TMO 모델의 분산 IPC에 적합한 신뢰성있는 멀티캐스트 전송 프로토콜

안진섭° 김영준 홍진표
한국외국어대학교
{jinsurby°, ddanggae, jphong}@hufs.ac.kr

A Reliable Multicast Transport Protocol Suitable for Distributed IPC of the TMO Model

Jin-Sub Ahn° Young-Jun Kim, Jin-Pyo Hong
Dept. of Information Communication Engineering, Hankuk University of Foreign Studies

요 약

분산 환경에서의 메시지 전송에 대한 TMO 모델은 실시간 프로그래밍, 분산 시스템 프로그래밍, 병행 프로그래밍, 객체 지향 프로그래밍의 모든 장점을 통합한 분산 실시간 객체 모델로서 운영체제의 오버헤드를 줄이고 프로세스 처리의 정밀도를 높인다. LTMOS는 TMO로 구성된 프로그램을 여러 개의 Linux 플랫폼에서 수행하기 위해 개발된 미들웨어 엔진으로서 기본적으로 분산 IPC 메시지를 유니캐스트로 전달하지만 효율성을 높이기 위해 멀티캐스트 전송 방식도 필요하다. 이에, 본 논문은 TMO 분산 IPC 환경에 멀티캐스트를 적용하여 효율성과 데이터의 신뢰성을 보장하는 ACK/NACK 기반 멀티캐스트 전송 프로토콜을 구현한다.

1. 서론

TMO(Time-triggered Message-triggered Object) 모델[1]은 실시간 프로그래밍, 분산 시스템 프로그래밍, 병행 프로그래밍, 객체지향 프로그래밍의 모든 장점을 통합하여 소프트웨어 설계를 위한 강력한 객체 모델로 90년대 초반부터 세계적으로 주목 받고 있는 새로운 프로그래밍 패러다임이다. 분산 환경에서 메시지 전송의 한 연구인 TMO 모델은 분산 실시간 객체 모델로서 분산 환경에서의 실시간 처리를 위해서 기본적으로 유니캐스트 전송을 사용한다. 대역폭 측면에서 멀티캐스트가 유니캐스트 보다 효율적이지만 멀티캐스트 전송방식을 사용할 경우 손실 없는 메시지 전송을 위해 '신뢰성있는 멀티캐스트 전송(Reliable Multicast Transport, 이하 RMT)' 프로토콜이 필요하다.

RMT 프로토콜은 신뢰성을 제공하지 않는 IP 멀티캐스트에 "에러제어 및 혼잡제어를 통한 멀티캐스트 데이터 전송에 신뢰성을 제공"하는 프로토콜이다. 본 논문은 리눅스 기반의 TMO 분산 IPC 환경에서 RMT 프로토콜을 적용하여 TMO 채널내의 분산 IPC 메시지 전달에 100% 신뢰성을 제공과 효율적인 대역폭의 사용을 목적으로 한다.

2. 관련 연구

2.1 TMO 모델

최근 분산 실시간 분야에서 새로운 패러다임으로서 세계적으로 활용되기 시작한 실시간 객체 모델인 TMO는 운영체제의 오버헤드를 줄이고 프로세스 처리에 정밀도를 높인다. 일반적으로 복수개의 TMO로 설계된 시스템으로 분산 환경에서 시간조건에 의해 수행된다. LTMOS(Linux TMO System)은 이러한 TMO로 구성된 프로그램을 단일 또는 복수개의 Linux 플랫폼에서 수행하기 위해 개발된 미들웨어 엔진이다.

* 본 연구는 정보통신부 및 정보통신연구진흥원의 대학IT연구센터 육성·지원사업의 연구결과로 수행되었음

TMO는 리눅스 기반의 실시간 쓰레드 스케줄러, 시간 조건과 분산 IPC 메시지에 의해 구동되는데, 이를 위해 실시간 쓰레드 멤버를 위한 동기화, 분산 IPC, 분산 클럭 동기화, 리눅스 커널간의 인터페이스 등이 필요하다.

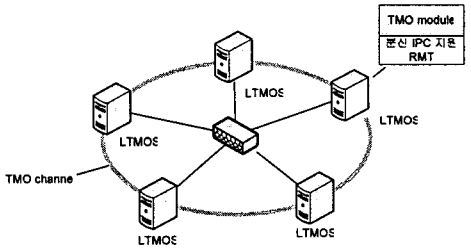

그림 1. LTMOS 분산 IPC 환경 구성도

TMO는 LTMOS의 IPC 도구로 제공되는 채널을 이용하여 TMO의 메소드 간에 통신을 할 수 있다. 채널은 네트워크를 추상화 한 양방향 분산 IPC 도구로 각각의 채널은 유일한 ID를 갖는다. 채널 ID는 분산 시스템 상의 모든 TMO에 동일하게 적용되는 전역적인 값이며, 모든 분산 노드의 TMO에 대하여 투명한 특성을 가지고 있다. 채널은 읽기, 쓰기, 읽기/쓰기의 용도로 할당하여 사용할 수 있고, 한 채널에 쓰여진 메시지는 분산 시스템상의 같은 채널을 읽기 모드로 사용하고 있는 모든 TMO에 전달된다. 이러한 리눅스 기반 분산 TMO 환경을 사용하면 실시간 객체 프로그래밍, 실시간 자원관리, 분산 시스템을 제공하는 종합적인 실시간 운영체제가 가능하다. 그림 1은 LAN에서 분산 IPC 환경을 구성한 것이다. 실시간 프로세스의 동작에 필요한 TMO 채널은 최고 32개까지 할당할 수 있다.

2.2 RMT 관련 표준

IETF의 RMT WG(Working Group)은 멀티캐스트 전송에서 신뢰성있는 전송 특성을 제공하기 위해 종단간 멀티캐스트 전송과 관련된 오류 및 흐름제어와 같은 신뢰성 있는 멀티캐스트 전송 관련 표준 기술을 개발하는 모임이다. 주요 연구 방향은 대용량 데이터(bulk-data)의 일대다 전송에 대해 초점을 맞추고, 각 전송방식에 따른 모델[2][3]을 정의하여 전송계층에는 NACK 기반의 NORM(Nack Oriented Reliable Multicast) 프로토콜[4][5]과 LCT 기반의 ALC(Asynchronous Layered Coding) 프로토콜에 대한 표준화를 완료하였다.

3. TMO 분산 IPC에 적합한 RMT 프로토콜 특성

3.1 TMO 분산 IPC 요구사항

TMO 모델의 분산 IPC에 멀티캐스트 전송방식을 적용할 경우 LAN에서 IPC 메시지의 처리시간(sender → receiver)은 10ms 이내, 타이머 해상도는 1ms 이어야 한다. Linux Kernel 2.4 버전은 기본 타이머 해상도가 10ms 이므로 TMO를 사용할 경우 타이머 해상도를 1ms로 재설정 한 후 커널 컴파일을 해야 한다.

3.2 RMT 프로토콜 설계에 대한 고려사항

NORM 프로토콜의 기본 동작방식은 receiver가 데이터 손실을 발견할 경우 sender에게 NACK 메시지를 전송하여 재전송 요청을 하는 프로토콜이다. Sender는 패킷이 손실될 경우 재전송 회수를 줄이기 위해 receiver가 자체적으로 에러복구를 하도록 FEC 인코딩을 적용하여 데이터를 전송하는 것으로 요약 할 수 있다.

TMO 모델의 분산 IPC를 위한 RMT 프로토콜 설계에는 NORM 프로토콜의 장점만을 취하였다. IPC 메시지 처리시간을 만족시키기 위해 프로토콜 내의 타이머는 1ms 단위로 동작하고, 데이터의 신뢰성 보장 측면에서 sender의 PCB(Protocol Control Block)에 TMO 채널 노드들의 IP 주소 목록인 acking node list를 유지한다. 또한 NORM 프로토콜이 NACK 기반의 응답 메시지를 사용했던 것과 달리 receiver의 응답 메시지는 데이터 수신에 대한 ACK/NACK를 비트맵 형태로 전달하여 효율성을 높인다. 그리고, LAN은 패킷이 손실될 확률이 매우 낮기 때문에 불필요한 오버헤드를 발생시키는 FEC 인코딩은 적용은 고려하지 않는다.

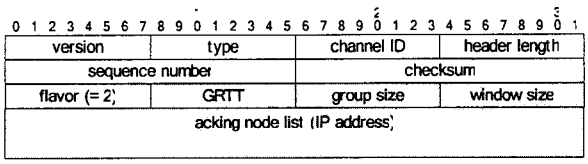

그림 2. CMD(GRTT_ADV) 메시지 포맷

그림 2의 version ~ checksum 필드는 TMO 분산 IPC 지원용 RMT 프로토콜의 모든 메시지에 공통적으로 사용하는 고정헤더(fixed header) 포맷이며, 다음 부분에 CMD, DATA, ACK 메시지 포맷이 위치한다. 고정 헤더 이후의 헤더 포맷은 sender의 CMD(GRTT_ADV) 메시지로서 연결 설정 후 receiver에게 Group RTT, 그룹 크기, acking node list를 전달한다.

4. RMT 프로토콜 구현

4.1 연결 설정 및 해제

4.1.1 Sender의 FSM

TMO 모듈의 active open 요청 후 RTT를 측정하기 위해 타임스탬프를 CMD(TS) 메시지로 전송하고, CMD(TS) SENT 상태로 이동한다. sender의 CMD(TS) SENT 상태는 크게 두 가지 의미를 갖는다. 기본 GRTT(Group RTT) 타이머가 만료되기 전까지 receiver들의 ACK(TS)를 수신하고 RTT 측정하며, PCB(Protocol Control Block)의 acking node list에 IP 주소를 추가한다. 이후 타이머가 만료되면 CMD(TS) SENT 상태에서 누적된 RTT 값들을 통해 RTO(Retransmission Timeout) 추정(estimation)을 한다. 그리고 1ms tick 단위로 표현한 RTO값과 acking node list를 receiver에게 CMD(GRTT_ADV) 메시지로 전달하고, Data Transfer 상태로 전이한다. sender는 이 과정에서 모든 메시지를 멀티캐스트로 전송하고, 재전송 타이머는 CMD 관련 메시지에만 유효하다. 그림 3은 TMO 분산 IPC 지원용 RMT 프로토콜의 FSM(Finite State Machine)를 나타낸다.

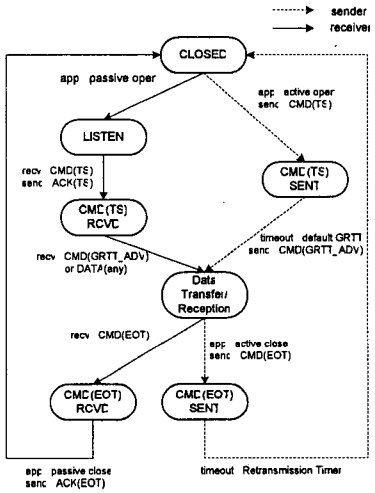

그림 3. TMO 모델 분산 IPC 지원용 RMT 프로토콜의 FSM

연결종료 과정은 연결 설정 보다 단순하다. TMO 모듈의 active close 요청에 따라 버퍼에 있는 모든 데이터를 전송한 후 CMD(EOT) 메시지를 전달한다. 모든 receiver로부터 ACK(EOT)를 수신할 때까지 CMD(EOT) 메시지를 재전송하며 마지막 CLOSED 상태로 이동한다.

4.1.2 Receiver의 FSM

receiver는 멀티캐스트 그룹에 먼저 참가(join) 한 후에 LISTEN 상태에서 sender의 CMD(TS) 메시지를 기다린다. CMD(TS) 메시지를 수신하면 타임스탬프 응답을 sender에게 전송하고 CMD(TS) RCVD 상태로 이동한다.

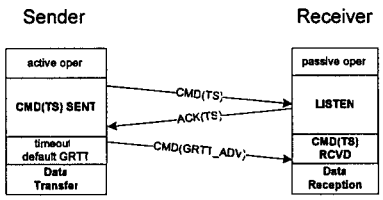

그림 4. 연결 설정 과정에서 sender/receiver 메시지 흐름

이후 receiver는 CMD(GRTT_ADV) 메시지 수신을 통해 RTO를 sender의 RTO 값으로 재설정하고, acking node list를 통해 자신이 TMO 채널에 존재하는지 확인한다. 만약 CMD(GRTT_ADV) 메시지를 수신하지 못하면 재전송 요청은 하지 않고, Data

Transfer/Reception 상태에서 처리한다.

4.2 데이터 송/수신

4.2.1 Sender의 데이터 송신 절차

Sender의 Data Transfer 상태에 ACK 수신과 재전송 과정에 대한 세부 FSM을 그림 5에 정의하였다. 'Data Transfer' 상태에서 ACK(TS) 메시지의 수신은 receiver가 CMD(TS) RCVD 상태에서 CMD(GRTT_ADV) 메시지를 수신하지 못한 경우를 나타낸다. 이 과정은 해당 receiver가 CMD(GRTT_ADV) 메시지를 재전송 받기 위한 것이다. 이때 receiver가 전송한 타임스탬프 응답 메시지는 sender의 RTO 추정에 반영되지 않는다.

Sender는 'Data Transfer' 상태에서 TMO 모듈의 요청 또는 송신 윈도우 크기가 0이 될 때까지 송신 버퍼에 저장된 데이터를 연속적으로 전송하고 CMD(FLUSH) 메시지로 receiver들의 ACK를 기다린다. 이러한 전송 방식은 TCP의 '세그먼트 전송 - delayed ACK'와 전혀 다르며 단 한번의 ACK를 대기한다는 점에서 NORM 프로토콜의 동작절차와 동일하다.

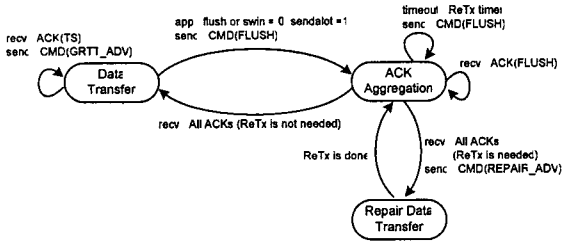

그림 5. 'Data Transfer' 상태에서 sender의 세부 상태 전이도

ACK(FLUSH) 메시지는 sender의 송신 윈도우 크기를 갖는 비트맵으로 정의되는데, receiver는 단일 응답 메시지로 sender가 전송한 모든 데이터 패킷에 대한 ACK/NACK 여부를 알릴 수 있다. sender는 'ACK Aggregation' 상태에서 acking node list의 모든 receiver로부터 ACK(FLUSH)를 수신하면 데이터의 재전송 여부를 결정한다. 재전송이 필요 없을 경우 다시 Data Transfer 상태로 전이하여 새로운 데이터를 전송하고, 재전송이 필요할 경우에는 재전송 시작을 알리는 일종의 힌트로서 CMD(REPAIR_ADV) 메시지를 전송하고, Repair Data Transfer 상태로 전이하여 복구가 필요한 패킷만 재전송 한다. 만약 TMO 채널에서 연속적인 IPC 메시지 전송이 계속된다면 Data Transfer → ACK Aggregation → Repair Data Transfer 등 일련의 과정을 반복해서 수행할 것이다.

4.2.2 Receiver의 데이터 수신 절차

데이터 수신 단계에서 receiver의 동작절차는 sender에 비해 단순하다. CMD(FLUSH) 메시지를 수신하면 receiver는 패킷수신 여부를 비트맵으로 구성하여 ACK(FLUSH) 메시지를 전송하고 'ACK Generation' 상태로 전이한다.

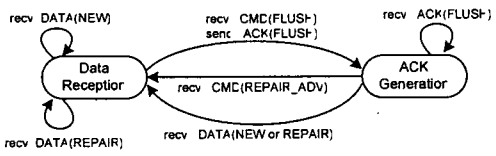

그림 6. 'Data Reception' 상태에서 receiver의 세부 상태 전이도

이 후 CMD(REPAIR_ADV) 메시지 또는 DATA 메시지 수신에 의해 다시 Data Reception 상태로 이동하는데, DATA 메시지의 타입(NEW, REPAIR)에 따라 재전송 데이터에 대한 수신 여부와 새로운 데이터의 수신 동작을 결정 할 수 있다.

4.2.3 Sender와 Receiver의 데이터 재전송 절차

그림 7은 TMO 채널에서 데이터 송/수신 예제로서 프로토콜의 메시지 흐름을 나타낸다. receiver1은 패킷손실이 없는 정상적인 절차이며, receiver2는 데이터 중에서 3, 4번 패킷 손실이 발생하여 재전송을 요청한다. receiver 1는 첫 번째 ACK Generation 상태 이후에 sender의 CMD(REPAIR_ADV) 메시지를 통해 데이터가 재전송될 것이라는 것을 인식한다. DATA 메시지는 종류(flavor)에 따라 구분이 가능하므로 불필요한 메시지는 처리하지 않는다.

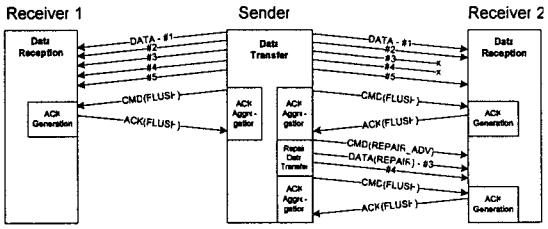

그림 7. Sender/Receiver 데이터 송/수신 과정의 예

4.3 구현 환경 및 테스트

TMO는 기본적으로 Linux Kernel 2.4 버전 이상을 요구하지만 타이머 해상도 문제로 커널 컴파일이 필요하다. TMO 분산 IPC 지원용 RMT 구현은 별도의 커널 컴파일이 필요 없도록 Fedora Core 3(kernel 2.6.10)에서 하였으며 LAN환경에서 4개의 노드 - sender, receiver 3 대로 구성한 환경에서 테스트 하였다.

5. 결론

일반적인 TMO 분산 IPC 환경은 TMO 채널 내에 노드의 수가 증가할수록 모든 IPC 메시지의 교환이 유니캐스트 전송방식으로 이뤄지므로 대역폭의 비효율적인 사용과 메시지의 전달 시간이 증가한다. 본 논문에서는 Linux TMO 분산 IPC 환경에 신뢰성있는 멀티캐스트 전송 프로토콜을 적용하기 위해 TMO 요구사항을 분석하고, RMT 관련 표준을 참고하여 프로토콜을 설계하고, 구현하였다. LTMOS 분산 IPC 환경은 네트워크 형태가 LAN이기 때문에 IP 멀티캐스트 패킷 포워딩이 발생하지 않으며 패킷 손실률 또한 매우 낮다. 실제 프로토콜이 정상적으로 동작하는지 검증하기 위해 recvfrom(), sendmsg()의 실행에서 10% 비율로 처리하지 않도록 함수를 재작성하여 재전송 절차 확인하였다.

참고문헌

[1] Real-time Leaders Co.Ltd, "LTMOS(TMO System on Linux)'s Programmer's Guide", 2002년 6월

[2] M. Handley, S. Floyd, "The Reliable Multicast Design Space for Bulk Data Transfer", IETF RMT WG, RFC 2887, August. 2000

[3] B. Whetten, L. Vicisano, "Reliable Multicast Transport Building Blocks for One-to-Many Bulk Data Transfer", IETF RMT WG, RFC 3048, January. 2001

[4] B.Adamson, C.Bormann, M.Handley and J.Macker, "Negative-Acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Building Blocks", IETF RMT WG, RFC 3941, November. 2004

[5] B.Adamson, C.Bormann, M.Handley and J.Macker, "Negative-Acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol", IETF RMT WG, RFC 3940, November. 2004