지 라우터를 설정하는 예이다. 설정 순서는 다음과 같다. (1) 'dsmarker'를 eth0에 루트 노드로 설정한다. (2) 클래스 ID 1:1 패킷의 ECN 비트를 무시하고 DSCP를 0x2e(EF)로 설정하기 위한 마스킹과 ORing을 수행한다. (3), (4) 각각 AF31과 AF21의 DSCP를 설정한다. (5), (6) 각각 우선순위 4, 5의 u32 분류자를 추가한다. (7), (8) 10.0.0./24를 전송률이 1M(2kburst) 이내일 때 클래스 ID 1:1에 매핑하고 1M 이상일 때 1:2로 매핑한다(AF21). (9)는 10.1.0.0/16에서 192.1.0.0/16으로 향하는 TCP 패킷을 클래스 ID 1:3(AF22)으로 매핑한다.

(그림 8)은 DiffServ의 코어 라우터(Core Rou-




```
1. tc qdisc add dev eth0 handle 1:0 root dsmark indices B4
2. tc class change dev eth0 classid 1:1 dsmark mask 0x3 value 0xb8
3. tc class change dev eth0 classid 1:2 dsmark mask 0x3 value 0x68
4. tc class change dev eht0 classid 1:3 dsmark mask 0x3 value 0x4B
5. tc filter add dev eth0 parent 1:0 protocol ip prio 4 handle 1:u32
        divisor 1
6. tc filter add dev eth0 parent 1:0 protocol ip prio 5 handle 2:u32
        divisor 1
7. tc filter add dev eth0 parent 1:0 prio 4 u32
        match ip dst 10.0.0.0/24
        police rate 1Mbit burst 2K continue
        flowid 1:1
8. tc filter add dev eth0 parent 1:0 prio 5 u32
        match ip dst 10.0.0.0/24
        flowid 1:2
9. tc filter add dev eth0 parent 1:0 prio 4 u32
        match ip dst 10.1.0.0/16
        match ip src 192.1.0.0/16
        match ip protocol 6 0xff
        match ip dport 0x17 0xffff
        flowid 1:3
```

(그림 7) DiffServ 에지 라우터 설정 예




```
1. tc qdisc add dev eth0 handle 1:0 root dsmark indices 64
        set_tc_index
2. tc filter add dev eth0 parent 1:0 protocol ip prio 1
        tcindex mask 0xfc shift 2
3. tc qdisc add dev eth0 parent 1:0 handle 2:0 cbq
        bandwidth 10Mbit allot 1514 cell 8 avpkt 1000 mpu 64
4. tc class add dev eth0 parent 2:0 classid 2:1 cbq
        bandwidth 10Mbit
        rate 1500Kbit avpkt 1000 prio 1 bounded isolated
        allot 1514 weight 1 maxburst 10 defmap 1
5. tc qdisc add dev eth0 parent 2:1 pfifo limit 5
6. tc filter add dev eth0 parent 2:0 protocol ip prio 1
        handle 0x2e tcindex classid 2:1 pass_on
7. tc class add dev eth0 parent 2:0 classid 2:2 cbq
        bandwidth 10Mbit rate 5Mbit avpkt 1000 prio 7
        allot 1514 weight 1 maxburst 21 borrow
8. tc qdisc add dev eth0 parent 2:2 red limit 60KB min 15KB
        max 45KB burst 20 avpkt 1000 bandwidth 10Mbit
        probability 0.4
9. tc filter add dev eth0 parent 2:0 protocol ip prio 2
        handle 0 tcindex mask 0 classid 2:2 pass_on
```

(그림 8) DiffServ 코어 라우터 설정 예

ter: CR)를 설정하는 예이다. (1) 'dsmarker'를 eth0에 루트노드로 설정한다. (2) ECN 비트를 제거하고 DSCP 필드를 꺼내기 위해 필터를 추가하는 과정이다. (3)은 2:0에 CBQ 큐잉 규칙을 설정한 것이다. (4)는 CBQ 타입의 자식 클래스 2:1을 설정하고 1.5Mbps rate 전송을 설정한 것이다. (5) CBQ 타입의 2:1에 최대 5개 패킷의 큐를 가진 pfifo 큐잉 규칙을 설정한 것이다. (6) 'skb→tc_index'가 0x2e(EF)인 모든 패킷을 2:1 클래스로 매핑한다. (7) 2:2 클래스가 BE(Best Effort) 트래픽을 다루도록 설정한 것이고 전송률을 5Mbps로 설정한다. 2:2 클래스에 borrow를 설정하여 여분의 대역폭을 빌려 쓸 수 있으나 2:1 클래스가 'isolated'로 설정되어 있으므로 최대 3.3Mbps를 추가로 빌려 쓸 수 있다. (8) BE 트래픽을 다루기 위해 RED를 버퍼 관리 스킴을 사용할 것을 설정한 것이다. (9) 0x2e 이외의 모든 패킷을 2:1로 매핑할 것을 설정한 것이다.

V. 결 론

본 고에서 리눅스 환경에서 QoS 메커니즘 구현을 위한 모듈로서 TC에 대해서 커널 공간과 사용자 공간에 대해 설명하였다. 아울러, 리눅스 커널 2.4에서 통합된 DiffServ 모듈에 대해 커널 공간에 새로이 추가된 요소들, 동작 메커니즘, 그리고 사용자 공간에서 DiffServ 라우터 설정 예를 살펴 보았다. 현재까지는 DiffServ 만이 리눅스 커널에 통합되어 있으나 IntServ나 MPLS의 구현을 위해서는 ISI의 RSVPd나 캠브리지 대학의 MPLS 리눅스 코드 등을 사용하여 구현할 수 있다.

참 고 문 헌

[1] Werner Almesberger, *"Linux Network Traffic Control - Implementation Overview 2001,"* EPFL ICA.

[2] Saravanan Radhakrishnan, *"Linux-Advanced Networking Overview"* ITTC, University of Kansas.

[3] Netherlabs, Gregory Maxwell, Remco van Mook Martijn van Oosterhout, Paul B Schroeder, Jasper Spaans, *"Linux2.4 Advanced Routing Howto"* TLDP.

[4] *ICA Different Services on Linux,* http://diffserv.sourceforge.net/

[5] Werner Almesberger, Jamal Hadi Salim, Alexey Kuznetsov, *"Differentiated Services on Linux"* EPFL ICA, Notel Networks, INR Moscow.

[6] P. Trimintzios, I. Andrikopoulos, G. Pavlou, C.F. Cavalcanti, D. Goderis, Y. T'Joens, P. Georgatsos, L. Georgiadis, D. Griffin, C. Jacquenet, R. Egan & G. Memenios, *"An Architectural Framework for Providing QoS in IP Differentiated Services Networks"* IST TEQUILA; 7th IFIP/IEEE International Symposium on Integrated Network Management, IM 2001.