NS-2를 이용한 네트워크 시뮬레이션 방법론

박성현 | 이승형
광운대학교

요 약

본고에서는 현재 일반적으로 가장 널리 사용되고 있는 네트워크 시뮬레이터인 NS-2(Network Simulator-2)에 대하여 알아본다. NS-2는 네트워킹의 성능분석을 목적으로 개발된 이벤트 기반(event-driven)의 시뮬레이터이다. 이 시뮬레이터의 적용 범위는 유선 네트워크의 경우 TCP/IP 프로토콜 패밀리와 각종 라우팅 프로토콜에 대한 시뮬레이션이 가능하며, 무선 네트워크에 경우에는 Ad Hoc 네트워크, WLAN, Mobile-IP와 Cellular network 등의 시뮬레이션이 가능하다. GUI가 부족하고 사용이 어렵다는 단점에도 불구하고, 공개된 소스와 두터운 이용자층에 힘입어 널리 사용되는 NS-2의 개요 및 시뮬레이션 방법에 대해 알아보기로 한다.

1. 서 론

1. NS-2 시뮬레이터 소개

NS-2 시뮬레이터는 TCP, UDP, FTP, HTTP 등과 같은 TCP/IP 프로토콜 패밀리와 라우팅 프로토콜 그리고 멀티캐스팅 프로토콜, RTP, SRN 등과 같은 다양한 인터넷 프로토콜을 시뮬레이션 할 수 있다. 그리고 Ad Hoc 네트워크, 이동통신망의 기지국 모델, WLAN, Mobile-IP 관련 프로토콜, UMTS, 위성 네트워크 등과 같은 무선 네트워크 까지 지원할 수 있는 적용 범위가 매우 넓은 네트워크시뮬레이터이다.

NS-2는 현재 학교나 연구소에서 네트워크를 연구하는 사람들이 가장 선호하고 즐겨 쓰는 시뮬레이터로 알려져 있다. OPNET[6]이나 QualNet[7]과 같은 유료 시뮬레이터의 장점인 뛰어난 GUI와 다양한 라이브러리에도 불구하고 NS-2가 선호되는 이유는 무엇보다도 NS-2의 소스가 공개되어 누구나 사용할 수 있고 사용자층이 두터워서 다양하고 많은 자료와 코드를 이용할 수 있다는 장점이 있기 때문이다. NS-2는 GUI가 불편하고 초보자가 접근하기 어렵다는 단점에도 불구하고, 특히 학교에서의 연구논문 작성에 가장 널리 쓰이는 도구이다.

2. NS-2 시뮬레이터의 탄생 및 성장

NS-2는 콜롬비아 대학에서 개발한 시뮬레이션 테스트베드인 NEST를 기반으로 UC 버클리에서 1988년에 개발한 네트워크 시뮬레이터이다. LBNL(Lawrence Berkely National Laboratory)[8]의 네트워크 연구 그룹에서 네트워킹에 적용할 수 있는 프로그램을 연구하였는데, 이 연구 결과로 발표된 것이 NS-1이라는 시뮬레이터이다.

NBNL에서 개발한 NS-1 시뮬레이터는 확장된 TCL(Tool Command Language) 스크립트 언어를 사용하였다. 1995년에 VINT(Virtual Internetwork Test-bed) 프로젝트의 일환으로 DARPA(Defense Advanced Research Projects Agency)에서 자금을 지원받아, NS-1 시뮬레이터가 완성되었다. 그리고 1996년에 NS-1 시뮬레이터의 기능을 더욱 더 향상시킨 NS-2가 발표되었다. NS-2 시뮬레이터에서는 NS-1에서 사용한 시뮬레이션 언어인 TCL 대신 MIT에서 개발하여 발표한 OTCL(Object TCL)을 사용하여 소스를 코딩하였다. NS-2는

NS-1 시뮬레이터와 완벽한 호환성을 지원하며, 지금도 계속해서 기능이 추가되어 소스 코드가 인터넷상에서 소개되고 있다[5].

NS-1 시뮬레이터는 SYN/FIN 패킷을 지원하지 않는 단방향 TCP (Tahoe, Reno, Vegas, SACK), SYN/FIN 패킷을 지원하는 양방향 TCP, CBQ(Class Based Queueing) 스케줄링 알고리즘, 동적 라우팅 알고리즘, Dense-mode 멀티캐스팅 프로토콜, 트래픽의 흐름 관리자, 텔넷소스, 등에 대한 시뮬레이션을 지원하도록 설계되었다. 이에 비해, NS-2 시뮬레이터는 NS-1 시뮬레이터에서는 지원하지 않았던 다중 경로 라우팅, RTP(Real Time Protocol), 집중형 멀티캐스트, Mobile-IP 프로토콜을 위한 이동 호스트(mobile host) 지원과 같은 기능들이 추가되었다. 이 밖에도 스케줄링 알고리즘도 추가되었는데, 추가된 스케줄링 알고리즘으로는 SFQ(Stochastic Fair Queueing)과 DRR(Deficit Round Robin) 알고리즘 등이 있다.

NS-2 시뮬레이터는 현재도 계속해서 기능이 추가되고 있으며, 성능 개선 작업도 진행되고 있다. 이 프로그램의 최신 버전 및 관련 자료들은 홈페이지에서 구할 수 있으며[5], 학교 및 연구소를 비롯하여 표준화 기술개발 그룹 등의 단체에서 최신 네트워크 기술에 대한 시뮬레이션 코드가 개발되고 있고 이들 소스의 대부분이 공개되어 누구나 자신의 프로토콜과 알고리즘 개발에 활용이 가능하다.

II. NS-2 시뮬레이션

1. 간단한 시뮬레이션 실행

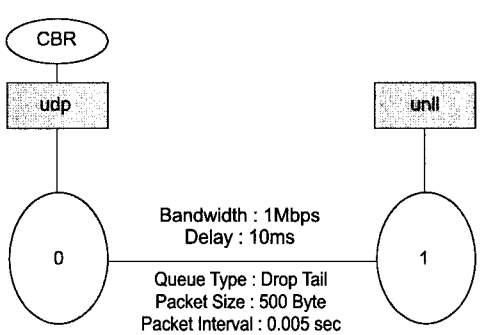
1.1 네트워크 설계

시뮬레이션에서 처음으로 하는 작업이 네트워크 시나리오 작성이다. 네트워크에 대한 시나리오 작업 단계에서 하는 것은 다음과 같다. 먼저 네트워크 토폴로지(topology)를 만들고, 다음으로 네트워크 시나리오(scenario)를 작성한다. 여기에서 네트워크 토폴로지는 노드와 링크의 특성을 결정하는 것이고, 네트워크 시나리오 작업은 노드에서 이용하는 프로토콜을 결정하고 노드에서의 트래픽 에이전트(traffic agent)와 애플리케이션 서비스를 결정하는 것이다.

NS-2 시뮬레이터에서의 트래픽 에이전트의 결정은 전송계층에서 이용할 프로토콜(TCP 혹은 UDP)을 결정하는 것이며, 응용 계층 프로토콜에서 FTP, HTTP, 텔넷, 그리고 CBR과 같은 구체적인 애플리케이션 서비스를 지정해주면 된다. 마지막으로 시뮬레이션 시간 및 후처리(post analysis)를 위한 작업을 지정해주면 된다.

(그림 1)과 같은 네트워크 토폴로지와 네트워크 시나리오에 대하여 소스를 코딩한 후, 이를 시뮬레이션 하는 방법에 대하여 예를 들어보도록 한다. 네트워크 토폴로지에 대한 시나리오는 두 노드를 양방향 링크로 연결하였다. 여기에서, 송신 노드는 0번 노드이고 목적지 노드는 1번 노드로 각각 정하였다. 그리고 양방향 링크의 대역폭은 1Mbps이며, 링크에서의 지연 시간은 10msec이다.

트래픽 발생은 다음과 같다. 0번 노드에서 1번 노드로 인터넷 전화와 같은 음성 트래픽을 발생하도록 하였다. 이를 애플리케이션으로 지정할 때에는 CBR 형태의 트래픽이 전송된다고 보면 된다. 이렇게 발생되는 CBR 트래픽은 UDP 에이전트 위에서 구동하도록 설정하였다. 결국 UDP의 형태의 트래픽이므로 목적지 노드인 1번 노드에서는 ACK 패킷을 만들지 않는다. 그러므로 목적지 노드는 NULL 에이전트로 표현하였다.



(그림 1) 네트워크 토폴로지

시뮬레이션 시간에 대한 설정은 처음 시뮬레이션이 시작한 후 0.5초가 지나면 송신 노드에서 CBR 트래픽을 전송하고 4.5초까지 계속 전송한다. 전체 시뮬레이션 시간은 5초로 하였다. 네트워크 토폴로지와 네트워크 시나리오를 모두 작성하였다면, vi 등의 에디터를 이용하여 스크립트 소스를 작

성하면 된다.

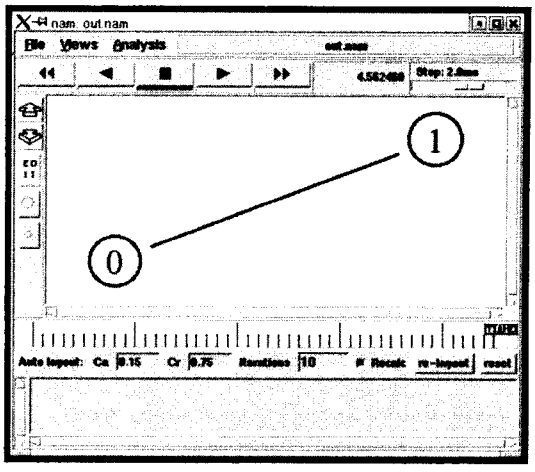
1.2 예제 스크립트

```
#1  set ns [new Simulator]
#2
#3  set tracdfd [open out.tr w]
#4  $ns trace-all $tracefd
#5
#6  set n0 [$ns node]
#7  set n1 [$ns node]
#8
#9  $ns duplex-link $n0 $n1 1Mb 10ms DropTail
#10
#11 set udp0 [new Agent/UDP]
#12 $ns attach-agent $n0 $udp0
#13
#14 set cbr0 [new Application/Traffic/CBR]
#15 $cbr0 set paqcketSize_ 500
#16 $cbr0 set interval_ 0.005
#17 $cbr0 attach-agent $udp0
#18
#19 set null0 [new Agent/Null]
#20 $ns attach-agent $n1 $null0
#21
#22 $ns connect $udp0 $null0
#23
#24 $ns at 0.5 "$cbr0 start"
#25 $ns at 4.5 "$cbr0 stop"
#26
#27 $ns run
```

(그림 2) NS-2 시뮬레이션 스크립트의 예

우선 라인 1에서 시뮬레이터 오브젝트(object)를 생성하고, 다음으로 추적파일을 생성하며 결과는 out.tr 파일에 저장한다. 라인 6과 7에서 시뮬레이션에 사용 될 노드 두 개를 생성하며, 라인 9에서 노드와 노드간의 양방향 링크를 생성한다. 링크의 대역폭은 1Mbps이고 전송 지연은 10msec로 정하였다. 이때, 노드에서 큐(queue)의 타입은 DropTail 큐를 이용한다. 라인 11에서 UDP 에이전트를 생성하고 이를 노드0에 붙이며, 라인 14에 CBR 트래픽 소스를 생성하여 UDP0에 붙인다. CBR 트래픽 소스의 패킷 크기는 500 바이트이고 패킷 간 시간 간격은 5msec로 세팅하였다. 라인 19에서 UDP 트래픽에 대한 싱크(sink)로 Null 에이전트를 생성하고 노드1에 붙인다. 결과적으로 노드1이 수신 노드 역할을 하게 된다. 라인 22에서 트래픽 소스(udp0)와 트래픽 싱크(null0)를 연결시킨다. 라인 24와 25에서 CBR 에이전트에 대한 이벤트를 스케줄하고 마지막으로 27 라인에서 시뮬레이션을 실행하는데, 이 명령은 스크립트의 마지막 부분에 위치시킨다.

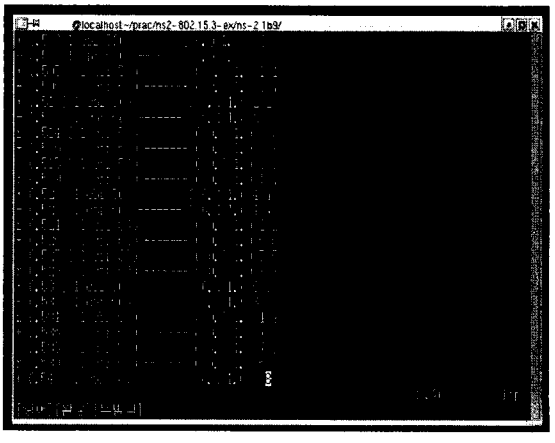
1.3 시뮬레이션의 결과 고찰



(그림 3) NAM을 이용한 시뮬레이션 수행

(그림 2)와 (그림 3)은 리눅스에서 간단한 스크립트 소스를 실행시켰을 때 디스플레이 화면에 나타나는 NAM 디스플레이 장치와 시뮬레이션 결과를 보인다. NS-2 시뮬레이터에는 NAM(Network Animator)이라는 애니메이션 툴이 제공되는데, 이 프로그램은 스크립트로 작성한 소스 코딩에 대한 시뮬레이션 실행, 결과 등을 디스플레이 장치에 직접 보여주는 기능을 제공한다[4].

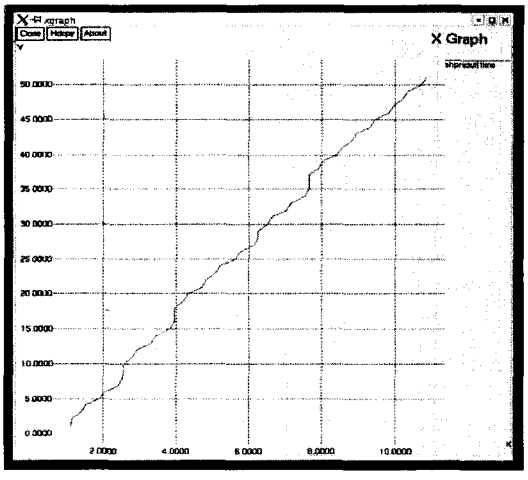
NS-2 시뮬레이터를 이용하면, 패킷 흐름을 눈으로 직접 볼 수 있는 NAM 외에도, 시뮬레이션 결과를 추적 파일로 얻을 수 있다. 즉, 패킷 흐름(packet flow)을 아주 짧은 시간 단위



(그림 4) 패킷의 흐름 추적

로 나누어 그때마다의 상태를 추적 할 수 있는 trace-all 기능이 있어서 사용자가 원하는 네트워크 성능 분석 자료를 제공하고 있다. (그림 4)에서와 같이, 시뮬레이션의 시작 시점부터 종료 시점 까지 시뮬레이션을 하면서 발생되는 패킷의 흐름이 모두 기록된다.

trace-all의 내용은 설계한 네트워크에서 패킷의 흐름을 추적한 결과를 시간대 별로 나누어 정리했기 때문에 매우 자세한 자료가 될 수 있지만, 성능의 변화 및 차이를 한눈에 보기는 힘들다. 이를 시각적으로 표시하기 위해서는 Xgraph 등의 그래프 도구를 사용하면 된다.

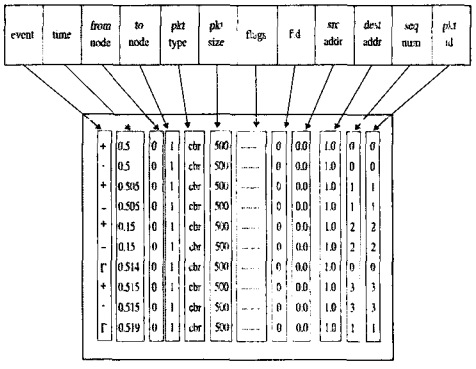


(그림 5) Xgraph 예

(그림 5)는 tr파일에 기록된 많은 trace결과 중 일부분만 발췌한 것으로, 이를 보면서 trace파일에서 패킷 흐름에 대한 추적 결과를 보는 방법을 설명한다. tr 파일에서 이용한 trace 포맷과 각 필드의 의미는 다음 (그림 6)과 같다.

먼저, event가 표시되는 부분이다. 맨 앞에 위치하고 있으며 "+, -, r, d"로 각각 표시된다. 여기에서, "+"는 큐에 들어오는 패킷을 의미하고 "-"는 큐에서 나가는 패킷을 의미한다. 그리고 "r"은 패킷이 다른 노드에 도착했음을 나타낸다. 이 밖에도 큐에서 드롭(drop)된 패킷은 "d"로 표시하는데, 이는 곧 폐기한 패킷을 의미한다.

event 다음에 있는 필드는 time을 나타내는 부분인데, 이는 패킷의 이동시간을 표시한다. 그리고 from/to node는 패킷의 이동 경로를 나타내며, pkt type은 패킷의 타입을 나타내



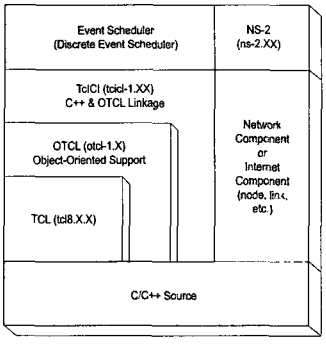
(그림 6) trace 파일의 포맷

고 pkt size는 패킷의 크기를 의미한다. flag 부분에 있는 —— 표시는 어떤 플래그도 사용하고 있지 않음을 의미한다. 여기에서 주의 깊게 보아야 할 곳은 scr_addr과 dst_addr인데 0.0, 3.0과 같이 나타나 있으며, 이는 node.port를 각각 의미한다.

seq num 필드의 내용은 네트워크 계층에서 이용하는 패킷의 순서 번호를 의미한다. 이때 주의할 것은 만약 UDP를 이용한다면 오리지널 프로토콜에서는 순서번호를 이용하지 않지만, NS-2 시뮬레이터에서는 UDP의 추적을 위하여 UDP에도 순서를 붙여서 분석하고 있다는 점이다. 마지막에 있는 pkt id 필드는 패킷의 고유 ID를 나타내는 필드이다.

III. NS-2 시뮬레이터의 구조

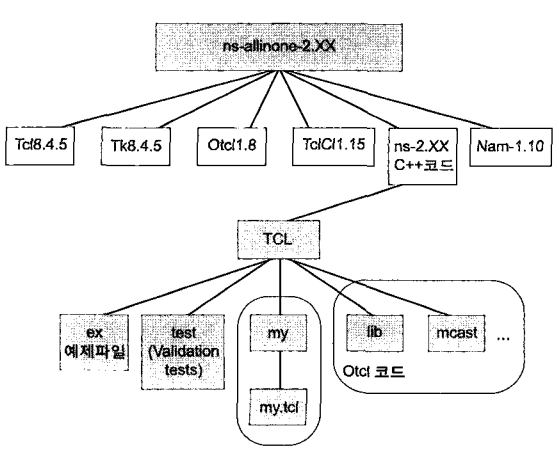
1. NS-2 시뮬레이터의 기본 구조



(그림 7) NS-2 시뮬레이터의 기본 구조

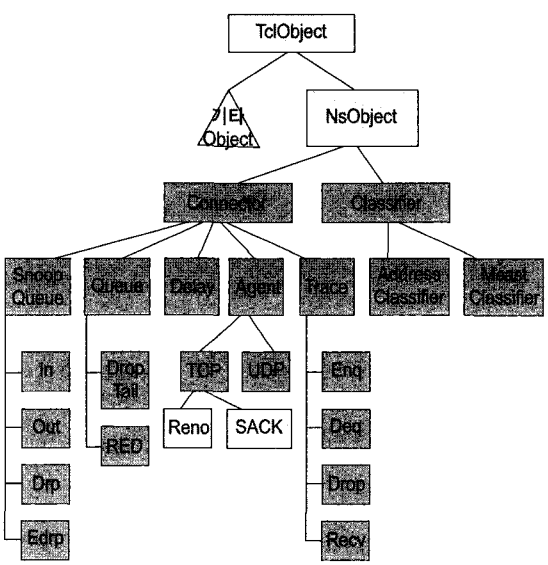
NS-2로 시뮬레이션을 하려면 시뮬레이션 툴의 기본 구조와 환경을 이해해야 한다. 사용자 입장에서 NS-2는 스크립트 언어인 TCL로 시나리오를 작성하여 시뮬레이션을 실행하지만, NS-2 시뮬레이션 환경 자체는 (그림 7)과 같이 여러 가지 프로그램들이 결합하여 하나의 전체적인 시스템을 이루고 있다. 따라서 자신의 프로토콜과 알고리즘을 새로운 모듈로 구현하여 기존의 NS-2 시스템에 통합하려면 이에 대한 이해가 필수적이며, 시뮬레이션 툴에 대한 이해는 스크립트를 올바로 작성하고 원활한 실행을 하는 데에도 큰 도움이 된다. NS-2 에서는 시뮬레이션 환경에 필요한 모든 소프트웨어 패키지를 'allinone' 이라는 이름으로 제공하고 있으며[5], NS-2의 설치는 이 패키지를 이용하여 손쉽게 할 수 있다. 참고로, NS-2의 사용을 위한 플랫폼은 Linux를 비롯한 Unix계열의 OS를 사용하는 것이 가장 무난하며, MS-Windows 기반의 소스도 제공이 되고 있기는 하지만 버그가 많고 실행에 어려움이 있는 것으로 알려져 있다.

NS-2 시뮬레이터에서 사용하는 프로그램 언어는 TCL 스크립트 언어에 객체 지향의 개념을 추가한 OTCL과 C++이다. 여기에서, C++로 기술된 부분들은 주로 소스코드를 정의하고 OTCL은 사용자 인터페이스와 매개 변수 등을 정의 한다. 또한 OTCL과 C++를 연동시키기 위해 Tclcl을 사용 한다. (그림 6)에서 각 프로그램 뒤에 붙은 X표시는 ns-allinone-2.XX 패키지에 포함된 프로그램 파일의 버전을 뜻한다. (그림 8)은 ns-allinone-2.XX 프로그램 패키지에 포함 된 디렉터리 구조이다.



(그림 8) NS-2 시뮬레이터의 디렉터리 구조

NS-2 시뮬레이터 구성을 보면 네트워크 컴포넌트(network component)가 거의 대부분을 차지하고 있다. 네트워크 컴포넌트에는 노드(node), 링크(link), 큐(queue), 에이전트(agent) 등이 포함된다. (그림 9)에 NS-2 시뮬레이터에 있는 OTCL 클래스의 계층 구조를 나타내었으며, 이 클래스 계층 구조에 있는 NsObject 부분에 네트워크 컴포넌트가 포함되어 있다. 즉, 이 클래스는 패킷을 처리하는 기본 네트워크 컴포넌트를 모두 포함하는 슈퍼클래스이다[2]. 이 네트워크 컴포넌트는 다시 두 개의 서브클래스인 연결(connector) 클래스와 분배(classifier) 클래스로 나누어진다.



(그림 9) OTCL 클래스의 계층 구조

2. 스크립트 언어와 C++ 클래스

2.1 TCL (Tool Command Language)

TCL은 버클리 대학에서 개발한 스크립트 언어로, 기능이 유사한 애플리케이션(application)들이 반복되는 문제점을 해결하기 위하여 처음 고안되었다. 그 결과 TCL은 애플리케이션의 확장과 제어를 편리하게 해주는 비교적 간단한 스크립트 언어로 발표되었는데, 이 스크립트 언어는 변수나 제어 구조 등을 제공하는 프로그래밍 툴인 동시에 C 라이브러리 인터페이스를 통한 언어의 확장성 측면에서 매우 편리하다는 장점을 갖는다. TCL의 가장 대표적인 확장 애플리케이션으로는 X-WINDOW의 툴킷(tool kit)인 tk가 있다. tk를 이

용하면 C++를 통한 복잡한 과정을 거치지 않고서도 사용자 인터페이스 환경을 쉽게 구축할 수 있다.

2.2 OTCL (Objected Tool Command Language)

```
Class mom
mom instproc greet {} {
        $self instvar age_
        puts "age_ year old mom say:
        How are you doing?"
        }

# Create a child class of  "mom" called "kid"
Class kid -superclass mom
kid instproc greet {} {
        $self instvar age_
        puts "age_ year old kid say:
        What's up?"
        }

# Create a mom and a kid object, set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function "greet" off each object
$a greet
$b greet
```

(그림 10) OTCL 소스의 예

OTCL 은 MIT에서 구현한 객체지향의 tcl로써 오브젝트 기반 프로그래밍에 지원할 수 있도록 기능을 확장한 스크립트 언어이다. OTCL의 구조적인 특징으로 tcl로 확장 될 수 있도록 설계되었으며, 다른 언어를 포함하려는 시도보다는 tcl의 문법과 개념을 바탕으로 개발되었다. C++에 비하여 간결하고 이식이나 상속이 가능한 구조를 가지고 있다[4]. OTCL 스크립트 언어로 작성한 간단한 소스를 예로 들어 설명을 하겠다(그림 10).소스에서 기존의 TCL에서 볼 수 없었던 클래스를 사용하며, C++와 매우 비슷한 구조를 가지고 있는 것을 알 수 있다.

- Class mom / Class kid -superclass mom

OTCL 스크립트 언어에서 사용하는 클래스는 오브젝트 클래스를 만드는 명령에 의해 만들어진다. 예제 스크립트에서 두 개의 오브젝트 클래스인 mom과 kid를 정의하고 있으며, kid는 mom의 자식 클래스(child class) 역할을 하고 있다. 그리고 -superclass를 보면 자식 클래스인 kid는 슈퍼클래스인 mom으로부터 상속 받는 것을 정의하고 있다.

- mom instproc greet{}

클래스 mom의 멤버 함수는 greet이다. 그리고 instproc는 멤버 함수를 오브젝트 클래스로 정의하기 위하여 사용하는 명령어이다.

- $self instvar age_

멤버 함수를 정의하는 방법을 나타내는 명령 형식으로, OTCL 스크립트 언어에서의 $self는 C++의 this 포인터와 개념이 비슷하다고 이해하면 된다. 그리고 instvar은 클래스나 슈퍼클래스에서 이미 명시된 변수 이름인가를 체크한다. 만약 변수 이름이 이미 선언되었다면 변수는 그 이름을 참고만 할 뿐 새로 만들지는 않는다.

- set a [new mom]

여기에서 new를 이용하여 오브젝트 인스턴스를 생성하였으며 생성한 인스턴스를 set 명령어를 이용하여 변수 a에 대응 시켰다. 위의 OTCL 스크립트 파일을 리눅스의 쉘에서 실행시키면 다음과 같은 결과가 나온다.

```
45 year old mom say:
        How are you doing?
15 year old kid say:
        What's up?
```

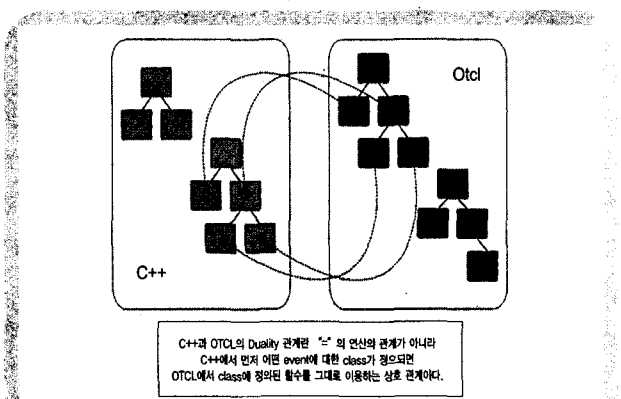
(그림 11) OTCL 소스의 실행 결과

2.3 스크립트 언어와 C++의 상관관계

NS-2 시뮬레이터에서의 OTCL 스크립트 언어와 C++의 상관관계를 알아보자. NS-2에서는 시뮬레이션을 하기 위해 네트워크 노드와 링크를 설정하는데, 이때 각 노드에 특정한 기능을 담당하기 위한 에이전트를 결정한다.

C++로는 네트워크 노드와 각 에이전트의 내부 기능을 기술하며, OTCL 스크립트 언어는 시뮬레이션을 위한 이벤트를 설정한다. 그러나 이것은 절대적인 것은 아니며, 주로 빠른 처리가 필요한 곳에는 C++를 이용하고, 상대적으로 에이전트의 구조를 자주 변경해야할 필요가 있을 때에는 OTCL을 이용하는 것이 일반적이다. (그림 12)에서 C++와 OTCL

스크립트 언어 간의 쌍대성(duality)을 보인다.



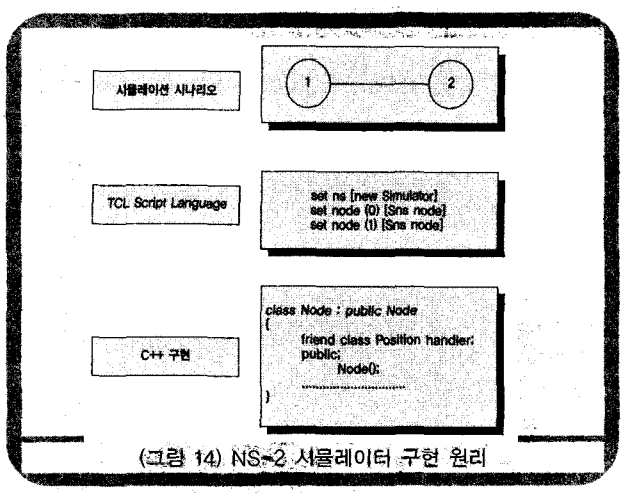
(그림 12) C++와 OTCL간의 쌍대성(duality)

C++ 멤버 함수와 OTCL 오브젝트 함수를 묶어주는 bind() 함수를 예로 들어서 이 둘의 상관관계를 알아본다. 여기서 사용한 인스턴스인 window_를 주의 깊게 살펴보도록 한다. window_ 뒤에 있는 200은 윈도우 사이즈를 의미하며, 결국 윈도우 사이즈를 나타내는 window_는 OTCL에서는 단지 어떤 값으로만 이용 될 뿐이며, 이에 대한 소스 코드는 없다.

```
# bind()의 C++ 멤버 함수

TcpAgent::TcpAgent() {
        bind("window_", &wind_);
        } //bind_time(), bind_bool(), bind_bw()

# bind()에 대한 OTCL 스크립트

$tcp set window_200
```

(그림 13) bind() 함수

이것의 실제 구현은 C++ 클래스에서 정의한다. 그리고 이 때 해당 클래스를 찾아가는 방법은 $tcp에서 지시하는 에이전트를 따라가면 곧 이것이 TCP 에이전트를 구현하고 있다는 것을 알 수 있다. TCP 에이전트의 디폴트 기능을 그대로 이용하지 않고 프로그래머가 이를 변경하고자 할 때 C++의 클래스 부분을 바꾸어야만 하기 때문에, 이는 매우 중요한 방법이다. 실제 NS-2 시뮬레이터에서는 이와 같은 방법으로 새로운 코드를 추가하여 사용자가 원하는 새로운 네트워크에 대한 시뮬레이션을 실행 한다. (그림 13)에서는 NS-2 시뮬레이터에서 노드 2개를 연결하는 경우에 대한 시뮬레이션 시나리오와 스크립트언어, 그리고 C++의 구현원리를 나타내었다.

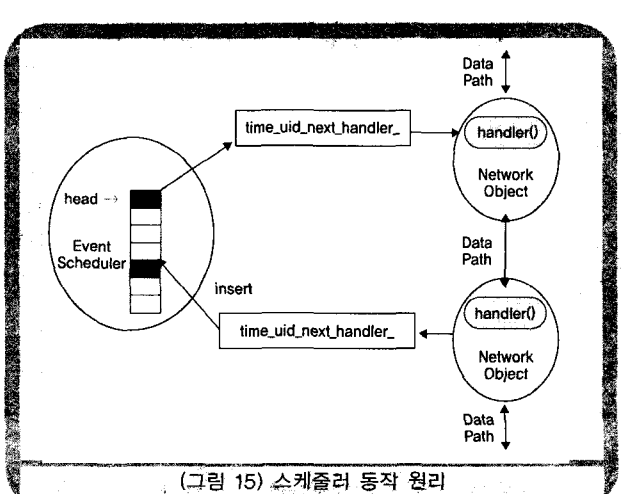


(그림 14) NS-2 시뮬레이터 구현 원리

3. 시뮬레이션 구성요소

3.1 스케줄러(scheduler)

스케줄러는 가장 먼저 도착한 이벤트를 선택하여 동작하며, 단위로 초(second)를 이용한다. NS-2 시뮬레이터는 단일 스레드(single-thread)로 구현되었기 때문에, 미리 설정된 시간 동안에는 오직 한 가지 이벤트만을 처리할 수 있다. 만약 둘 이상의 이벤트를 처리하도록 스케줄링 하였다면, 스케줄러에서는 먼저 스케줄링 된 이벤트를 먼저 처리하도록 한

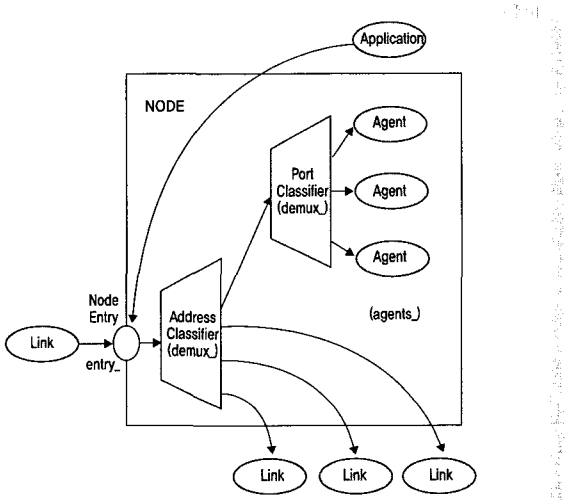


(그림 15) 스케줄러 동작 원리

다. (그림 15)에서 네트워크 오브젝트는 노드나 링크와 같은 네트워크를 구성하는 기본 요소를 말한다. 네트워크 오브젝트들이 함수를 사용하여 송신자에서 수신자 까지 패킷을 전달하게 된다.

3.2 노드(node)

노드는 노드 엔트리 오브젝트(node entry object)와 분류기 오브젝트(classifier object)로 구성된 혼합 형태의 오브젝트로 유니캐스트(unicast) 노드와 멀티캐스트(multicast) 노드가 있다.



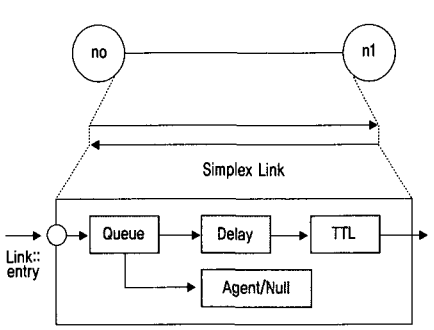
(그림 16) 유니캐스트 노드

유니캐스트 노드는 오직 목적지 네트워크 디바이스 한 곳으로만 데이터를 전달하는 형태를 말한다. 유니캐스트 전달 방식으로 패킷을 보내는 노드를 유니캐스트 노드라 하며, 주소 분류기(address classifier)와 포트 분류기(port classifier)의 두 개의 오브젝트로 구성된다. 주소 분류기는 유니캐스트 라우팅 기능을 수행한다. 이 분류기는 받은 패킷을 올바른 에이전트나 링크로 전달한다. 에이전트는 TCP나 UDP와 같은 전송 계층 프로토콜을 의미하며, 애플리케이션은 응용계층에서 제공되는 FTP, HTTP, 텔넷과 같은 애플리케이션 서비스를 말한다. NS-2에서는 유니캐스트 노드를 디폴트(default) 노드로 사용하고 있다.

멀티캐스트는 패킷 하나를 복사하여 똑같은 패킷들을 다수의 네트워크 디바이스로 전달하는 것이며, 이는 모든 네트워크 장치에게로 패킷을 전달하는 브로드캐스트와는 구별되는 개념이다. 멀티캐스팅 기능을 지원하는 노드를 멀티캐스트 노드라고 한다. 멀티캐스트 노드는 다음과 같이 구성된다. 우선 멀티캐스트 노드는 유니캐스트 노드의 기본 구성 요소인 주소 분류기와 포트 분류기는 그대로 이용하며, 여기에 추가적으로 스위칭 기능이 있는 분류기(switch)와 멀티캐스트 분류기(multicast classifier), 그리고 패킷 복사장치(replicator)를 사용하여 구성된다. 멀티캐스트 노드에 있는 분류기는 전달된 패킷에서 멀티캐스트 패킷만 분류해 내는 스위칭 기능을 제공하며, 멀티캐스트 분류기는 멀티캐스트 라우팅 기능을 제공한다. 그리고 패킷 복사장치는 멀티캐스팅 패킷을 복사한 후, 복사한 패킷을 링크나 에이전트에 전달하는 역할을 수행한다.

3.3 링크(link)

링크는 노드와 함께 네트워크를 구성하는 중요한 네트워크 컴포넌트 오브젝트 중 하나이다. 링크의 종류에는 한 쪽 방향으로만 데이터를 전달할 수 있는 단방향 링크와 양방향으로 데이터를 전달할 수 있는 양방향 링크가 있다[1,2]. NS-2 시뮬레이터에서는 단방향 링크를 단방향 멤버 함수로 구현하고, 디폴트 링크로 사용한다. 단방향 링크는 오직 한 쪽 방향으로만 데이터를 전송 하는 링크를 말한다. 그러므로 양방향 링크를 만들려면 단방향 링크 두 개를 서로 반대 방향으로 향하게 하여 구현해야 한다. (그림 17)에 단방향 링크를 모델링 한 것을 나타낸다.



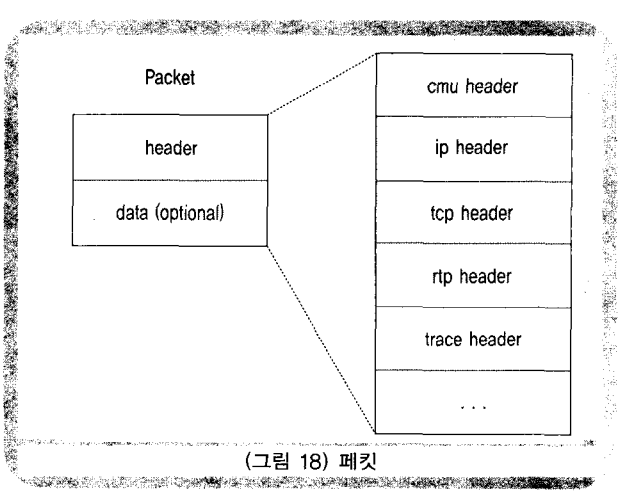
(그림 17) 링크

NS-2 시뮬레이터에서 사용한 링크 모델 개념에서 주의할 점은 링크에 큐가 포함되어 있다는 것이다. 큐잉 이론에서

큐는 노드 구성요소로 들어가지만 NS-2 시뮬레이터에서는 전송 지연과 노드 지연 등을 효과적으로 구할 수 있는 방안으로 노드의 출력 큐를 링크에 추가하여 모델링 하였다. 결과적으로, 큐에서 패킷이 빠져나가면 지연 오브젝트에서 링크 지연 시간을 계산하고, 큐에서 폐기된 패킷들은 null 에이전트로 가서 폐기된다. 마지막에 있는 TTL 오브젝트는 수신한 패킷 헤더의 TTL 파라미터를 계산한 후, TTL 필드를 갱신한다. NS-2 시뮬레이터에서 링크를 설정 할 때에는 NS-2 명령과 함께 옵션을 지정하면 된다. 이 때 정의하는 옵션으로는 링크타입, 대역폭, 지연, 그리고 링크에서 이용하는 큐 타입 등이 있다.

3.4 패킷(packet)

NS-2 패킷은 헤더의 스택과 선택적 데이터 공간으로 구성되어있다(그림 18). 패킷 헤더 포맷은 시뮬레이터 오브젝트가 생성될 때, 등록 된 헤더의 스택에서 초기화 된다. 즉 IP 헤더, TCP 헤더, RTP 헤더 같은 공통 헤더가 초기화 되고, 이것이 모여서 헤더 스택을 구성한다. 이 때, trace 헤더도 미리 설정되어 있다면 함께 초기화 된다. 여기에서, RTP 헤더는 UDP에서 이용하는 헤더를 말한다[1,2].



(그림 18) 패킷

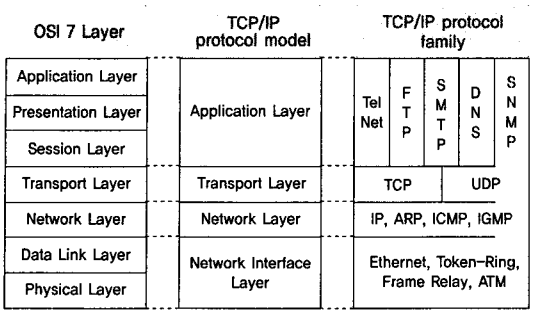
헤더 스택을 구성하는 모든 헤더의 초기치는 모두 오프셋(offset) 값이다 즉, 특정 헤더를 이용하던지 하지 않던지 간에 에이전트에서 패킷을 하나 할당할 때 등록된 전체 헤더 스택은 모두 생성되며, 패킷 스택에 있는 임의의 헤더에 오프셋 값으로 액세스 할 수 있다. 일반적으로 패킷은 헤더 스택과 데이터 공간을 나타내는 포인터로 구성된다. 여기에서, 데이터 공간은 null 값을 갖는다. 실제 네트워크에서 패킷은 데이터 공간에 할당되어 있는 실제 애플리케이션 데이터를 전송한다. 그러나 NS-2 시뮬레이터의 비 실시간 시뮬레이션에서는 의미가 없는 데이터를 전달하고 있기 때문에 이를 지원하기 위하여 구현된 애플리케이션과 에이전트가 차지하는 양은 적다고 볼 수 있어서 null값으로 구현되었다.

3.5 에이전트(agent)

노드를 모두 생성하였다면 네트워크 계층 프로토콜, 즉, 네트워크의 토폴로지는 모두 완성된 것이나, 네트워크 계층의 상위 계층 프로토콜인 전송 계층 프로토콜과 그 위의 응용 계층 프로토콜이 아직 없기 때문에 어떤 트래픽도 발생시키지 않는다. 따라서 새로 생성된 노드는 아직 어떤 기능이나 역할을 수행하지 못한다. 노드를 만들었다면 네트워크 계층의 바로 상위 계층 프로토콜인 전송 계층 프로토콜을 생성하여 노드에 붙여야 한다.

전송 계층 프로토콜을 생성하여 노드에 사용자가 원하는 기능을 수행할 수 있도록 하는 클래스가 에이전트 클래스이다. 원하는 기능이란 노드에서 전송 계층 프로토콜인 TCP나 UDP 패킷을 만들어 전송하거나, ACK 패킷을 전달하는 기능을 말한다. 즉, 에이전트는 NS-2 시뮬레이터에서 전송 계층 프로토콜에 대한 이벤트를 이끌어내는 오브젝트이다. 노드와 노드 간에 이동할 수 있는 TCP-entity나 UDP-entity를 말하는 것이다. 결국 이 entity의 특성을 어떻게 부여하느냐에 따라 에이전트의 동작이 달라진다고 볼 수 있다. 전송계층 프로토콜에서의 TCP나 UDP를 TCP 에이전트나 UDP 에이전트라고 하며, 에이전트는 에이전트 클래스를 이용해서 구현된다. 결국 에이전트는 프로토콜 계층 모델과 함께 이해해야 소스 코딩하는데 어려움이 없다.

(그림 19)의 TCP/IP 프로토콜 패밀리에 있는 네트워크 계층은, 네트워크 토폴로지를 구성하는 노드와 링크를 생성하면서 구현된다. 이렇게 생성된 노드에 상위 계층 프로토콜인 전송 계층 프로토콜과 응용 계층 프로토콜을 attach 시키면 결국 한 노드에서 원하는 트래픽을 생성하여 전달 할 수 있게 된다[2]. 여기에서, 전송 계층 프로토콜인 TCP나 UDP 역할을 수행하는 것이 TCP 에이전트나 UDP 에이전트이다.

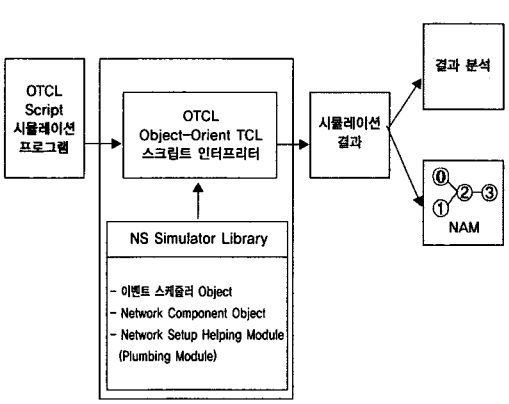


(그림 19) TCP/IP 프로토콜 set

IV. 결 론

네트워크 시뮬레이터인 NS-2는 유닉스 및 리눅스 환경에서 최적으로 구동되고 유선이나 무선 네트워크에 모두 적용되며 무상으로 인터넷에서 다운로드 할 수 있는 시뮬레이터이다. NS-2 시뮬레이터를 구성하는 기본요소에는 이벤트 스케줄러와 네트워크 컴포넌트가 있다. 이러한 기본 요소는 NS-2 시뮬레이터 라이브러리에 구현되어 있다.

현재 국내에서 통신네트워크를 연구하는 학생 및 연구자들이 가장 널리 사용하는 시뮬레이터로써 갈수록 사용자층이 확대되고 있다. 그러나 한편으로는 불편한 GUI와 복잡한 내부구조 등의 문제로 인해서 초보자들은 높은 기술 장벽을 경험하고 이를 해결하기 위해서는 많은 시간과 노력을 투자해야하는 단점이 있다. 이에 따라 소프트웨어 기반기술이 약한 우리나라의 경우에는 새로운 프로토콜 및 시스템에 대한 NS-2 소스코드의 개발이 외국에 비해 저조한 실정이며, 네트워크 신기술 개발을 위해 필수적인 시뮬레이터 기술에 더 많은 투자를 해야 할 필요가 있다.



(그림 20) NS-2 시뮬레이터의 기본구성 요소

참 고 문 헌

[1] Eitan Altman and Tania Jimenez "NS Simulator for beginners" 2003-2004 Univ. de Los Andes, Merida, Venezuela and ESSI, Sophia-Antipolis, France.

[2] Jae Chung and Mark Claypool "NS by Example"

[3] www.isi.edu/nsnam/ns/tutorial

[4] Kevin Fall and Kannan Varadhan "The ns Manual" The VINT Project. A Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC.

[5] www.isi.edu/nsnam/ns

[6] www.openet.com

[7] www.qualnet.com

[8] www.lbl.gov

약 력

박 성 현

2005년 광운대학교 전자공학부 학사
2007년 광운대학교 전파공학과 석사
현재 광운대학교 박사과정
관심분야 : 무선네트워크, 4G MAC

이 승 형

1988년 연세대학교 전자공학과 학사
1990년 연세대학교 전자공학과 석사
1999년 Ph.D, Texas at Austin
1990년 ~ 1995년 국방과학연구소
1999년 ~ 2000년 삼성종합기술원
2000년 ~ 현재 광운대학교 교수
관심분야 : 무선네트워크, 4G MAC