TCP/IP Hardware Accelerator를 위한 TCP Engine 설계

준회원 이 보 미, 정 여 진, 정회원 임 혜 숙*

TCP Engine Design for TCP/IP Hardware Accelerator

Bomi Lee, Yeojin Jung *Associate Member*, Hyesook Lim* *Regular Member*

요 약

Transport Control Protocol (TCP)은 소프트웨어로 구현되어 네트워크로 입출력되는 데이터를 처리하는 역할을 한다. 네트워크 기술의 향상으로 CPU에서 수행되는 TCP의 처리가 새로운 병목점으로 등장하고 있다. 또한 iSCSI와 같은 Storage Area Network (SAN) 에서도 TCP의 고속 처리가 전체 시스템의 성능을 결정하는 주요 관건이 되고 있다. 이러한 TCP를 하드웨어로 구현할 경우, 엔드 시스템에서의 CPU의 부하를 줄이고, 고속의 데이터 처리가 가능하여진다. 본 논문에서는 TCP의 고속 처리를 위한 전용 하드웨어 엔진에 관하여 다룬다. TCP 하드웨어는 TCP Connection을 담당하는 블럭과 Receive Flow 를 위한 Rx TCP 블럭, Transmit Flow를 위한 Tx TCP 블럭으로 구성된다. TCP Connection 블럭은 TCP connection 상태를 관리하는 기능을 수행한다. Rx TCP 블럭은 네트워크로부터 패킷을 받아 헤더와 데이터 처리를 담당하는데, 헤더 정보를 parsing 하여 전달하고, 데이터를 순서에 맞게 조립하는 역할도 담당한다. Tx TCP 블럭은 CPU로부터 온 데이터를 패킷을 만들어 네트워크로 전송하는 기능, 신뢰성 있는 데이터 전송을 위한 재전송 기능, Transmit Window 의 관리와 Sequence Number를 생성, 관리하는 기능을 담당한다. TCP 하드웨어 엔진을 검증하기 위한 여러 가지 Testcase들이 수행되었으며, 구현된 TCP 전용 하드웨어 엔진을 0.18 마이크론 기술을 사용하여 Synthesis 한 결과, 입출력 데이터를 저장하기 위한 버퍼를 제외하고, 51K 게이트가 소요됨을 보았다.

Key Words : TCP/IP, Hardware Accelerator, TCP Connection Management, Sequence number Management, Reliable Transfer

ABSTRACT

Transport Control Protocol (TCP) has been implemented in software running on CPU in end systems, and the protocol processing has appeared as a new bottleneck due to advanced link technology. TCP processing is a critical issue in Storage Area Network (SAN) such as iSCSI, and the overall performance of the Storage Area Network heavily depends on speed of TCP processing. TCP Engine implemented in hardware reduces the load of CPU in end systems as well as accelerates the protocol processing, and hence high speed data processing is achieved. In this paper, we have proposed a hardware engine for TCP processing. TCP engine consists of three major blocks, TCP Connection block, Rx TCP block, and Tx TCP block. TCP Connection block is responsible for managing TCP connection states. Rx TCP block is responsible for receive flow which receives packets from network and sends to CPU. Rx TCP performs header and data processing and sends header information to TCP connection block and Tx TCP block. It also assembles out-of-ordered data to in-ordered before it transfers data to CPU. Tx TCP block is responsible for transmit flow which transfers data from CPU to network. Tx TCP performs retransmission for reliable data transfer and management of transmit window and sequence number. Various test-cases are used to verify the TCP functions. The TCP Engine is synthesized using 0.18 micron technology and results in 51K gates not including buffers for temporal data storage.

* 이화여자대학교 정보통신학과 SoC Design 연구실 (hlim@ewha.ac.kr)
논문번호 : 040090-0225, 접수일자 : 2004년 2월 26일
※본 연구는 삼성전자의 지원으로 수행되었습니다.

Ⅰ. Introduction

인터넷은 세계 각지의 엔드 시스템들을 연결하여 데이터 교환이 가능하도록 한 컴퓨터 네트워크로, 지난 10년간 매년 두 배에 가까운 빠른 성장을 거듭하고 있다. 인터넷의 성장은 글로벌 네트워크를 형성하고 있는 엔드 시스템의 수 적인 증가뿐만 아니라 네트워크가 제공하는 서비스나 응용 프로그램의 트래픽 증가도 함께 하는데, 대표적인 서비스로 웹, 이메일 서비스, 전자 상거래, 인터넷 전화나 Video on Demand (VOD), 멀티미디어 서비스 등이 있다. 또한 이더넷으로 연결되어 있는 링크 속도가 gigabit이상으로 빨라짐에 따라 엔드 시스템에서의 데이터 처리는 네트워크의 새로운 병목점으로 등장하고 있다. 과거에는 네트워크 속도가 병목이었으나 현재는 optical 기술의 발달로 link 속도는 OC-192 (10Gbps)까지 증가하였으며 multimedia application등의 개발로 user가 원하는 데이터 량이 급격히 증가하여 엔드 시스템에서의 데이터 처리가 병목이 되고 있다. 따라서 네트워크의 성능을 향상시키기 위해 엔드 시스템에서의 데이터 처리 속도를 빠르게 할 수 있는 연구가 필요하다 [1].

현재 iSCSI와 같은 storage network을 위한 효율적인 데이터 전달에 대한 연구가 활발히 진행되고 있으며 storage network에서는 TCP/IP 프로토콜의 고속 처리가 주요 관건이 된다 [2]. TCP/IP 프로세싱은 엔드 시스템에서의 데이터 프로세싱의 많은 부분을 차지하므로 TCP/IP 프로토콜을 하드웨어로 설계할 경우 CPU의 부하를 줄일 수 있으며 프로세싱 속도를 증가시킬 수 있다 [3]. TCP/IP 프로토콜의 하드웨어 구현은 현재 소프트웨어로 구현되어 쓰이는 TCP/IP 프로토콜과의 호환성을 고려하여 연구되었다. [1] 논문은 TCP/IP를 하드웨어로 구현하는데 있어 TCP의 많은 기능 중 부하가 많이 걸리는 체크섬 계산, 메모리 접근 그리고 버퍼관리 기능만을 하드웨어로 구현한 반면 본 논문은 congestion control을 제외한 TCP의 모든 기능을 하드웨어로 구현하였다.

본 논문은 '고속의 데이터 처리를 위한 TCP/IP hardware accelerator 구현' 프로젝트의 일부분으로 설계된 TCP 하드웨어 엔진을 다룬다. 본 논문은 II장에서 하드웨어로 구현한 TCP 엔진의 전반적인 동작에 대해 설명한다. III, IV, V장에서는 각각 TCP Connection Control, Receive(Rx) TCP, Transmit(Tx) TCP의 기능에 대해서 살펴본다. 그리고 VI장에서는 TCP 블록의 testcases에 대해 살펴본 후 VII장에서는 TCP Engine의 synthesis 결과를 살펴보고 VIII장에서는 간단히 결론을 맺는다.

Ⅱ. Overview

TCP는 packet-switch 네트워크 상에서 엔드 시스템 사이의 reliable한 데이터 전송을 가능하게 하는 프로토콜이다. Reliable한 데이터 전송은 네트워크 상에서 손상되거나 중복되거나 유실된 패킷들의 존재를 알아내고 이러한 패킷이 완전히 전송 될 수 있도록 하는 것이다. TCP는 reliable한 데이터 전송을 가능하게 하기 위해 error detection, re-transmission, cumulative acknowledgment, timer, sequence number and acknowledgment number등의 요소를 갖는다 [4]. 또한 TCP는 connection-oriented 서비스를 제공한다. 두 엔드 시스템의 프로세스가 서로 통신하기를 원하는 경우, 데이터 전송 전에 handshake 동작을 수행하여 connection을 형성한다. Connection을 형성한 TCP는 connection에 해당되는 window size, sequence number등의 정보를 기억한다. TCP connection은 circuit switching이나 virtual circuit에서의 connection과는 달리 직접 연결되지 않은 두 개의 엔드 시스템에서만 적용되는 개념으로 중간에 위치한 라우터나 스위치는 이러한 connection의 존재를 알지 못한다 [5].

TCP는 그림1과 같은 구조의 패킷을 사용한다. TCP 패킷의 헤더는 reliable 데이터 전송을 위한 sequence number, acknowledgment number field를 가지고 있다. Port number는 패킷이 전달될 application을 지정하는데 쓰며 flow control을 위한 receive window field를 포함하고 있다. Flag field는 FIN, SYN, RST, ACK등의 field들을 가지고 있는데 패킷의 종류에 따라 해당 field를 1로 만든다. Checksum field는 헤더와 데이터의 합의 1's complement인 값을 가지는데 패킷의 에러 여부를 판단하는데 쓰인다.

그림 2는 TCP/IP Hardware Accelerator의 전체적인 블럭 다이아그램을 보여주며 빗금친 블럭들이 TCP 엔진에 포함된다. Receive Direction의 데이터 흐름은 MAC으로부터 Packet Parser를 거쳐 들어온 패킷을 Rx TCP가 처리하여 Rx TCP Buffer에 저

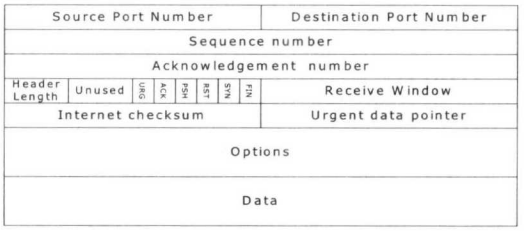


그림 1. TCP 헤더 구조

장하고, Rx TCP Buffer에 저장된 데이터를 Host Interface가 외부 RAM에 전달하여 CPU가 읽어가도록 한다. Transmit Direction은 CPU가 전송하고자 하는 데이터를 외부 RAM에 저장하면 그 데이터를 Host Interface가 읽어 Tx TCP Buffer에 저장한다. 이렇게 저장된 데이터는 Tx TCP에서 패킷으로 만들어져 Network Interface를 통해 MAC으로 전달되게 된다.

TCP 엔진은 그림 3과 같이 three-way handshake를 이용한 connection setup을 담당하는 TCP Connection 블럭, 다른 엔드 시스템으로부터 들어온 패킷을 처리하는 Receive direction TCP 블럭 (Rx TCP) 그리고 다른 엔드 시스템으로 보내기 위한 패킷을 생성하는 Transmit direction TCP 블럭 (Tx TCP)으로 구성된다.

Ⅲ. TCP connection 블럭

3.1 Control

TCP는 데이터 패킷을 전송하기 전에 three-way handshake를 수행하기 때문에 connection-oriented 특성을 갖는다. 이러한 connection-oriented 기능을

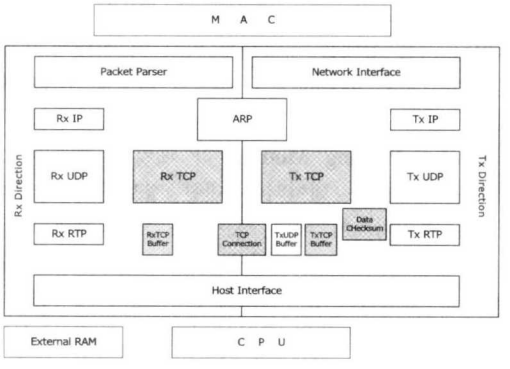


그림 2. TCP/IP Hardware Accelerator 블록 다이아그램

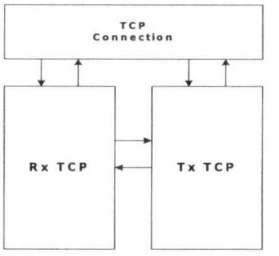


그림 3. TCP 블럭의 구성

수행하는 블럭이 TCP connection 블럭이다. TCP connection 블록은 Host Interface로부터 들어오는 명령들과 Rx TCP로부터 받은 flag field의 타입에 따라 현재 TCP의 state를 결정하고 관리한다. TCP connection 블럭에 구현된 finite state machine은 그림4와 같다.

3.2 Function

TCP Connection 블럭은 CPU의 명령에 따라, 혹은 입력된 패킷의 종류에 따라 TCP의 현재 state에서 다음 state로의 이동을 결정한다. 'CLOSED' state는 connection이 닫혀 있는 상태로 외부에서 오는 패킷을 전혀 받아들이지 않는 state이다. 'SYN-SENT', 'SYN-RCVD', 'LISTEN' state는 멀리 떨어져 있는 엔드 시스템과 connection을 형성하는 state이다. Three-way handshake를 통해 connection이 완전히 열리면 'ESTABLISHED' state가 된다. 'ESTABLISHED' state는 데이터 패킷과 그에 대한 응답 패킷들을 주고받게 된다. 'CLOSE WAIT', 'FIN WAIT-1', 'FIN WAIT-2', 'TIMED WAIT', 'LAST ACK', 'CLOSING' state들은 connection이 닫히는 과정으로 connection이 완전히 닫히면 'CLOSE' state로 돌아가게 된다.

Ⅳ. Rx TCP 블럭

4.1 Overview

Rx TCP는 Packet Parser로부터 패킷을 받아 처리하는 모듈이다. Rx TCP는 data flow와 control flow로 나눌 수 있는데 data flow는 Packet Parser에서 받는 데이터 패킷을 처리하는 부분이고 control flow는 data flow를 제어하고 관리하기 위한 부분이다. 그림 5에서 보이는 바와 같이 data flow에는 패킷의 헤더 처리를 위한 Header Parsing Module(②)과 패킷의 데이터를 전달하기 위한 Data

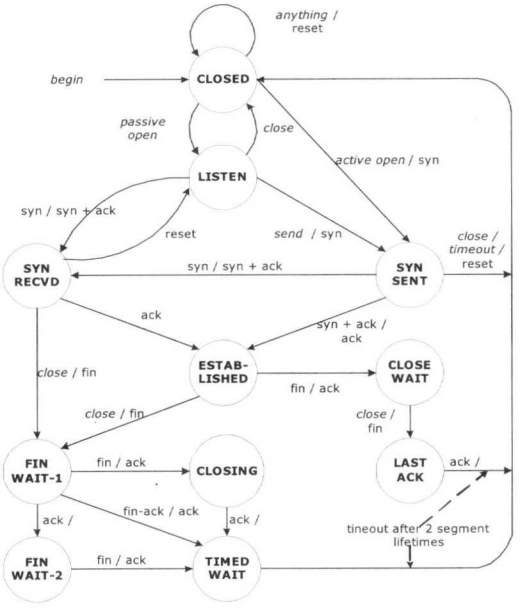


그림 4. TCP Connection 블럭의 FSM

Transfer Module(④)이 있다. Control flow는 Rx TCP의 전체 flow를 제어하는 Rx TCP Control Module(①), 데이터 전달을 제어하는 Data Transfer Control Module(③)과 그밖에 데이터 재조립 및 관련 제어를 위한 Window & Sequence Number Management Module(⑤)을 포함한다.

4.2 Functions

Rx TCP Flow Control Module 그림 5의 ①에 해당하는 'Rx TCP Control Module'은 Rx TCP의 전체 flow를 제어하는 module이다. 이 module은 그림 6의 Rx TCP Control Finite State Machine (FSM)에 따라 Packet Parser로부터 받은 패킷을 처리한다. Rx TCP Control FSM은 매 패킷에 대하여 수행되는 기본적인 flow로, Rx TCP는 이 FSM에 따라 Packet Parser로부터 header valid 신호와 함께 패킷의 헤더를 받기 시작하고, 이어 데이터 부분을 받는다. 패킷의 수신이 끝나면 Rx TCP는 헤더 정보들을 TCP Connection Control 블록과 Tx TCP에게 보내고, in-order로 저장된 데이터가 데이터 전달을 위한 threshold보다 커지는 경우, HI에게 receive interrupt를 보낸다. Rx TCP는 헤더의 sequence number를 판단하여 duplicated 패킷인 경우나 오버플로우인 경우, 패킷을 버리게 된다. 또, Packet Parser가 패킷을 수신하는 도중 생긴 에러로 인하여

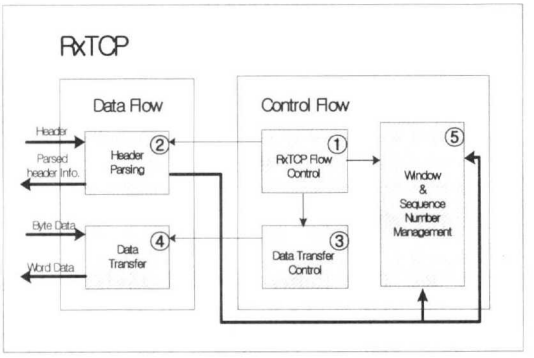


그림 5. Rx TCP 블럭다이아그램

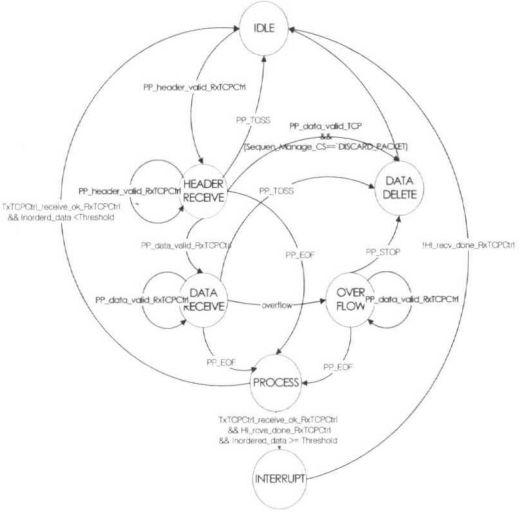


그림 6. Rx TCP FSM

패킷을 버려야 하는 경우, toss 신호를 보내는데, 이 신호를 받은 경우에도 패킷을 버리게 된다. Rx TCP control flow는 패킷을 수신함에 있어 Rx TCP의 기본 상태를 제어하게 되고 이 상태에 따라 그림 5의 다른 module들도 적절한 기능을 수행하게 된다.

Header Parsing Module 각 패킷의 헤더는 그 패킷이 전달하는 데이터에 대한 정보들로 패킷의 처리에 쓰인다. 이 헤더 정보들은 그 성격에 따라 각기 다른 module로 전달되어야 한다. 'Header Parsing Module(그림 5의 ②)'은 패킷의 헤더를 받아 parsing하고 패킷의 수신이 끝나면, 다른 module들에게 parsing한 헤더 정보를 보내는 일을 수행한다. Rx TCP가 parsing한 헤더 정보들이 전달되는

module에는 TCP Connection Control 블록과 Tx TCP, HI가 있다. Rx TCP는 매 패킷에 대하여 TCP의 state를 결정하는 데 쓰이는 flag 신호들을 TCP Connection Control 블록에게 보내고, Tx TCP에게는 전송할 패킷의 헤더에 쓰이는 정보들을 보내는데, acknowledgment number, sequence number, receive window size 등이 여기에 해당된다. HI에게는 패킷의 헤더로부터 Source/Destination Port Number와 데이터가 저장된 buffer의 주소와 저장된 데이터의 크기를 알려주는데, 이 정보들은 패킷 단위로 이루어지지 않고 in-order로 저장된 데이터가 threshold를 넘어가서 HI에게 receive interrupt를 주는 경우에만 알려주게 된다.

Data Transfer Control Module 'Data Transfer Control Module(그림 5의 ③)'은 data flow의 'Data Transfer Module'의 데이터 전달이 적절하게 이루어지도록 제어하는 기능을 담당한다. Packet Parser에서 Rx TCP로 들어오는 데이터는 byte 단위로 들어오는데 반해, Rx TCP는 Rx TCP Buffer에 word 단위로 데이터를 쓰게 된다. 따라서 Packet Parser에서 들어오는 byte 데이터를 word 데이터로 조립하고 적절한 위치에 저장해야 한다. Data Transfer Control Module은 그림 5의 ④의 'Data Transfer Module'이 이러한 기능을 잘 수행하도록 상태를 제어해 준다. 그림 7은 Data Transfer Control FSM으로 크게 주소를 체크하고(ⓐ) Rx TCP Buffer로부터 데이터를 prefetch하고(ⓑ) byte 데이터를 word 데이터로 조립하는 상태로 나뉜다(ⓒ).

Rx TCP는 out-of-order로 들어오는 패킷의 데이터들의 재조립을 용이하게 하기 위해서 데이터를 Rx TCP Buffer에 저장할 때, sequence number와 Rx TCP Buffer의 byte 주소를 1대 1로 일치시킨다. 따라서 데이터가 저장되는 Rx TCP Buffer의 word 주소는 데이터의 sequence number와 initial sequence number의 차이 값을 4로 나눈 값을 사용된다. 이와 같은 방식의 경우, sequence number와 initial sequence number와의 차이값이 4의 배수로 시작하여 4의 배수로 끝나는 경우에는 데이터가 완전한 word로 조립되어 Rx TCP Buffer에 저장되게 되지만, 그렇지 못한 경우에는 불완전한 word로 조립되게 된다. Rx TCP로 들어오는 패킷은 반드시 in-order로 들어오는 것이 아니기 때문에 현재 수신한 패킷의 시작 워드 또는 끝 워드의 일부분이 Rx TCP Buffer에 저장되어 있는 경우가 있을 수 있으므로, 불완전한 데이터가 저장될 주소의 Rx TCP

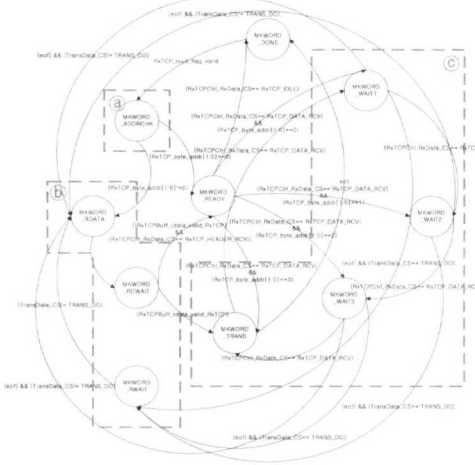


그림 7. Data transfer Control FSM

Buffer의 데이터를 읽어와 불완전한 워드와 조립하여 저장하게 된다. ⓐ에서는 byte주소를 이용하여 Rx TCP Buffer의 존재하는 데이터와 수신된 데이터와의 조립이 필요한지를 판단하게 된다. Data Transfer Control FSM의 ⓑ는 ⓐ에서 데이터 조립이 필요하다고 판단되어 Rx TCP Buffer로부터 데이터를 읽어오기 위한 상태들이다. 이와 같은 과정은 데이터 수신에 앞서 헤더를 수신하는 과정에서 sequence number가 들어오면 바로 수행되고 데이터가 수신되기 시작하면 ⓒ의 상태들을 차례로 거치면서 byte 데이터를 word 데이터로 조립한다. ⓒ의 4개의 상태들은 데이터의 byte 주소에 따라 각 byte 데이터가 word 데이터의 어느 위치에 조립되는지를 제어한다. 그림 8은 위에서 설명한 불완전 데이터를 조립하는 과정을 보이고 있다.

Data Transfer Module 'Data Transfer Control Module'의 상태에 따라 데이터를 조립하여 저장하고 있다가 6000byte(word * 1500 entries) 크기의 Rx TCP Buffer에 저장하는 module이 그림 5의 ④에 해당하는 'Data Transfer Module'이다. Packet Parser에서 받은 패킷의 byte 데이터들은 이 module에서 word 데이터로 조립된뒤 한번에 4 word씩 Rx TCP Buffer에 저장된다. Rx TCP가 데이터를 Rx TCP Buffer에 저장하고 HI에게 interrupt를 보내면, HI는 Rx TCP Buffer에서 데이터를 읽어가서 External RxRAM에 데이터를 저장하게 된다. Rx TCP 내부에는 byte 데이터를 word 데이터로 조립하기 위한 레지스터 외에는 데이터를 버퍼링하기

위한 레지스터를 가지고 있지 않다. 따라서 데이터를 손실없이 저장하기 위해서 Rx TCP는 Rx TCP Buffer의 액세스에 있어서 HI보다 우선권을 가지게 된다. HI는 Rx TCP Buffer에 액세스하기 위해서 Rx TCP Buffer Controller와의 handshaking이 필요하지만 Rx TCP가 Rx TCP Buffer에 데이터를 쓰거나 읽기를 원할 때는 언제든지 Rx TCP Buffer를 액세스할 수 있도록 하였다.

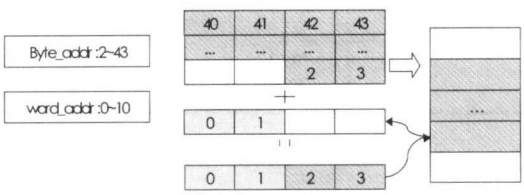


그림 8. Word Assembling procedure

Window & Sequence Number Management Module 그림 5의 ⑤에서는 receive window management 및 out-of-order로 들어오는 패킷을 in-order로 조립하기 위해 sequence number를 관리하는 기능을 수행한다. 본 논문에서 구현된 Rx TCP는 Rx TCP Buffer의 주소 범위를 receive window로 하여 수신된 데이터의 sequence number와 Rx TCP Buffer의 주소를 1대 1로 일치시킴으로써 패킷의 재조립 과정을 용이하도록 하였다. 따라서 receive window의 초기값은 Rx TCP Buffer의 크기인 6000byte와 일치하고 데이터를 수용할 수 있는 receive window의 크기는 Rx TCP Buffer의 크기에서 in-order로 저장된 데이터의 크기를 뺀 값이 된다. Rx TCP가 HI에게 Rx TCP Buffer에서 읽어가야 하는 데이터의 크기를 알려주면 HI는 외부 RAM의 공간을 고려하여 데이터를 읽어가고 그 응답으로써 읽어간 데이터의 크기를 알려주게 된다. Rx TCP는 HI가 읽어간 데이터의 크기를 사용하여 start_ack_number를 갱신하게 된다. End_ack_number와 start_ack_number의 차이가 Rx TCP Buffer에 in-order로 저장된 데이터가 되므로 start_ack_number를 갱신하면 HI가 읽어간 데이터가 저장된 Rx TCP Buffer 공간이 비게 됨을 의미하게 된다. 따라서 Rx TCP는 Rx TCP Buffer의 read address와 write address를 별도로 관리할 필요가 없게 된다. 또 TCP는 Cumulative ACK를 사용하는데, 이것은 패킷의 acknowledgment number를 받으면, acknowledgment number 바로 앞의 sequence number를 가지는 데이터까지 모두 제대로 수신했음을 의미한다. 따라서 out-of-order로 수신되었다가 in-order와의 갭이 채워진 경우를 포함하여 in-order인 모든 패킷에 대하여 acknowledgment를 보낼 필요없이 in-order인 데이터의 end_ack_number에 대해서만 acknowledgment를 보내면 된다. Rx TCP는 in-order인 데이터의 end_ack_number+1을 Tx TCP에게 알려주어 전송하는 패킷의 acknowledgment number로 사용하도록 한다.

Sequence number와 Rx TCP Buffer의 주소를 일치시켰기 때문에 Rx TCP는 들어오는 패킷의 in-order, out-of-order의 여부에 상관없이 데이터의 sequence number에 해당하는 위치에 데이터를 저장하게 된다. 그러나 HI로 전달되는 데이터는 in-order인 데이터만을 전달하게 되므로 Rx TCP Buffer에 저장된 데이터 중 어디까지가 유효한 in-order 데이터인지 알아야 하고 out-of-order로 된 데이터에 대한 정보도 관리하고 있다가 새로 받은 데이터가 in-order와 out-of-order인 데이터 사이에서의 갭을 채우는지를 판단하여 in-order데이터에 대한 정보를 갱신하여야 한다. 이와 같은 Sequence Number Management를 위하여 'Window & Sequence Number Management Module'은 내부에 sequence number management table을 가진다. Rx TCP는 최대 10개까지 out-of-order로 들어온 패킷의 정보를 sequence number management table에 저장할 수 있으며, sequence number management를 위해서 sequence number management table의 엔트리 중 몇 개를 사용할 것인지는 HI의 CSR을 통해서 CPU가 설정할 수 있다. Sequence number management는 그림 9의 FSM을 통해 수행되는데 그 과정은 다음과 같다. 우선 패킷의 헤더를 parsing하여 얻은 sequence number와 TCP Connection state로 판단하여(그림 9의 ⓐ) 들어온 패킷이 SYN 패킷의 경우에는 이 패킷의 sequence number를 initial sequence number(그림 9의 ⓑ)로 정한 후, number processing을 끝내고 그렇지 않은 경우에는 end_ack_number와 in-order인지 비교한다(그림 9의 ⓒ). End_ack_number와 in-order인 경우는 그림 11의 ⓓ에서, sequence number management table에서 이 패킷의 sequence number와 in-order인 것이 있는지 찾는다. 즉, 수신된 데이터가 end_ack_number와 sequence number table에 저장된 엔트리 사이의 갭을 채우는 경우를 찾아보는 것이다. 수신된 데이터가 end_ack_number와 out-of-

order로 판단되면(그림 9의 ⓒ) 그림 9의 ⓔ에 따라 우선 sequence number management table의 엔트리 중 in-order인 것이 있는지 혹은sequence number management table의 엔트리들 사이의 갭을 채우는지를 확인하고 위의 어떤 경우에도 해당되지 않는 경우에는 빈 엔트리에 저장한다.

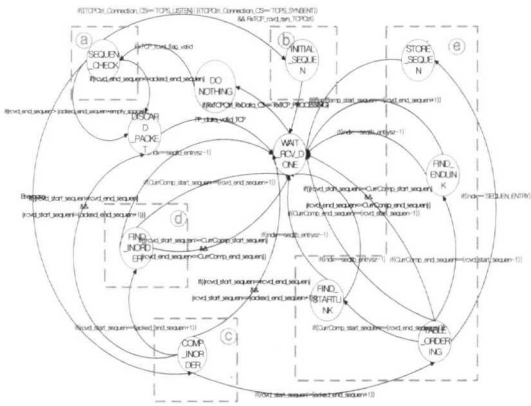


그림 9. Sequence Number Management FSM

그림 10은 그림 9의 Sequence Number Management FSM에 따라 Sequence Number Management Table에서 sequence number가 어떻게 관리되는지를 설명한 그림이다. 그림 10의 좌측의 표와 같이, ①->⑥의 순서로 다음과 같은 sequence number를 가지는 패킷이 들어올 때 Sequence Number Management Table의 entry와 Acked End Sequence Number가 어떻게 변하는지를 보이고 있다. 우선 ①, ②번 패킷은 in-order로 들어온 것이기 때문에 Sequence Number Management Table에 sequence number가 저장되지 않고 Acked End Sequence Number만이 업데이트 된다. 그 후에 들어오는 ③, ④번 패킷은 앞의 데이터들과 out-of-order이면서 ③, ④번 sequence number간에도 갭이 존재하기 때문에 Sequence Number Management Table에 서로 다른 엔트리로서 저장된다. ⑤번 패킷은 Acked End Sequence Number와는 out-of-order이지만, ③, ④번 sequence number 사이의 갭을 채우므로 다른 엔트리로서 저장되지 않고 첫번째 엔트리 를 갭이 채워진 sequence number로 업데이트 시키고 두번째 엔트리를 무효화시킨다. ⑥번 패킷처럼 Acked End Sequence Number와 Sequence Number Management Table에 저장된 엔

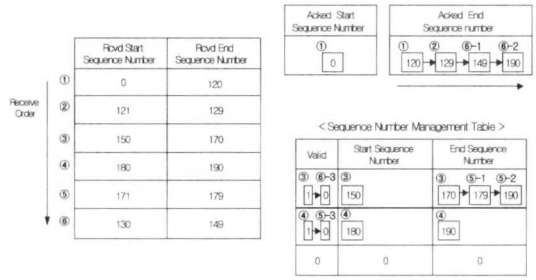


그림 10. Sequence Number Management

트리 사이의 갭이 채워지는 경우 우선 Acked End Sequence Number가 업데이트 되고 이어 Acked End Sequence Number와 in-order인 엔트리를 Sequence Number Management Table에서 찾아 Acked End Sequence Number를 다시 업데이트 시키고 Sequence Number Management Table의 해당 엔트리는 무효화된다.

V. Tx TCP

5.1 Overview

Tx TCP 블럭은 TCP protocol중 transmission을 담당하는 부분으로 크게 패킷을 생성하는 동작과 reliable한 데이터 전송을 위한 동작들을 수행한다. Rx TCP 블럭에서 전달된 받은 패킷의 정보와 Host Interface에서 전달된 패킷의 정보를 조합하여 각각의 상황에 적절한 패킷을 생성한다.

5.2 Control

그림11은 Tx TCP의 Finite State Machine을 보여주고 있다. Tx TCP 블럭은 TCP protocol의 transmit 기능을 담당한다. Tx TCP 블럭은 'SET_HEADER_INFO' state에서 전송 패킷의 헤더 정보를 수집하여 헤더를 생성하는 동작을 수행한다. 'SET_ DATA_INFO' state에서는 패킷의 데이터 정보를 이용하여 Tx TCP Buffer로부터 패킷의 데이터를 읽어 Tx TCP FIFO에 저장하는 동작을 수행한다. 'FAST_RETRANS' state는 duplicated ack를 3번 받은 경우 해당 패킷을 재전송하기 위해 패킷의 정보를 모으는 state이다. 'SEND_SEG' state에서는 생성된 패킷이나 재전송을 위한 패킷을 Network Interface 블럭으로 1byte씩 연속적으로 전달하는 동작을 수행한다.

5.3 Function

Tx TCP 블럭은 패킷 생성 기능과 전송한 패킷들의 reliable 전송을 보장하기 위한 기능 등을 수행한다. Tx TCP 블럭의 기능 중 하나는 각 상황에 맞는 타입의 패킷을 생성하는 것이다. Tx TCP 블럭에서 생성되는 패킷의 종류를 살펴보면

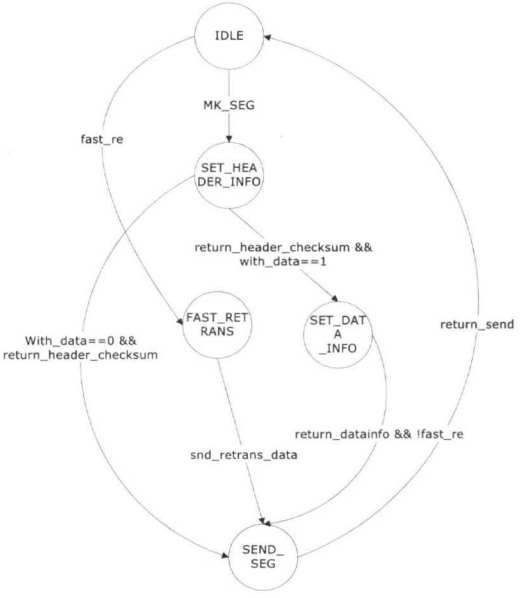


그림 11. Tx TCP FSM

- SYN packet
- SYN-ACK packet
- ACK packet
- ACK+DATA packet
- DATA packet
- FIN packet
- FIN-ACK packet
- RST packet

이 있다.

다른 하나의 기능은 reliable 데이터 전송을 보장하는 것이다. 이를 위해 수행되어야 하는 동작들을 살펴보면

- Duplicated acknowledgment 처리
- Fast retransmit 수행
- Transmit window 관리

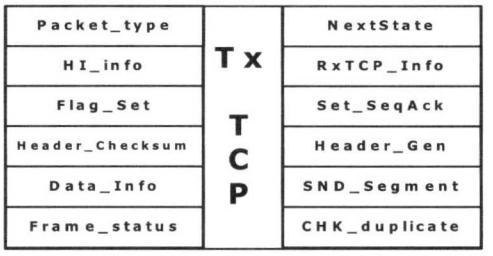


그림 12 Tx TCP의 기능별 sub-module

- Delayed acknowledgment 수행
- Sequence number 생성, 관리

가 있다.

Tx TCP 블럭은 위의 대표적인 두 가지 기능을 수행하고 추가적으로 데이터 checksum 블럭을 가진다. 데이터 checksum 블럭은 Tx TCP 블럭에서 직접 데이터 checksum을 계산하는 경우 Host Interface 블럭이 외부 RAM에서 읽어온 데이터를 내부 Tx TCP buffer에 쓰기 위한 메모리 액세스 한번, Tx TCP 블럭이 데이터 checksum 계산하기 위한 메모리 액세스 한번 그리고 Tx TCP 블럭이 데이터를 전송하기 위한 메모리 액세스 한번이 필요하다. 결국 하나의 패킷을 처리하기 위해 세 번의 메모리 액세스가 필요하게 된다. 이러한 메모리 액세스 횟수를 줄이기 위해 Tx TCP 블럭 밖에 데이터 checksum 블럭을 두었다. 데이터 checksum 블럭은 Host Interface 블럭이 Tx TCP buffer에 쓰는 데이터를 중간에 받아 데이터 checksum을 계산하게 된다. 이를 위해 Host Interface에서 데이터 segmentation을 수행하고 Host Interface 블럭은 하나의 패킷에 대한 데이터를 전송하는 동안 패킷 valid signal을 set하여 데이터 checksum 블럭이 패킷에 대한 checksum을 계산하게 한다. Tx TCP에서 수행되는 기능들은 그림12에서 볼 수 있듯이 몇 개의 sub-module들로 구성되어 있다.

Packet Type Module Connection setup을 위해 생성하여야 하는 패킷의 타입을 결정한다. Connection setup을 위한 패킷에는 syn packet, syn-ack packet, ack packet, rst packet, fin packet이 있다. 이러한 패킷 타입의 결정은 TCP 헤더의 flag field에 표현한다. TCP Connection module은 TCP의 state와 받은 패킷의 타입 또는 Host Interface 블럭으로부터의 명령에 따라 connection 패킷의 타입을 결정하게 된다.

Set_SeqAck Module 전송할 패킷의 타입이 결정

되면 패킷의 타입에 따라 sequence number와 acknowledge number를 선택한다. Syn 패킷은 initial sequence number를 갖게 되는데 initial sequence number는 Tx TCP에서 random하게 선택하게 된다

Ack 패킷의 acknowledgment number는 Rx TCP 블럭에서 전달받아 사용한다. 데이터 패킷의 sequence number는 next sequence number를 사용하게 되는데 next sequence number는 현재 전송할 패킷의 sequence number에 데이터 길이를 더하여 결정된다.

Header_Gen Module 패킷의 타입과 sequence number, acknowledgment number가 결정되면 헤더 정보들을 이용하여 TCP checksum을 제외한 TCP 헤더를 포맷에 따라 완성한다. "Header_Checksum module"은 TCP 헤더와 TCP pseudo 헤더의 checksum을 계산하고 데이터 checksum 블럭으로부터 데이터 checksum을 받아 TCP 패킷의 checksum을 완성한다.

SND_Segment Module Tx TCP 블럭에서 만든 헤더를 Network Interface로 전송하는 기능을 수행한다. 헤더가 완성되면 Network Interface로 1 bytes씩 전송한다.

Frame Status Module Tx TCP의 reliable 데이터 전송을 위해 transmit window 관리와 duplicated acknowledgment 패킷의 재전송 등의 기능을 수행한다. "Frame Status module"은 위와 같은 기능을 수행하기 위해 전송한 패킷의 sequence number, 데이터 length, buffer address 정보 등을 관리하는 테이블을 가지고 있다. 테이블은 전송한 패킷의 acknowledgment 여부를 나타내는 bit을 가지고 있어 acknowledgment를 받지 못한 패킷은 '1', acknowledgment를 받은 패킷은 '0'의 값을 가지고 있게 된다. 그림13은 Frame Status Table의 예를 보여주고 있다.

Tx TCP의 transmit window는 그림14와 같다. Window size는 전송할 수 있는 패킷의 총 개수와 같다. 전송한 후 acknowledgment를 받은 패킷은 transmit window 밖 앞부분, 전송하였으나 아직 acknowledgment를 받지 못한 패킷은 transmit window의 앞부분, 아직 전송하지 못한 패킷은 transmit window의 뒷부분에 위치하게 된다. Tx TCP 블럭은 window의 첫번째 패킷에 대하여 retransmit timer를 set한다. 이 패킷이 timeout 될

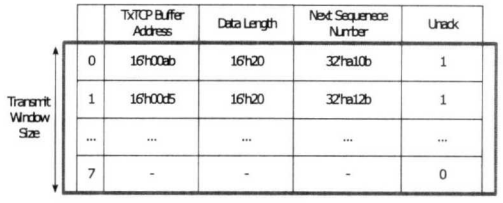


그림 13. Frame Status Table 예

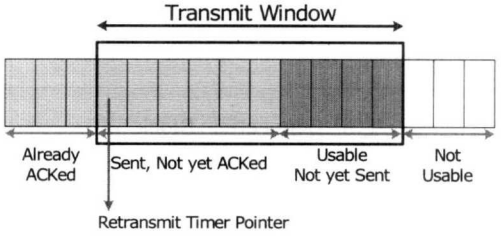


그림 14. Transmit Window

때까지 acknowledgment를 받지 못한다면 Tx TCP 블럭은 이 패킷을 재전송 한다.

"Frame Status Module"은 Transmit Window 관리를 위한 동작과 함께 Delayed acknowledgment 기능을 수행한다. 이는 첫 번째 데이터 패킷에 대한 acknowledgment 패킷을 보내기 위해 delayed acknowledgment timer동안 기다리게 된다. Timer에 기록된 시간 동안 기다리던 중 두 번째 데이터 패킷이 들어온 경우 cumulative ack 특성을 이용하여 두 번째 패킷에 대한 ACK 패킷만을 보내기 위함이다. Timeout 동안 두 번째 데이터 패킷이 들어오지 않은 경우 첫 번째 데이터 패킷에 대한 ACK를 보내게 된다. "Frame status module"은 테이블의 패킷 중 acknowledgment를 받은 패킷의 buffer 끝 address를 Host Interface에 알려주게 된다. 이는 해당 패킷의 데이터를 더 이상 저장할 필요가 없기 때문이다. 또한 "Frame Status Module"은 테이블이 window-1만큼 차면 더 이상 패킷을 전송할 수 없다는 것을 Host Interface 블럭에 알려준다. Tx TCP는 데이터 저장을 위해 6000bytes의 buffer를 가지고 있다.

VI. Testcases

TCP Connection, Rx TCP, Tx TCP 블럭은 표1의 케이스에 대한 테스트를 하였다.

Ⅶ. Synthesis Report

표2는 Synopsys의 Design Analyzer(version 2000.05)와 Samsung 0.18 마이크론 기술(std130)을 이용하여 TCP Connection, Rx TCP, Tx TCP 블럭의 synthesis 결과를 보여준다. Rx TCP Buffer, TxTCP Buffer를 제외한 TCP Engine의 synthesis 결과 51260게이트가 소요됨을 알 수 있었다.

표1. 각 블럭의 testcases

TCP Connection Block
1. TCP 전체 FSM의 각 state 전환
Rx TCP Block
1.TCP state에 따른 헤더 및 데이터 처리 2.다양한 크기의 바이트 데이터를 워드 데이터로 정확히 조립 3.워드 데이터의 첫 워드나 마지막 워드가 불완전한 워드를 형성하는 경우 Rx TCP Buffer에서의 prefetch 및 불완전 워드와의 조립 4.sequence number에 따른 addressing 및 저장 5.다양한 크기의 in-order로 들어오는 패킷들에 대한 처리 6.다양한 크기의 out-of-order로 들어오는 패킷들을 sequence number management table에 저장 7.Rx TCP Buffer Controller와의 동작 8.Duplicated 패킷 및 overflow에 대한 동작
Tx TCP Block
1.Connection setup을 위한 패킷 생성 확인 2.Data 없는 응답 패킷 생성 확인 3.ACK 없는 data 패킷 생성 확인 4.Data + ACK packet 생성 확인 5.Duplicated ACK 패킷 받은 경우 테스트 6.Fast retransmit 수행 7.응답 패킷 받은 경우 8.Retransmit timer가 timeout 된 경우 9.Delayed ack 패킷의 생성 10.헤더 정보가 정확하게 생성되는지 여부

표2. 각 블럭의 Synthesis report(area)

	Area report
TCP Connection	547
Rx TCP	23453
Tx TCP	27260
Total	51260

Ⅷ. Conclusion

멀티미디어 서비스 등 고속의 데이터 처리를 필요로 하는 응용 프로그램의 증가와 링크 속도의 향상으로 엔드 시스템에서의 고속의 데이터 처리가 중요한 이슈로 등장하게 되었다. TCP/IP 프로토콜은 매우 복잡하고 소프트웨어로 구현되어 있어 전체 통신 시스템의 병목점으로 작용하고 있어 TCP/IP 프로토콜의 성능을 향상시키는 방법에 대해 많은 연구가 이루어지고 있다.

본 논문에서는 전체 통신 시스템의 성능 개선을 위한 TCP/IP 프로토콜의 하드웨어구현 중 TCP 엔진의 구현을 설명하였다. TCP 엔진은 TCP Connection을 담당하는 블럭과, RxFlow를 위한 Rx TCP 블럭, TxFlow를 위한 Tx TCP 블럭으로 나누어 설계하였다. TCP Connection 블럭은 HI로부터의 명령과 Rx TCP에서 받은 flag를 이용하여 TCP의 connection state를 관리하는 역할을 하고 Rx TCP와 Tx TCP는 이 state에 따라 동작하게 된다. Rx TCP 블럭은 네트워크로부터 패킷을 받아 헤더와 데이터 처리를 담당한다. 우선 헤더를 parsing하여 TCP Connection 블럭과 Tx TCP 블럭으로 헤더 정보를 주고, out-of-order 데이터는 in-order로 재조립하여 HI로 하여금 외부 Rx RAM에 쓰도록 하였다. Rx TCP 블럭은 Rx TCP Buffer의 주소와 sequence number를 1대1로 일치시켜 재조립이 용이하도록 하였다. Out-of-order인 데이터들을 위해서는 sequence number를 별도로 관리하여 매 패킷이 들어올 때마다 데이터 사이의 갭을 채우는지를 확인하였다. Tx TCP 블럭은 Rx TCP가 받은 패킷에 대한 ack를 전송하고 HI를 통해 전달 받은 CPU가 보내고자 하는 데이터를 패킷으로 만들어 전송하는 역할을 한다. 또한 reliable한 데이터 전송을 위한 재전송 기능과 cumulative acknowledgement 기능, transmit window의 관리와 sequence number의 생성 관리 기능을 수행한다. 또한 데이터 checksum 블럭에서 생성된 데이터 checksum값을 이용하여

TCP 패킷의 checksum을 완성한다.

본 논문에서 제안한 TCP 하드웨어는 chip의 형태로 제작되지 못하여 performance evaluation을 수행할 수 없었지만 기존 소프트웨어로 구현된 TCP를 대신하여 쓰일 경우 전체 네트워크의 성능을 크게 향상시킬 것으로 기대된다. 또한 본 논문에서 제안한 TCP의 하드에어 구현은 앞으로의 TCP/IP hardware 설계를 위한 지침이 될 수 있을 것이다.

참 고 문 헌

[1] 진교홍, 이정태, "고속통신을 위한 TCP/IP 프로토콜의 하드웨어 설계 및 구현", 한국정보과학지, Vol.12, No.1, pp135-153, Feb.1998

[2] "Introduction to iSCSI", http://www.10gea.org

[3] "Introduction to TCP/IP Offload Engine (TOE)", http://www.10gea.org

[4] James F. Kurose, Keith W. Ross, "Computer Networking : A Top-Down Approach Featuring the Internet", Addison Wesley, 2002

[5] rfc793, "Transmission Control Protocol"

이 보 미 (Bomi Lee) 준회원

shoutoi@ewha.ac.kr

2003년 2월 : 이화여자대학교 정보통신학과 학사

2003년 3월 ~ 현재 : 이화여자대학교 정보통신학과 석사과정

<관심분야> Router나 switch등의 Network 관련 SoC설계, TCP/IP관련 하드웨어 설계

정 여 진 (Yeojin Jung) 준회원

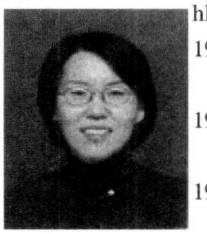
surya@ewha.ac.kr

2003년 2월 : 이화여자대학교 정보통신학과 학사

2003년 3월 ~ 현재 : 이화여자대학교 정보통신학과 석사과정

<관심분야> Router나 switch등의 Network 관련 SoC설계, TCP/IP관련 하드웨어 설계

임 혜 숙 (Hyesook Lim) 정회원



hlim@ewha.ac.kr

1986년 2월 : 서울대학교 제어계측공학과 학사

1986년 8월 ~ 1989년 2월 : 삼성 휴렛 팩커드 연구원

1991년 2월 : 서울 대학교 제어계측공학과 석사

1996년 12월 The University of Texas at Austin, Electrical and Computer Engineering 박사

1996년 11월 ~ 2000년 7월 : Lucent Technologies Member of Technical Staff

2000년 7월 ~ 2002년 2월 Cisco Systems Hardware Engineer

2002년 3월 ~ 현재 : 이화여자대학교 정보통신학과 조교수

<관심분야> Router나 switch등의 Network 관련 SoC설계, 통신관련 SoC 설계