SPIN을 이용한 고속 네트워크 인터페이스 카드의 펌웨어 검증

°진 현 욱*, 방 기 석**, 최 진 영**, 유 혁*
고려대학교 컴퓨터학과

*{hwjin, hxy}@os.korea.ac.kr
**{kbang, choi}@formal.korea.ac.kr

Formal Verification of MCP using SPIN

°Hyun-Wook Jin, Ki-Seok Bang, Jin-Young Choi, and Hyuck Yoo
Department of Computer Science and Engineering, Korea University

요 약

고속 네트워크 인터페이스 카드에서 수행되는 펌웨어의 기능이 다양해지고 그 개발이 자유로워짐에 따라 체계적인 펌웨어의 정형검증이 요구된다. 본 논문은 정형검증 도구인 SPIN 을 사용하여 미리넷 네트워크 인터페이스 카드에서 수행되는 펌웨어인 MCP 를 정형검증하여 데이터 송신에 오류가 없음을 보인다. 이를 통해서 본 논문은 펌웨어에 대한 가능한 정형검증 방법을 보이고 SPIN 의 적용 범위를 확대한다.

1. 서론

미리넷(Myrinet) [1], 기가비트 이더넷(Gigiabit Ethernet) [2]과 같은 고속 네트워크가 등장함에 따라 단순했던 네트워크 인터페이스 카드(NIC; Network Interface Card)에 다양한 기능과 Gbps 의 성능이 요구되고 있다. 이러한 요구를 충족시키기 위해서 NIC 의 하드웨어 뿐만 아니라 NIC 에서 수행되는 펌웨어(firmware)의 구조가 복잡해지고 있으며, 또한 펌웨어를 사용자가 직접 개발할 수 있도록 도구가 제공되어 [1][3][4] 특정 통신 시스템을 위해 자체적으로 펌웨어를 개발 또는 수정하는 경우가 증가하게 되었다 [5][6][7][8].

이와 같이 NIC 의 펌웨어 개발이 활발히 이루어지고는 있으나 정형기법을 통한 검증은 간과되고 있다. 그러나 펌웨어의 오류에 의해서 발생되는 오동작은 찾아내기 힘들 뿐만 아니라 고속 네트워크의 물리적 대역폭을 상위 프로토콜 계층에게 제공하는 데 치명적일 수 있다. 예로서 펌웨어의 오류에 의해 NIC 에서 발생하는 패킷의 간헐적 손실 및 순서 뒤바뀜을 살펴보자. 이와 같은 오동작을 상위 프로토콜에서는 네트워크의 혼잡 등에 의한 것으로 보고 대처할 것이며, 특히, TCP 의 경우는 패킷 손실 방지와 정렬을 보장하므로 펌웨어의 오류는 더욱 발견하기 어렵다. 하지만 이러한 오류는 TCP 에서의 데이터 재전송과 재정렬 등을 위한 작업을 수행하게 하므로 그에 따른 오버헤드가 발생하여 고속 네트워크의 대역폭을 상위 계층이 충분히 사용할 수 없게 된다.

본 논문은 현존하는 LAN 중에서 가장 빠른 Myrinet 의 NIC 에서 수행되는 펌웨어인 MCP(Myrinet Control Program)[1]를 정형검증한다. MCP 의 정형검증을 위한 첫 단계로 본 논문에서는 송신과정만을 모델링하고 검증한다. 정형검증 도구로서는 SPIN[9]을 사용한다. SPIN 은 Bell 연구소에서 개발되어 널리 사용되고 있는 모델 체킹 도구로서 비동기 프로세스 시스템을 디자인하고 검증하는 데 사용된다. SPIN 은 PROMELA(PRocess MEta LAnguage)[10]라는 명세언어를 사용하여 디자인된 명세를 받아들여 선형 시제 논리(LTL; Linear Temporal Logic)[11]로 정의된 정확성(correctness)을 검증함으로써 시스템의 오류를 검사한다. 본 논문에서는 MCP 의 송신 부분을 명세하고 송신시 패킷의 손실이 일어나지 않음을 검증하였다.

본 논문은 다음과 같이 구성되어 있다. 본 서론에 이어 2 장에서는 MCP 에 대한 기본 구조와 수행 방식에 대해서 설명한다. 3 장에서는 MCP 의 송신 관련 부분을 모델링하고 정형검증한다. 마지막으로 4 장에서는 본 논문의 결론을 맺는다.

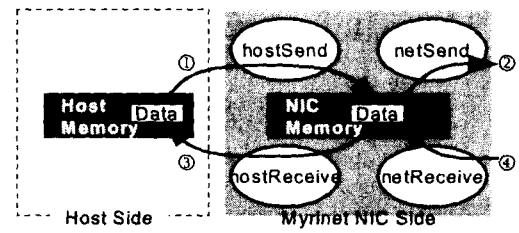

그림 1 MCP의 기본 구조

2. MCP (Myrinet Control Program)

미리넷은 양방향(full-duplex)으로 1.28+1.28Gbps 의 대역폭을 제공하는 초고속 LAN 이다. MCP 는 Myrinet NIC 의 펌웨어로서 Myrinet NIC 에 있는 SRAM 에 로드되어 RISC 프로세서에 의해서 수행된다. MCP 는 호스트와 네트워크 사이의 데이터 송수신을 관리하며, 이를 위해서 미리넷 패킷 헤더의 조작과 DMA(Direct Memory Access), 인터럽트 발생을 담당하게 된다

또한 스카우트 메시지(Scout Message)를 송수신하여 네트워크를 매핑하는 등의 다양한 기능을 수행한다.

MCP 의 송수신을 담당하는 부분은 그림 1 과 같이 네개의 객체들로 구성되어 있으며 – hostSend, netSend, hostReceive, netReceive – 이 네 객체들 각각은 비동기적으로 수행된다. hostSend 와 netSend 는 송신을 담당한다. hostSend 는 호스트에서 NIC 으로의 데이터 이동을 담당하며 (그림 1 의 ①), netSend 는 NIC 에서 네트워크로의 데이터 송신을 담당한다 (그림 1 의 ②). NIC 에서의 패킷 수신은 hostReceive 와 netReceive 에 의해서 수행된다. hostReceive 는 네트워크로부터 NIC 으로 수신된 데이터를 호스트에 전달하며 (그림 1 의 ③), netReceive 는 네트워크로부터 도착한 패킷을 NIC 으로 수신하는 역할을 한다 (그림 1 의 ④).

호스트와 NIC 간의 데이터 이동과 NIC 과 네트워크간의 데이터 이동은 모두 MCP 가 DMA 와 관련된 레지스터들을 세팅함으로써 이루어진다. 그리고 MCP 는 호스트로부터의 송신 요청, 네트워크로부터의 새로운 패킷 도착, DMA 종료 등을 ISR(Interrupt Status Register)을 통해서 인식하게 된다.

3. SPIN을 이용한 모델링 및 검증

3.1 모델링

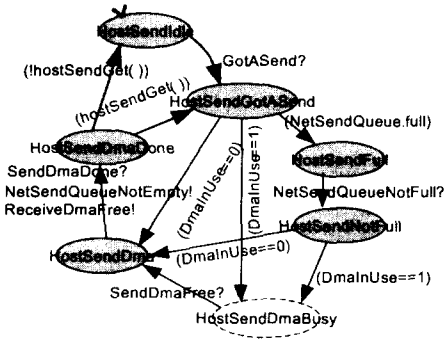

그림 2 hostSend 의 DFA

본 논문은 MCP 의 송신 부분만을 모델링하고 검증한다. 즉, 2 장에서 설명되어진 hostSend 와 netSend 만을 대상으로 한다.

그림 2 는 hostSend 의 동작을 상태전이도의 형태로 모델링한 것이다. HostSendIdle 이 시작 상태이며, HostSendGotASend 는 호스트로부터 송신할 패킷이 있음을 인지한 상태이다. HostSendIdle 에서 NIC 의 버퍼에 가능한 공간이 없을 때에는 HostSendFull 상태로 전이하며, 빈 버퍼가 있을 경우에는 HostSendDma 상태로 전이하여 패킷을 호스트에서 NIC 으로 DMA 한다. HostSendFull 상태에서는 가용한 버퍼를 기다린 후 HostSendNotFull 상태로 전이되고 HostSendDma 상태로 전이하여 DMA 를 수행한다. HostSendDma 상태에서 DMA 가 종료되면 HostSendDmaDone 상태로 전이된다. HostSendDmaBusy 상태는 본 논문에서는 고려하지 않으며, 수신을 위한 부분까지 함께 검증할 때 포함될 것이다.

그림 3 은 그림 2 의 상태전이도를 기반으로 PROMELA 를 사용하여 모델링한 것의 일부이다. 그림 3 에서 hsqueue 는 호스트에서 송신 요청한 데이터가 있는 큐이다. 즉, 아직 NIC 으로 DMA 되지 않고 호스트 메모리에 있는 패킷들의 큐이다. tohs 는 hostSend 에게 이벤트를 알리기 위한 채널이다.

```
proctype hostSend()
{
int msg, num = 0;

do
:: (hs_state == HostSendIdle) ->
                hsqueue?msg;
                if
                ::(msg == GotASend) ->
                                hs_state = HostSendGotASend
                :: else ->
                                skip
                fi
:: (hs_state == HostSendGotASend) ->
                if
                :: (full(nsqueue)) ->
                                hs_state = HostSendFull
                :: (nfull(nsqueue) && DmaInUse==0) ->
                                DmaInUse = 1;
                                hs_state = HostSendDma
                fi
:: (hs_state == HostSendFull) ->
                tohs?msg;
                if
                ::(msg == NetSendQueueNotFull) ->
                                hs_state = HostSendNotFull
                :: else ->
                                skip
                fi
                                :
od
}
```

그림 3 PROMELA 로 모델링된 hostSend 의 일부

그림 4 는 netSend 를 상태전이도로 모델링한 것이며 NetSendIdle 이 시작 상태이다. NetSendGotASend 는 NIC 에 네트워크로 송신할 데이터가 있음을 인지한 상태이고, NetSendBusy 상태로 전이하여 NIC 에서 네트워크로 패킷을 송신한다. 송신이 완료되면 NetSendDmaDone 상태로 전이한다. NetSendWaiting 상태는 수신을 고려할 때 모델링에 포함될 것이다.

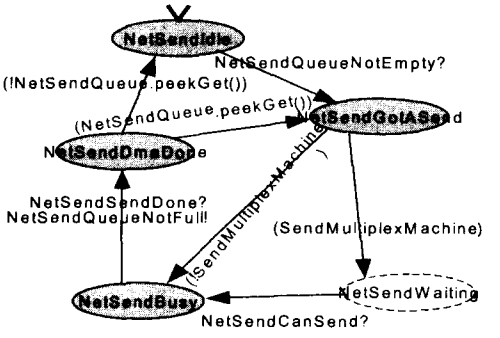

그림 4 netSend 의 DFA

그림 5 는 PROMELA 로 netSend 를 모델링한 것이다. 그림 5 에서의 nsqueue 는 NIC 으로 DMA 되어 netSend 에 의해서 네트워크로 송신되기를 기다리는 패킷들의 큐이며, tons 는 netSend 에게 이벤트를 알리기 위한 채널이다.

```
proctype netSend()
{
int msg, num = 0;

do
:: (ns_state == NetSendIdle) ->
        nsqueue?msg;
        if
        ::(msg == NetSendQueueNotEmpty) ->
                ns_state = NetSendGotASend
        :: else ->
                skip
        fi
:: (ns_state == NetSendGotASend) ->
        if
        ::(SendMultiplexMachine == 0) ->
                SendMultiplexMachine = 1;
                ns_state = NetSendBusy
        :: else ->
                skip
        fi
:: (ns_state == NetSendBusy) ->
        ns_flag = 1;
        tons?msg;
        if
        ::(msg == NetSendSendDone) ->
                SendMultiplexMachine = 0;
                tohs!NetSendQueueNotFull;
                ns_state = NetSendDmaDone
        :: else ->
                skip
        fi
                        :
od
}
```

그림 5 PROMELA 로 모델링된 netSend 의 일부

3.2 검증

앞에서 모델링된 hostSend 와 netSend 에 대해서 데이터 송신 시 패킷 손실이 발생하는 지를 검증하였다. 패킷 손실이 발생하지 않음을 보장하려면, hostSend 가 HostSendGotASend 상태에서 HostSendDmaDone 상태로 전이되는 것과 netSend 가 NetSendGotASend 상태에서 NetSendDmaDone 상태로의 전이됨이 보장되어야 한다. 또한 hostSend 의 HostSendDmaDone 상태는 netSend 의 NetSendGotASend 상태를 야기해야 한다. 이를 위해 그림 6 의 (a)와 같이 검증에 필요한 인수들을 정의하고 그림 6 의 (b)와 같이 LTL 로 만족해야할 조건들을 표현하였다. 이번 연구에서는 전송의 요구가 있을 경우 전송이 이루어지는 가를 확인하였다. 검증결과 그림 7 과 같이 오류가 없음을 알았으며, 이는 MCP 가 데이터 송신시 프로그램 오류에 의해서 패킷 손실이 발생하지 않음을 의미한다.

```
#define  p  (hs_state == HostSendGotASend)
#define  q  (hs_state == HostSendDmaDone)
#define  r  (ns_state == NetSendGotASend)
#define  s  (ns_state == NetSendDmaDone)
```

(a) Symbol Definitions

```
[] ( p -> <> q )
[] ( r -> <> s )
[] ( p -> <> s )
```

(b) LTL Formulae

그림 6 정형 검증

4. 결론 및 향후 계획

본 논문은 Myrinet NIC 에서 수행되는 펌웨어인 MCP 를 정형검증하였다. 검증은 MCP 의 송신만을 대상으로 했으며, DFA 와 PROMELA 를 사용해서 모델링하고 SPIN 을 사용하여 검증하였다. 검증 결과, MCP 는 데이터 송신 시 패킷 손실을 야기하지 않음을 증명하였다.

이번 연구에서는 MCP 의 모든 구성요소를 모델링하지는 않았다. 실제로 MCP 의 기능을 모두 정형검증 하기 위해서는 Receive 부분도 모델링되고 송신과 함께 검증을 해야 한다. 이러한 작업이 모두 이루어진다면, 고속 네트워크 NIC 의 모든 기능을 정형명세함으로써 MCP 의 정확성 검증이 가능하리라 생각한다. 특히 메시지를 연속해서 전송하고 수신할 경우 그 순서 역시 정확성의 중요한 요소가 된다. 향후 수신 부분의 기능을 명세하고 MCP 가 만족해야 할 특성에 대해 검증을 수행할 계획이다.

이와 같이 본 논문은 고속 네트워크의 NIC 에서 수행되는 펌웨어를 SPIN 을 사용하여 정형검증함으로써 펌웨어에 대한 가능한 정형검증 방법을 보였을 뿐만 아니라, SPIN 의 적용범위를 펌웨어를 포함하는 프로토콜 계층으로 확장하였다.

```
Full statespace search for:
        never-claim             +
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid endstates       - (disabled by never-claim)

State-vector 412 byte, depth reached 516, errors: 0
  176666 states, stored (278036 visited)
  267211 states, matched
  545247 transitions (= visited+matched)
   85522 atomic steps
```

그림 7 Spin 을 이용한 MCP 의 정형검증 결과

참고 문헌

[1] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su, "Myrinet - A Gigabit-per-Second Local-Area Network," IEEE MICRO, February 1995.

[2] Gigabit Ethernet Alliance, IEEE 802.3z. The Emerging Gigabit Ethernet Standard, 1997.

[3] Alteon Co., http://www.alteon.com.

[4] Packet Engines Co., http://www.packetengines.com.

[5] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser, "Active Messages : a Mechanism for Integrated Communication and Computation," Proceedings of the 19th ISCA, May 1992.

[6] Scott Pakin, Vijay Karamcheti, and Andrew Chien, "Fast Messages : Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors," IEEE Concurrency, 1997.

[7] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels, "U-Net : A User-Level Network Interface for Parallel and Distributed Computing," Proceedings of the 15th ACM SOSP, December 1995.

[8] Andrew Gallatin, Jeff Chase, and Ken Yocum, "Trapeze/IP: TCP/IP at Near-Gigabit Speeds," Proceedings of 1999 USENIX Technical Conference, June 1999.

[9] Gerard J. Holzmann, "The Model Checker SPIN," IEEE Transactions on Software Engineering, May 1997.

[10] Gerard J. Holzmann, Design and Validation of Computer Protocols, Prentice Hall, 1991.

[11] A. Pnueli, "The Temporal Logic of Programs," Proceedings of the 18th IEEE symposium Foundations of Computer Science, 1977.