Network Simulator 2

이름 : 전성윤 (roland)

작성일 : 2012-08-15

E-Mail : jsy0251@naver.com

뉴하트 홈페이지 : http://newheart.kr

목차

1. 서론

A. 문서 소개

본 문서에서 소개할 NS2 는 네트워크 프로젝트나 네트워크 관련 논문을 제작할 때 매우 유용하게 사용할 수 있는 시뮬레이터이다. 이 툴의 작동원리를 이해해보면서 네트워크 활동이 어떻게 이루어지는 지, 네트워크 구성요소가 네트워크 상에서 어떻게 작동하고 있는 지를 심층적으로 공부해 볼 수 있다. 또한 네트워크 지식에 기반한 상상을 NS2 를 통해서 적은 비용으로 섬세하게 구현해 볼 수 있다. 본 문서는 NS2 를 통해서 위에 설명한 바를 이루는 데에 필수적인 내용을 정리하고 있다.

B. NS2 를 통한 연구를 하게 된 계기

일반적으로, NS2 는 정보통신학회나 대학/대학원에서 연구용으로 사용되고 있다. 그러나, 학부생 차원의 연구로서 프로토콜을 비롯한 여러 네트워크 분야 아이템에서 창의적인 아이디어가 떠오르면, 학부생도 NS2 를 통해 자신이 원하는 디자인에 대해 신뢰성 있는 증명을 해볼 수 있을 것으로 기대된다. NS2 를 배워가는 단계에서 얻고자 하는 바는 먼저 미시적으로는 네트워크 구성요소의 작동원리를 알고자 하고, 거시적으로는 네트워크 구축을 통한 성능 차이를 느끼고 평가하고자 한다. NS2 를 통해 연구를 하려면 종종 네트워크의 기존의 표준들과 개선되는 표준들을 파악해야 하는데, 이런 점들 또한 앞으로 네트워크 중심의 공부를 하는 데에 도움이 될 것이라는 기대를 한다.

2. 본론

A. NS2 소개

1） NS2의 개요.

A ） 네트워킹을 연구하기 위하여 개발된 이벤트 기반(event-driven)의 시뮬레이터이다.

- Event-driven : 사용자가 미리 정해놓은 이벤트들을 스케줄링하고, 이를 시뮬레이션 했을 때 산출되는 결과는 결국 그 이벤트들에 기반을 두고 있다는 뜻이다.

B ） NS2는 지금도 계속해서 개선되고 있다. 기능이 추가되고 소스 코드가 인터넷상에서 소개 되고 있다.

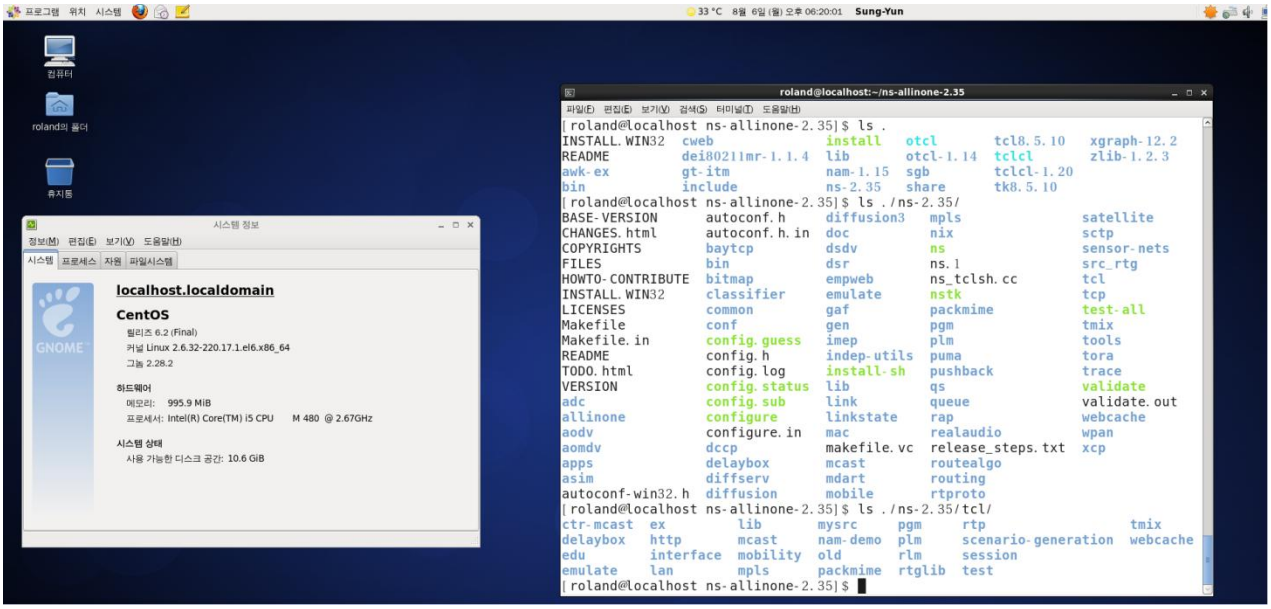
- 1988년에 UC 버클리에서 개발된 'REAL' 네트워크 시뮬레이터를 기초로, 네트워크 연구 그룹인 LBNL 이 네트워크에 적용할 수 있는 프로그램을 연구 및 개발하였다. 이 연구 결과로 발표된 것이 NS1이라는 네트워크 시뮬레이터이다. 프리웨어로 인터넷 상에서 다운받아 쓸 수 있다.
- 1996년에 NS1의 기능을 더욱 더 향상시킨 NS2가 발표되었다. NS1에는 없던, Multi-Path 라우팅, RTP, SRN, 집중형 멀티캐스트, Mobile-IP 프로토콜을 위한 이동 호스트(Mobile Host)지원과 같은 기능을 추가하였다. 이 외에도 NS2에는 SFQ, FQ, DRR 알고리즘 같은 스케줄링 알고리즘도 추가되었다.

C ） 유선 네트워크 뿐만 아니라 무선 네트워크의 프로토콜에 대한 시뮬레이션을 지원한다.

- TCP/IP 프로토콜 패밀리(TCP, UDP, FTP, HTTP 등), 라우팅 프로토콜, 멀티캐스팅 프로토콜, RTP, SRN 등의 인터넷 프로토콜을 시뮬레이션할 수 있다.
- Ad Hoc 네트워크, 이동 통신망의 기지국(Base Station) 모델, WLAN, Mobile-IP 관련 프로토콜, UMTS, 위성 네트워크(Satellite Network) 등과 같은 무선 네트워크 등의 시뮬레이션을 지원할 수 있다.

D ） UNIX 기반 환경에 최적화 되어 있고, 두개의 개발 언어를 사용한다.

- 유닉스 및 리눅스 환경에서 최적으로 구동되고, 개발 언어로는 OTCL과 C++을 사용한다.
- NS1에서는 TCL 언어를 사용한 반면, NS2에서는 MIT에서 개발한 OTCL(Object TCL)을 사용하여 개발한다. OTCL로는 NS2의 사용자 인터페이스(User Interface)와 매개 변수 등을 정의하였다.



[Figure 1] 좌측에서 개발환경의 시스템 정보, 우측에서 NS2의 버전, ns-allinone패키지의 콘텐츠, ns의 콘텐츠, tcl의 콘텐츠를 확인 가능.

2) 개발환경

A) 문서 내의 개발 환경에 대한 시스템 정보는 Figure 1.에서 볼 수 있다.

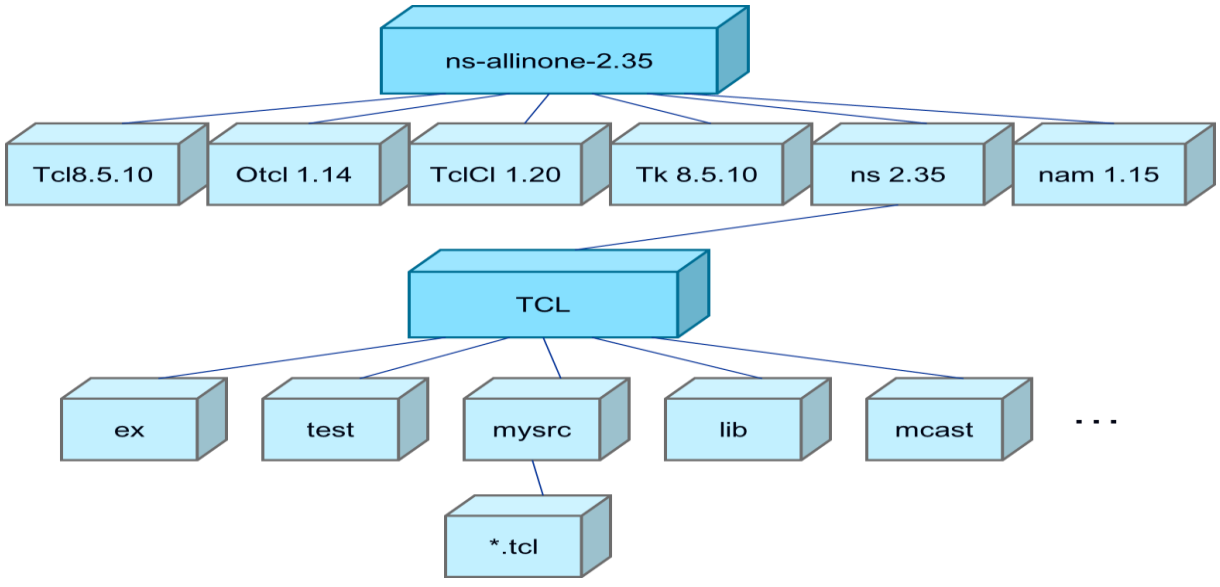
B) ns-allinone 디렉터리는 사용자 홈 디렉터리 아래 두었고, 아래에는 ~로 표시하였다.

C) 설치 시에 xlib패키지와 g++패키지가 없어서 libx11-dev, g++을 apt-get을 이용해 설치하였다.

D) 설치 후에 아래의 환경변수들을 추가하였다.

- LD_LIBRARY_PATH=~/ns-allinone-2.35/otcl-1.14:/hosts/roland/ns-allinone-2.35/lib
- TCL_LIBRARY=~/ns-allinone-2.35/tcl8.5.10/library
- PATH=~/ns-allinone-2.35/bin:~/ns-allinone-2.35/ns-2.35/bin:~/ns-allinone-2.35/tcl8.5.10/unix:~/ns-allinone-2.35/tk8.5.10/unix

3) NS2 시뮬레이터의 기본적인 디렉터리 구조



[Figure 2] [Figure1]에서 기본적인 디렉터리들에 대해 설명하기 위해, 기본적인 디렉터리들만 구조화함.

A) Tcl/Otcl : 개발언어인 Otcl를 사용하기 위해 필요한 내용을 담고 있다. 문법에 대한 것은 뒤에서 다룬다.

B) TclCl : 두 개발언어 C++과 Otcl을 연동시키는 데 필요한 내용을 담고 있다.

C) Tk : Tcl의 확장 애플리케이션으로, C++을 통한 복잡한 개발과정을 쉽게 할 수 있게 해준다.

D) ns : 주로 네트워크 구성요소들의 기능을 C++로 자세히 구현한 내용을 담고 있다.

E) ex/test : Tcl언어 예제코드(ex)와 유효성 검사(test)를 제공해준다.

F) lib/mcast … : 네트워크 구성요소들의 인터페이스를 Otcl로 구현한 내용을 담고 있다.

G) mysrc : 일반적으로 시뮬레이션을 추가나 변경할 때 source위치로 이 곳을 정해둔다.

4) 자주 사용되는 ns-allinone 패키지의 기본 패키지들.

A) nam

- NS2에서 기본적으로 제공하는 Network AniMator라는 애니메이션 툴이다.
- tcl코드 내에서 nam-tracing 기능을 추가하면 시뮬레이션을 영상화 한, *.nam파일이 생성된다.
- 네트워크 토폴로지를 보여주고, 실행버튼을 누르면, 네트워크 시나리오 대로 시뮬레이션을 실행하고, 그 실행 과정을 디스플레이 장치에 직접 보여주는 기능을 한다.
- 그래픽 환경에서만 보여줄 수 있기 때문에, GUI에서만 지원이 되는 기능이다.

B) xgraph

- NS2의 trace-all기능은 패킷 flow를 아주 짧은 시간 단위로 나누어 그때마다의 패킷의 상태를 추적(tracing)하는 기능이다. tcl코드 내에서, trace-all기능을 추가하면 시뮬레이션 후 출력되는 *.tr 파일로 패킷의 흐름에 대한 자세한 정보를 볼 수 있다.
- 하지만, *.tr파일의 패킷의 흐름은 너무 자세하기 때문에 한 눈에 볼 수 없다는 단점이 있다. 이러한 단점을 극복하기 위하여 그 흐름을 거시적으로 볼 수 있도록, xgraph라는 기본 패키지가 제공된다. 이 패키지는 그래프를 그래픽으로 보여준다.
- 그래픽 환경에서만 보여줄 수 있기 때문에, GUI에서만 지원이 되는 기능이다.

C) raw2xg

- xgraph는 *.tr 파일의 형식을 바로 읽어 올 수 없기 때문에, raw2xg라는 툴을 이용해서 *.tr 파일의 형식을 *.xg (임의의 확장자) 파일 형식으로 변환해준다.
  (편의를 위해 확장자를 통해 파일의 종류를 나누기로 약속한다.)
- 변환 시 주의 사항은, raw2xg는 *.tr파일을 xgraph가 읽을 수 있는 형태로 변환하여 콘솔 창으로 출력. 즉, 표준 출력하기 때문에, *.xg파일로 리다이렉션(Redirection)해야 한다는 것이다.

D) Tracegraph

- Tracegraph는 현재 2.05버전까지 나왔으며, 무료로 받을 수 있다,
- ns-allinone패키지에 들어있는 기본적으로 들어있는 툴은 아니지만, 자주 사용되고 있다.
- 2D나 3D 차트가 지원되고, 히스토그램(histogram)을 그릴 수 있다.
- 다양한 형태의 통계 자료를 수치로 표시할 수 있다.
- 1000 라인이 넘은 trace파일은 처리하지 못한다는 단점이 있다.

5) 개발언어 간(OTCL 과 C++)의 관계

A) TCL

(1) 소개

- TCL(Tool Command Language)은 버클리 대학에서 개발한 스크립트 언어이다.
- 기능이 유사한 애플리케이션들이 반복되는 문제점을 해결하기 위하여 처음 고안되었다.
- 변수나 제어 구조 등과 같은 프로그래밍 툴(Programming Tool)인 동시에 C 라이브러리 인터페이스(C Library Interface)를 통한 언어의 확장성 측면에서 매우 편리하다는 장점을 가진다.

(2) 기본 문법

- proc proc_name {argument} { ... }
  프로시저를 정의한다. proc_name 는 프로시저 이름에 해당하고, argument 는 프로시저에 넘겨줄 인자에 해당한다. { ... }안에 프로시저를 정의한다.
- set inst 60
  변수를 선언한다. inst 는 변수명이고 60 은 변수값이다.
- [expr _expression]
  _expression 부분에는 표현식이 들어간다. [expr ...]은 이 표현식을 계산하는 인터프리터를 만드는 기능을 수행한다.
- $instance
  변수 값을 나타낸다.$는 반드시 변수이름과 함께 사용해야 한다.
- puts " hello TCL, [expr pow($d,$k)]"
  " "안에 있는 문자열을 출력하는 명령이다. [expr ...], $inst, 즉, 변수도 들어갈 수 있다.

B) OTCL

(1) 소개

- OTCL(Object TCL)은 MIT 에서 구현한 객체지향의 TCL 로 Tcl/Tk 를 객체지향 프로그래밍까지 지원할 수 있도록 기능을 확장한 스크립트 언어이다.
- 구조적인 특징은 TCL 까지 철저하게 동적으로 확장될 수 있도록 설계되었으며, C++에 비하여 간결하고 이식이나 상속이 가능한 구조를 가지고 있다.

(2) 추가된 기본 문법

- Class Class_name / Class Kid_Class_name -superclass Parent_Class_name
  클래스 이름에 해당하는 클래스가 선언되며, 아래에서 멤버변수와 멤버함수를 추가하는 방법을 제시하고 있다.

- instproc init/instproc destroy

  생성자와 소멸자에 해당하며, 이후에 나올 문서 중에서 다루기로 한다.
- Class_name instproc proc_name { argument } { ... }

  멤버함수를 정의하며, proc 를 통한 정의와 형식이 비슷하다.
- $self instvar inst

  멤버변수를 정의하며, inst 에 변수명이 들어간다. 만약, 기존에 같은 이름(inst)의 멤버변수가 있다면, 그 이름을 참고만 할 뿐 변수를 새로 만들지는 않는다.
- Set a [new Parent]

  new 를 이용하여 객체(Object Instance)를 생성하였으며, 이를 set 명령으로 변수 a 에 대응시켰다. 객체를 생성하고 변수명에 대응하는 방법은 여러 개이나 이후에 나올 문서 중에서 다루기로 한다.
- ns ex_otcl.tcl

  OTCL 스크립트 파일인 ex_otcl.tcl 을 리눅스의 쉘 프롬프트(Shell Prompt)에서 실행시키는 명령으로 ns 파일명.tcl 의 명령 형식을 따른다.

C) 개발언어 (C++과 OTCL)간의 상관 관계 : 쌍대성

- NS2 시뮬레이터에서는 시뮬레이션을 하기 위해 너트워크 노드(Node)와 링크(Link)를 설정하는데, 이때 각 노드에 특정한 기능을 담당하기 위한 에이전트(Agent)를 결정한다.
- 주로, C++로는 네트워크 노드와 각 에이전트의 내부 기능을 기술하며, OTCL 스크립트 언어는 시뮬레이션을 위한 이벤트(Event)를 설정한다.
- C++와 OTCL 의 Duality 관계란, "="의 연산관계가 아니라, C++에서 먼저 어떤 이벤트에 대한 클래스가 정의되면, OTCL 에서는 클래스에 정의된 함수를 그대로 이용하는 상호 관계이다.
- 위의 내용은 절대적인 것은 아니며, 빠른 처리가 필요한 곳에는 C++를 이용하고, 상대적으로 에이전트의 구조를 자주 변경해야 할 필요가 있을 때에는 OTCL 을 이용하는 것이 일반적이다.

  < 예제>

  1. Bind()의 C++ 멤버 함수

     TcpAgent::TcpAgent(){

     bind("window_",&wind_);

     }
  2. Bind()에 대한 OTCL 스크립트

     $tcp set window_ 200
- 에이전트들의 디폴트 기능을 사용하지 않고, 사용자가 지정한 값을 사용하려면 C++코드를 수정해야 할 때가 있기 때문에, 수정해야 할 C++클래스와 OTCL 클래스가 어떻게 연동되어있는지에 대해서도 파악하고 있어야 한다.

6) NS2 를 통한 네트워크 시뮬레이션 작성 플로우 차트

A) 네트워크 토폴로지/시나리오 작성 단계

(1) 네트워크 토폴로지란 네트워크 구조, 통신 방법, 특성에 대한 정의를 뜻한다. 네트워크 시나리오는 네트워크 지식을 기반으로 구현하고자 하는 네트워크를 설계하는 단계이다

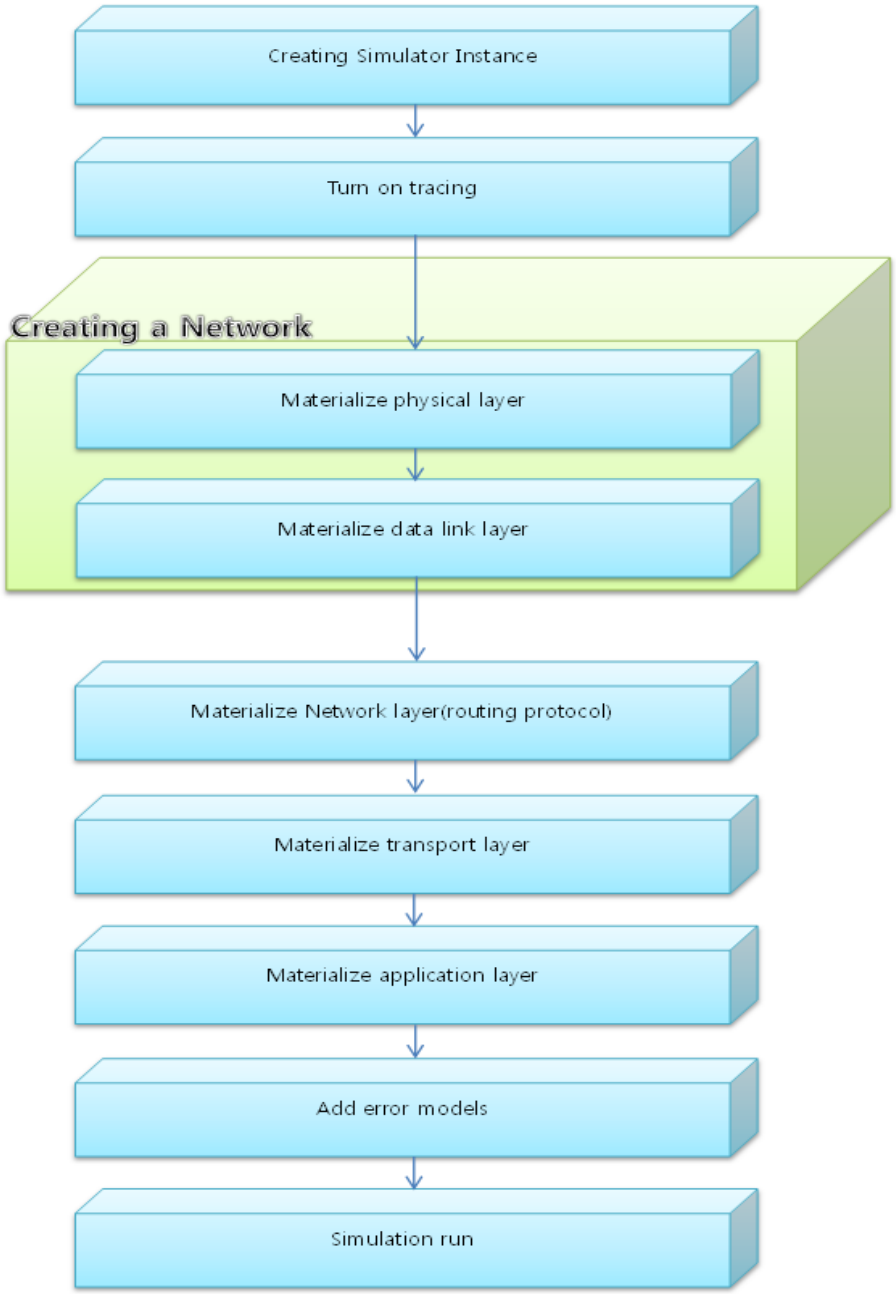
```
#                    RM sender
# topology:      2 o           o 5  RM recv1
#                    \ 0     1 /
#     TCP send1 3 o--o-------o--o 6  RM rec 2
#                  /          |\
#     TCP send2 4  o          | o 7  TCP recv 1
#                             o 8  TCP recv 2


# TCP sender 1 starts to send at t=0sec
# SRM sender starts to send at t=10s
# tcp sender 2 starts to send at t=20s
# BW of link 0-1 = 1 Mbps and delay is 20ms . For rest of the
# topology BW = 10Mbps and delay of 5ms.
# flowmonitors compare thruput of TCP and rm flows in links 1-5, 1-6,
# 1-7 and 1-8.
```

Figure 3. 위 그림은 네트워크 토폴로지의 예이다.

B) TCL 스크립트 소스코드 작성 단계

(1) 라이브러리를 이용해서 시나리오를 구현한다.



[Figure 4] TCL 스크립트 소스코드 작성 단계를 도표로 만들었다.

C) C++ 또는 OTCL 스크립트를 통해 구현하는 단계 : 확장

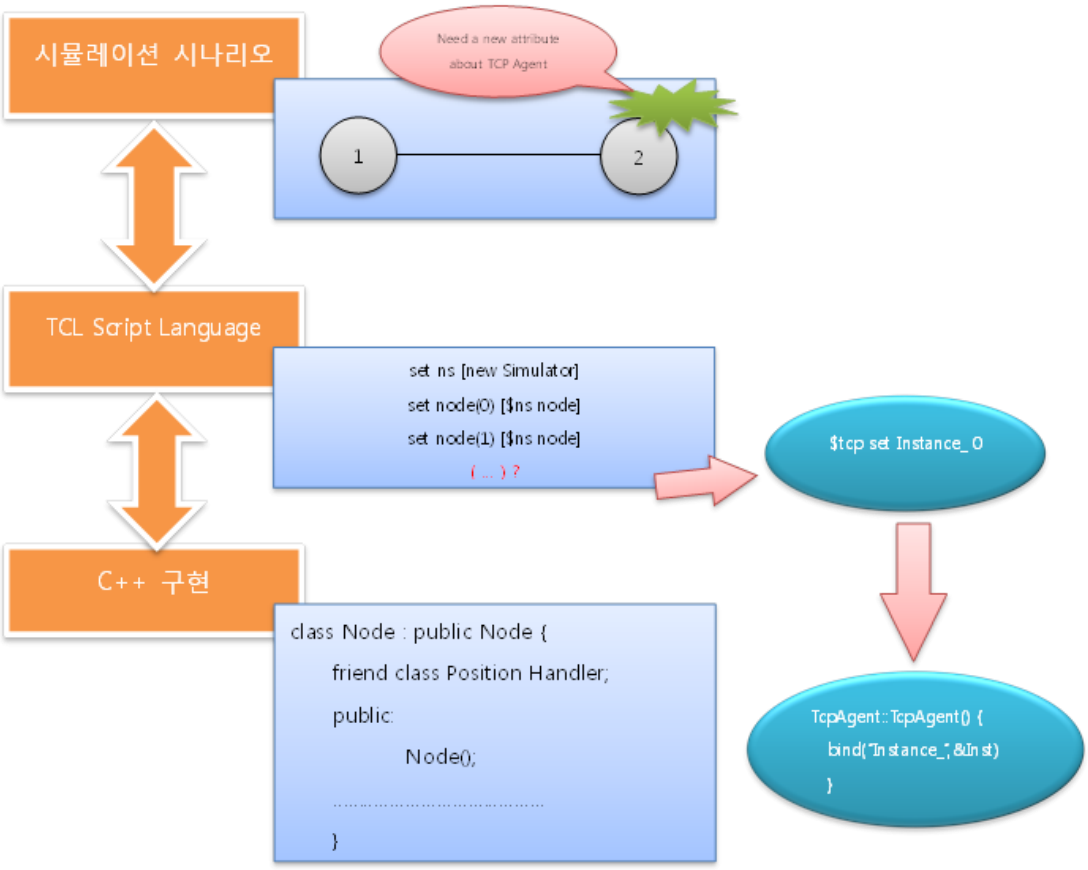
(1) 스크립트 소스코딩 중 필요한 요소가 있으면 추가한다. 네트워크 요소가 추가되는 원리를 (Figure5.)로 정리하였다. 본 문서는 Base 버전 이므로 실제로 NS2 를 확장하는 자세한 예제들은 나중에 나올 문서에서 다루기로 하고, 개념적인 부분만 살펴보기로 한다.

(2) 또, tcl 파일을 실행시켰을 때 에러가 뜨는 경우, 소스 코드에서 잘못된 부분이 있으면 해당 요소가 있는 클래스들을 찾아서 수정한다.

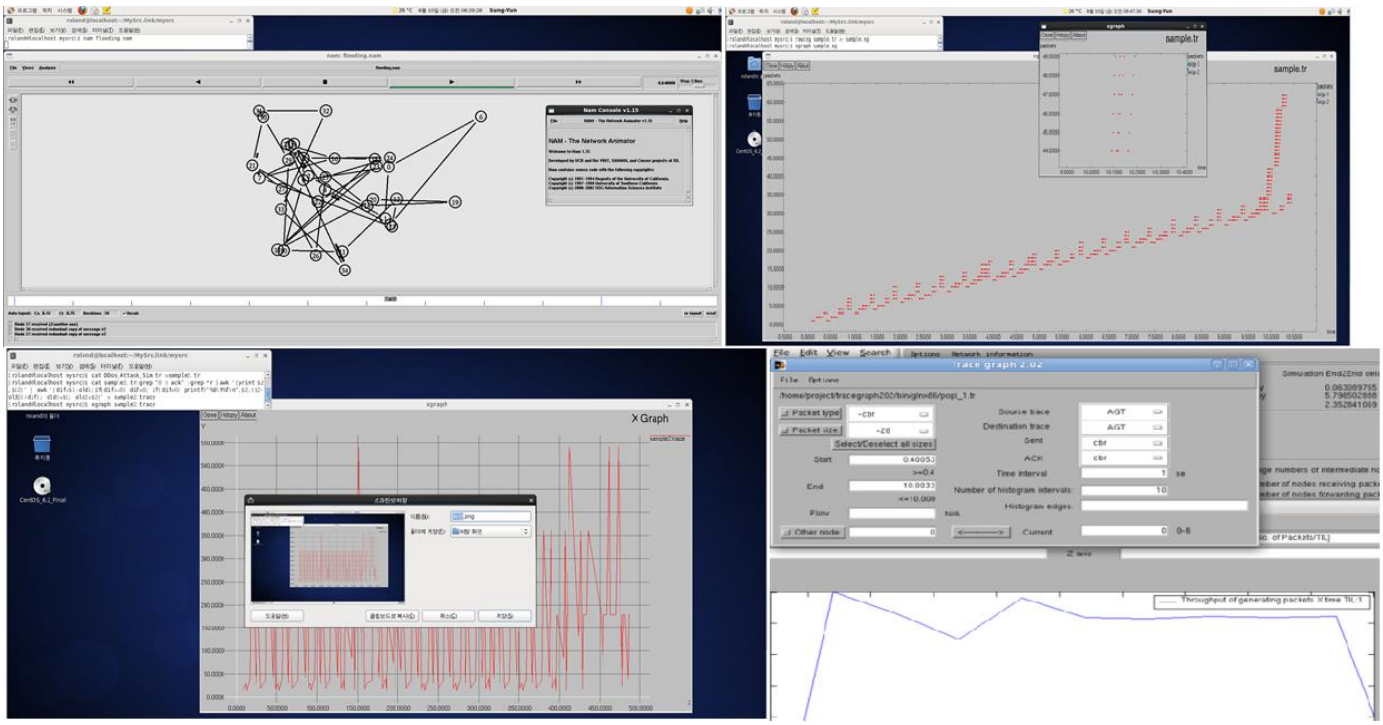
D) 결과 분석 단계

(1) Animator, 그래프 툴(2.A.iv참고), awk 명령 등을 이용해서 결과를 거시적으로 분석하고, 필요한 부분은 적절한 명령을 통해서 미시적인 부분을 추출해서 분석한다.

(2) Figure6.에서 결과 분석에 쓰이는 툴과 명령을 확인할 수 있다.



[Figure 5] 네트워크 요소가 추가되는 원리



[Figure 6] 결과 분석에 쓰이는 툴과 명령

(왼쪽 상단 : NAM, 오른쪽 상단 : Xgraph, 왼쪽 하단 : awk명령어로 Xgraph를 응용, 오른쪽 하단 : tracegraph)

C. 시뮬레이터의 구성요소

1) 이벤트 스케줄러

A) 이벤트 스케줄러가 필요한 이유 : NS2 는 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 처리(FSFD, First Scheduled First Dispatched)하도록 한다.

(1) 스케줄러 종류

(A) 비실시간 스케줄러

- List 스케줄러 : 이벤트는 linked-list 구조로 시간 순서대로 스케줄링 된다. 동시에 발생되는 이벤트에 대하여 FIFO 를 적용하여 이벤트를 실행시킨다.
- Heap 스케줄러 : List 구조에 비해 heap 구조가 삽입과 삭제에 걸리는 시간이 짧기 때문에, 더 많은 이벤트를 처리할 수 있다.
- Calendar 큐 스케줄러 : David Wetherall 이 구현하였다. 데이터 구조의 연속된 변화를 어떤 해의 달력을 기준으로 기록하는 방식으로 구현하였다. 즉, 수년 동안의 같은 월/일에 발생하는 이벤트들을 어느 날로 기록하였다.

(B) 실시간 스케줄러

- RealTimeScheduler 라는 이름으로 스케줄러가 있고, 시뮬레이터가 실제 네트워크에서 상호 작용하는 에뮬레이션 방법을 이용하여 구현되었다.

(2) 비실시간 스케줄러는 논리적으로 모두 동일한 동작을 수행한다고 볼 수 있다. 하지만 이렇게 구분하는 이유는 스케줄러의 역호환성을 지원하기 위해서이다.

- 역호환성 : 새로 나온 버전의 프로그램이나 시스템이 이전에 나온 버전도 지원하는 것을 말함

B) 스케줄러들에 관한 내용은 ~/ns-allinone-2.35/ns-2.35/common/scheduler.cc 에 C++코드가 있고, ~/ns-allinone-2.35/ns-2.35/tcl/lib/ns-lib.tcl 에 OTCL 스크립트가 있다.

2) 노드와 링크

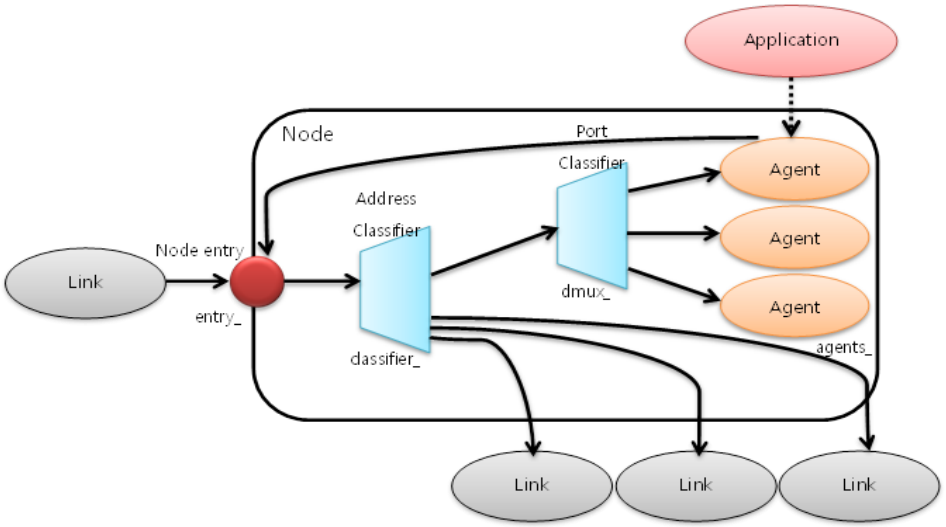
A) 노드

(1) 호스트 PC, 허브 또는 네트워크 중계용 장비 (라우터, 중계 위성, 중계 차 등)에 해당한다. 주로 네트워크 토폴로지에서 원으로 형상화된다.

(2) 단말노드에서 트래픽을 발생시키는 특성을 정의해주면, 그 노드는 호스트 PC 가 되고, 그 다음 노드들은 링크에 따라, VPN 일 경우에는 서버가 될 수도 있고, 또한 허브도 될 수 있을 것이다.

(3) 노드는 유니캐스트 노드와 멀티캐스트 노드로 나뉘어 진다.

(A) 유니캐스트 노드



[Figure 7] 유니캐스트 노드.

- 어드레스 분류기는 유니캐스트 라우팅 기능을 수행한다. 즉, 받은 패킷을 포트 classfier 로 보내어 agent 에게 보내거나, 연결된 링크에게 보내는 기능을 한다.
- 포트 분류기는 받은 패킷을 해당 포트에 연결된 agent 로 보내 준다.
- 여기서 Agent 는 전송 계층 프로토콜을 의미하며 TCP, UDP 등이 해당되고, Application 은 응용계층 프로토콜을 의미하며, FTP, HTTP, Telnet 과 같은 애플리케이션 서비스를 말한다.

(B) 멀티캐스트 노드

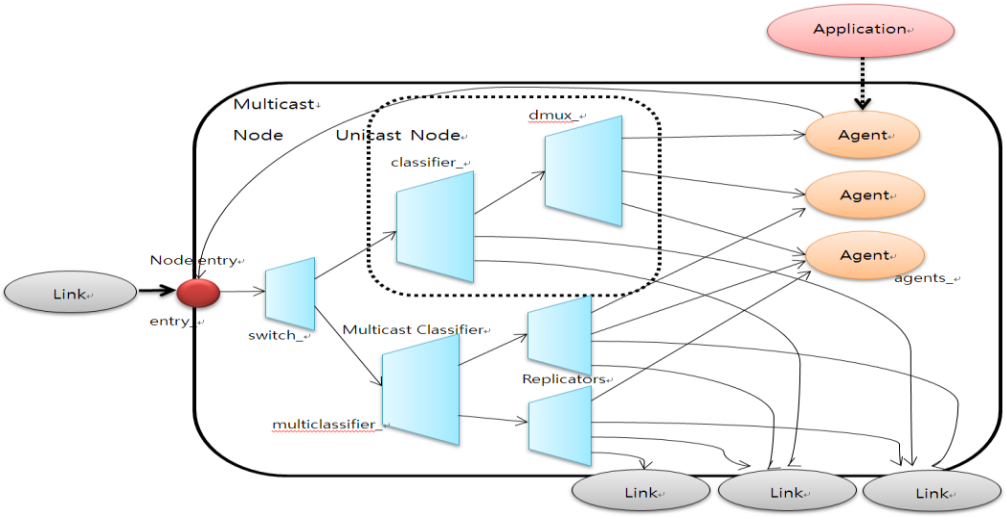


Figure 8. 멀티캐스트 노드

- 멀티캐스트는 패킷 하나를 복사한 똑 같은 패킷들을 다수(multiple)의 네트워크 디바이스로 전달하는 것이다. 이는 모든 네트워크 장치로 패킷을 전달하는 Broadcast 와는 구별되는 개념이다. 멀티 캐스트를 지원하는 노드를 멀티캐스트 노드(multicast node)라고 한다.
- 멀티 캐스트 노드는 유니캐스트 노드(Unicast Node)에 스위칭(switch_)기능이 있는 분류기, 멀티캐스트 (multicastclassifier_)분류기, 그리고 패킷 복사장치(Replicator)를 더 추가하여 구성된다.
- 스위칭 기능이 있는 분류기는 멀티캐스트 패킷만 분류하는 스위칭 기능을 제공한다.
- 멀티캐스트 분류기는 멀티캐스트 라우팅 기능을 제공한다.
- 패킷 복사장치는 멀티캐스팅 패킷을 요청된 수만큼 복사한 후, 복사한 패킷을 링크나 에이전트에 전달하는 역할을 수행한다.

(4) 노드에 관한 내용은 ~/ns-allinone-2.35/ns-2.35/common/node.cc 에 C++코드가 있고, ~/ns-allinone-2.35/ns-2.35/tcl/lib/ns-node.tcl 에 OTCL 스크립트가 있다.

B) 링크

(1) 노드의 연결이다. 연결의 특성을 정해줌으로써 노드 간의 통신을 정의할 수 있다. 즉, 물리적 계층에서의 배선이나 선의 종류. 무선의 통신의 경우, 주파수 등의 통신 매체에 대한 특성들을 정의하는 것이다.

(2) 링크의 종류

(A) 단방향 링크(simplex-Link)

(B) 양방향 링크(duplex-Link)

(3) 링크 시뮬레이션(Link Simulation)

(A) NS2 시뮬레이터에서 링크를 이용할 때에는 단방향 링크나 양방향 링크의 기본 모델을 그대로 사용하지 않고, 새로운 오브젝트를 추가하여 시뮬레이션에 활용한다. 즉, 이벤트 추적(trace)을 위한 오브젝트를 추가한다거나 큐 모니터(Queue Monitor)를 추가하는 것과 같은 세부적인 이벤트를 기록하여 네트워크에서 패킷에 대한 정확한 시뮬레이션을 수행한다.

(B) Trace 오브젝트가 추가된 링크 모델

- 네트워크에서 실제 패킷을 전달하고, 패킷의 지연 시간을 측정하는 것과 같은 실제 네트워크에서 발생되는 모든 이벤트를 네트워크 활동이라고 한다. NS2 에서 네트워크 활동을 추적하는 방법은 trace-all 기능과 nam 을 위한 namtrace-all 기능을 추가하는 것이다.

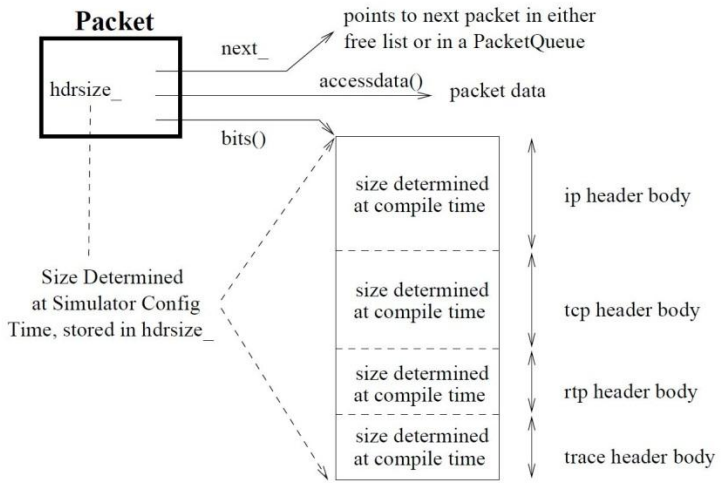
(C) Queue 모니터링

- Snoop 큐 오브젝트를 추가시켜서 큐 모니터링을 할 수 있다. 큐에서 패킷이 관리되고 출입되는 활동들에 대한 데이터를 얻을 수 있다.

(4) 링크에 관한 내용은 ~/ns-allinone-2.35/ns-2.35/link 에 C++코드가 있고, ~/ns-allinone-2.35/ns-2.35/tcl/lib/ns-link.tcl 에 OTCL 스크립트가 있다.

3) 패킷 헤더와 패킷 스케줄링

A) 실제 패킷 헤더와 ns2 의 패킷헤더 비교



[Figure 9] NS2에서 구현하는 패킷의 헤더.

(1) 실제 패킷에서, 패킷 헤더는 OSI 계층별로 추가되는 헤더들의 스택이 형성되어 Physical 계층에서 완전한 패킷 헤더가 되어 통신한다.

(2) NS2에서, 패킷 헤더는 Figure 9.에서 보듯이, 구조체로 먼저 정의되어있다.

(3) 패킷에 관한 내용은 ~/ns-allinone-2.35/ns-2.35/common/packet.cc 에 C++코드가 있고, ~/ns-allinone-2.35/ns-2.35/tcl/lib/ns-packet.tcl에 OTCL스크립트가 있다.

B) 패킷 스케줄링

(1) 이벤트 스케줄러는 전체 이벤트에 대한 스케줄링을 담당한다. 반면, 패킷 스케줄링은 레디큐에 있는 패킷들이 서비스되거나 폐기될 지를 결정해준다.

(2) 레디큐에서 패킷을 보관하는 중에는 버퍼 관리 기법에 의해서, 패킷 스케줄링에 알맞은 큐가 조성된다.

(3) NS2 시뮬레이터에 현재 구현되어 있는 버퍼 관리 기법들은 다음과 같다.

(A) DropTail(FIFO) : First In First Out 구조. 즉, 먼저 큐에 도착한 패킷을 먼저 서비스하는 기법이다. 구현이 간단하다. 하지만,

(B) RED : Random early detected. 가장 먼저 발견되는 패킷을 먼저 서비스하는 기법이다. DropTail 보다 버퍼관리를 완료하는 데 걸리는 시간이 적어진다.

(C) CBQ : Class-Base Queueing. 우선순위와 라운드-로빈 스케줄링에 쓰이는 기법이다.

(D) FQ : FQ(Fair Queueing), SFQ(Shortly Fair Queueing), DRR(Deficit Round-Robin)의 기법이 존재한다.

(4) 폐기된 패킷은 드롭 큐에 저장된다. 드롭 큐는 드롭된 패킷을 저장하며, 드롭 큐를 통해서 드롭된 패킷에 대한 통계를 낼 수 있다.

(5) 패킷 스케줄링 알고리즘들에 관한 내용은 ~/ns-allinone-2.35/ns-2.35/queue/ 에 C++코드가 있고, ~/ns-allinone-2.35/ns-2.35/tcl/lib/ns-packet.tcl 에 OTCL 스크립트가 있다

4) 라우팅 프로토콜

A) 유니캐스트

(1) 네트워크에서 유니캐스트가 의미하는 것은 출발지 노드와 목적지 노드간에 일대일 통신을 하는 것을 의미한다.

(2) 네트워크 시뮬레이터에서 유니캐스트는 유니캐스트 노드를 의미하는 데, 유니캐스트 라우팅 프로토콜을 입힌 노드가 유니캐스트 노드가 된다.

(3) NS2 의 노드는 디폴트로 유니캐스트 노드이다.

(4) 유니캐스트에 관한 내용은 ~/ns-allinone-2.35/ns-2.35/node.cc 에 C++코드가 있고, ~/ns-allinone-2.35/ns-2.35/tcl/lib/ns-node.tcl 에 OTCL 스크립트가 있다

B) 멀티캐스트

(1) 네트워크에서 멀티캐스트가 의미하는 것은 한 출발지 노드에서 여러 개의 목적지 노드를 향하여 메시지를 주는 것이다. 멀티캐스팅은 멀티캐스트를 지원하는 노드에서만 가능하다.

(2) 네트워크 시뮬레이터에서 멀티캐스트는 앞에서 노드를 설명한 것과 같이 패킷 복사기를 통해서 멀티캐스팅이 구현된다.

(3) 멀티캐스트에 관한 내용은 ~/ns-allinone-2.35/ns-2.35/mcast/ 에 C++코드가 있고, ~/ns-allinone-2.35/ns-2.35/tcl/mcast/ 에 OTCL 스크립트가 있다

5) 에이전트

A) NS2 에서의 에이전트란?

(1) 네트워크에서 TCP, UDP 와 같은, OSI 7 Layer 중 Layer 4 를 구현한 것이다.

(2) 노드 내에 있는 TCP 들은 각각 포트(port)를 통해서 application layer 와 network layer 를 연결해준다.

(3) TCP, UDP 와 같은 전송계층 프로토콜이 에이전트에 해당한다.

(4) NS2 에서는 노드에 에이전트를 attach 하면 에이전트 간에 통신이 가능하다.

(5) NS2 에서는 TCP 와 UDP 가 구현되어있으며, 각 에이전트는 아래와 같이 설명된다.

B) TCP

(1) TCP 는 연결 지향형 전송 계층 프로토콜로 신뢰성 있는 연결을 하기 위해서 연결 대상이 되는 시스템에 데이터를 전송하기 전에 반드시 연결을 설정하고, 데이터 전송이 모두 완료된 후에는 연결을 해제한다.

(2) 이 때, TCP 는 3 번에 걸쳐 TCP 세그먼트을 주고 받는 3-way handshake 메커니즘을 사용한다.

(3) 3-way handshake

(A) 연결 설정 메커니즘

- SYN Segment
  - 송신 호스트에서 연결 요구를 위한 세그먼트(SYN Segment)를 수신 호스트에게 보낸다. SYN Segment 의 플래그 설정은 SYN 은 set 되어있고 ACK 은 Not set 되어있다.

```
Flags: 0x02 (SYN)
    0... .... = Congestion Window Reduced (CWR): Not set
    .0.. .... = ECN-Echo: Not set
    ..0. .... = Urgent: Not set
    ...0 .... = Acknowledgment: Not set
    .... 0... = Push: Not set
    .... .0.. = Reset: Not set
    .... ..1. = Syn: Set
    .... ...0 = Fin: Not set
```

[Figure 10] WireShark를 통해 본 SYN Segment 의 Flags 부분

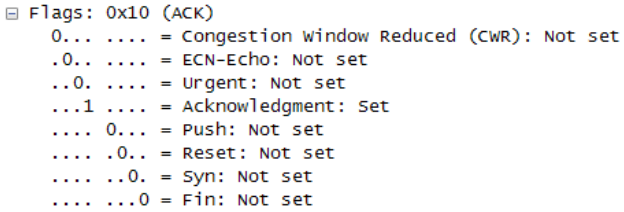
  - 이 세그먼트는 애플리케이션 계층의 데이터는 포함하지 않는다.
  - 실제로 SEQ 값은 0 으로 시작한다. 하지만, NS2 는 호스트 구분을 위해 0,1 이 아닌 다른 랜덤 넘버로 시작하는 것을 채택하고 있다.
- SYN ACK Segment
  - 수신 호스트에서 SYN Segment 를 받으면, 송신 측에서 연결요청을 하고 있음을 감지하고, 연결을 허락하는 SYN ACK Segment 를 송신 호스트에게 보낸다. SYN ACK Segment 의 플래그 설정은 SYN 과 ACK 둘다 set 되어있다.

```
Flags: 0x12 (SYN, ACK)
    0... .... = Congestion Window Reduced (CWR): Not set
    .0.. .... = ECN-Echo: Not set
    ..0. .... = Urgent: Not set
    ...1 .... = Acknowledgment: Set
    .... 0... = Push: Not set
    .... .0.. = Reset: Not set
    .... ..1. = Syn: Set
    .... ...0 = Fin: Not set
```

[Figure 11] WireShark를 통해 본 SYN_ACK Segment의 Flags 부분

  - 이 세그먼트는 애플리케이션 계층의 데이터는 포함하지 않는다.
  - SYN Segment 에서 받은 SEQ 값은 그대로 전달한다.

- ACK
  - 송신 호스트의 관점에서 보았을 때, 송신 호스트는 수신 호스트로 연결 요청을 한 후 응답을 기다리고 있다가 수신 호스트로부터의 SYN ACK Segment 를 받으면 최종 연결 확인 응답인 ACK Segment 를 보내고 송신 호스트는 연결 설정 작업을 모두 끝낸다.



[Figure 12] WireShark를 통해 본 ACK Segment의 Flags 부분

  - 이 세그먼트는 애플리케이션 계층의 데이터는 포함하지 않는다.
  - 수신 호스트가 보낸 SEQ 를 하나 증가하여 수신호스트에게 연결 작업이 완료되었음을 알리는 신호를 보낸다.

(B) 연결 해제 메커니즘

- 연결 설정 메커니즘에서는 SYN 플래그를 설정하는 것과는 반면 연결 해제 메커니즘은 연결 설정 메커니즘과 같은 순서로 FIN 플래그를 이용하여 한다.
- 현재 서비스나 프로세스가 다른 TCP 연결이 필요로 하거나 다른 서비스나 프로세스가 현재 서비스가 점유하고 있는 TCP 연결을 필요로 하는 상황이 일어나기 때문에 연결 해제가 필요하다.
- 연결 해제를 하면 위 상황에서 서비스나 프로세스는 네트워크 자원(eg. Port)를 다시 할당 받을 수 있다.

(4) NS2 에서 TCP 전송을 시뮬레이션 하기 위해서는 Agent/TCP 와 Agent/TCPSink 를 한 쌍으로 설정해야 한다. 송신 TCP Agent 의 역할은 Agent/TCP 클래스가 하고, 수신 TCP Agent 의 역할은 Agent/TCPSink 가 한다.

(5) NS2 에서 TCP Agent 는 One-Way Agent 와 Two-Way Agent 로 분류할 수 있다. 현재, Two-Way Agent 는 개발 중에 있다.

(A) One-Way Agent 의 경우는 TCP 수신 에이전트는 TCP Sink 로만 동작한다. 따라서 호스트 간에 일방적인 TCP 연결이 성립된다. 반면 Two-Way Agent 는 TCP 수신 에이전트와 송신 에이전트가 대칭적으로 존재해서 상호적인 TCP 연결을 가능하게 할 수 있다.

C) UDP

(1) 비연결형 전송 계층프로토콜로 TCP 와 달리 연결 설정, 유지와 관련된 절차가 없는 신뢰성이 낮은 전송 서비스를 제공한다.

(2) 이 프로토콜은 데이터의 손실이 발생하여도 많은 영향을 주지 않는 통신서비스에 주로 쓰인다.

(3) 최근에는 전송 효율이 높은 비연결형 서비스에 쓰이고 있다. 예를 들면 VoIP(Voice over IP), 인터넷 전화와 같은 음성을 위한 통신이 필요한 곳에서 쓰인다.

(4) UDP 에서는 별도의 오류 제어를 하지 않고 Checksum 으로만 오류를 검사한다. 따라서, TCP 와 오류를 조치하는 방법도 달라진다. 오류가 발생해도 무시하는 경우도 있지만, 예를 들면 수신측에서 오류가 발생하였다는 것을 ICMP 를 사용하여 송신측 호스트에게 알려주는 방법도 있다.

(5) NS2 에서 UDP 전송을 시뮬레이션 하기 위해서는 Agent/UDP 와 Agent/NULL 을 한 쌍으로 설정해야 한다. 송신 UDP Agent 의 역할은 Agent/UDP 클래스가 하고, 수신 UDP Agent 의 역할은 Agent/NULL 가 한다.

D) 에이전트에 관한 내용은 ~/ns-allinone-2.35/ns-2.35/common/agent.cc 에 C++코드가 있고, ~/ns-allinone-2.35/ns-2.35/tcl/lib/ns-agent.tcl 에 OTCL 스크립트가 있다

6) 애플리케이션 서비스

A) 애플리케이션 서비스란 FTP, Telnet, HTTP, DNS, SNMP 와 같은 Application Layer 의 프로토콜들을 말한다.

B) NS2 에서 애플리케이션 서비스는 트래픽 생성기(Traffic Generator)와 시뮬레이션 할 애플리케이션으로 구현된다.

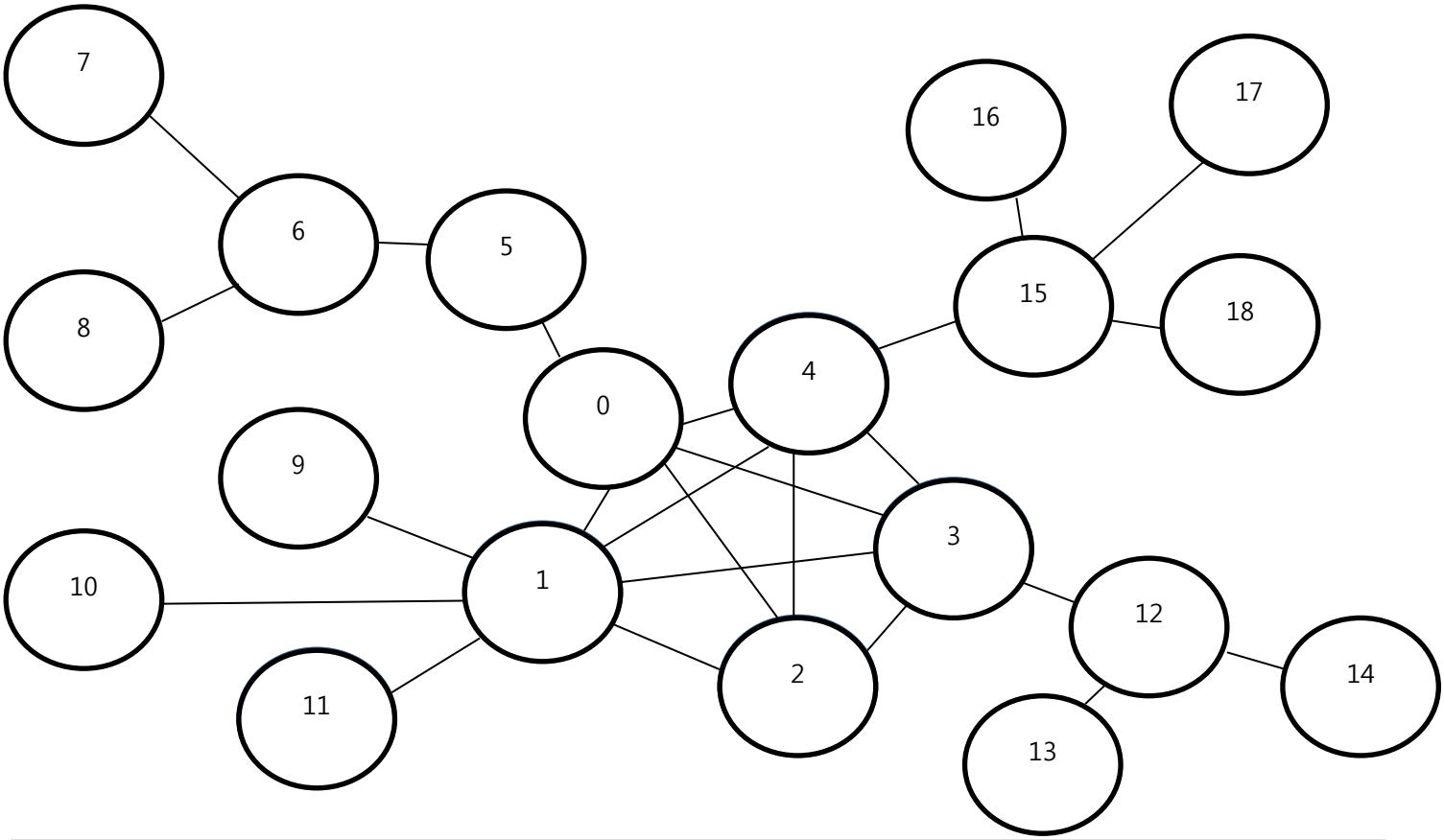
C) 이들은 모두 TCP/IP 의 계층 위에 attach 되기 때문에, TCP/IP 프로토콜 모델에 맞도록 트래픽을 생성해야 한다.

D) 송신 에이전트와 수신 에이전트가 결정되고 서로 연결이 되고 나면, attach 된 애플리케이션 서비스가 트래픽을 발생시켜 네트워크 활동이 이루어진다.

E) 애플리케이션 서비스에 관한 내용은 ~/ns-allinone-2.35/ns-2.35/apps/ 에 C++코드가 있고, ~/ns-allinone-2.35/ns-2.35/tcl/ ftp, http 등의 OTCL 스크립트가 있다

D. 네트워크 시뮬레이션 : LAN

1) Topology :



설명

먼저, 모양은 0-5번 노드를 오각형과 ★모양으로 이어서 0-5노드에 연결된 노드들 간에 통신이 가능하도록 구성하였다. 특히, 이 구성에서 확인하고자 하는 것은 2번 노드는 실제로 최단거리로 통신하기 위해서는 어느 패킷이라도 거치지 않아야 할 노드이기 때문에 시뮬레이션에서도 이 점이 반영되는 지이다.

두 번째로, 각 링크에 따라 전송 속도를 다르게 했을 때 nam을 통해서는 어떻게 속도차이가 반영되는 지를 확인하는 것이다.

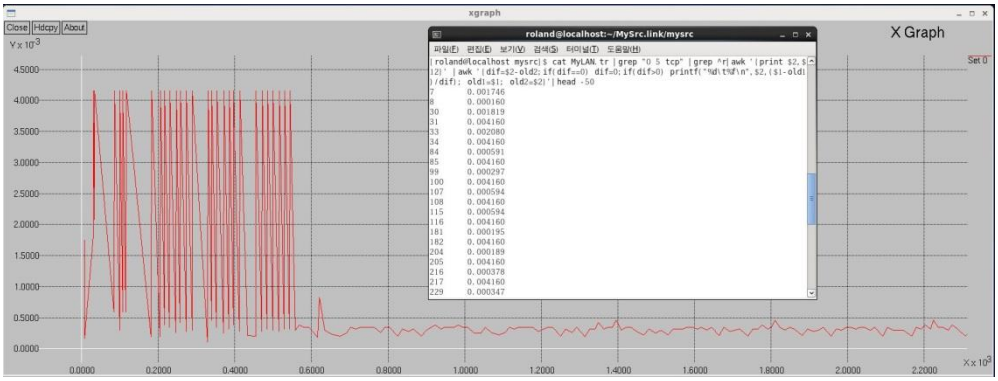
마지막으로, 특정하게 0->5 노드로 가는 패킷에 trace 기능을 통해서 다음 packet id를 가지는 패킷이 5번 노드로 들어오는 데 걸리는 평균 시간 간격을 분석한다. 그 평균 시간 간격이 나타내는 의미는 네트워크 전체의 패킷 id에서 0 -> 5로의 이벤트가 발생하는 시간 간격이다. 즉, 짧을수록 0->5로의 이벤트가 자주 발생하고 있다는 것이고, 길수록 이벤트가 띄엄띄엄 발생하고 있다는 뜻이다.

링크의 속성과 어떤 노드와 어떤 노드가 연결되어 있는 지는 글로 설명하는 것보다 소스코드를 참고하면서 보는 것이 좋을 것이라 판단되어 따로 기재하지 않는다.

시뮬레이션 시간은 15초로 하였다. 추가로 설명하면 nam은 시뮬레이션 속도 조절이 가능하기 때문에 소스코드에서 정한 시뮬레이션 시간이 15초라고 하여서 nam을 통한 시뮬레이션 시간이 15초라는 것은 아니다.

2） Analyzation :

- nam 을 이용한 시각화 된 시뮬레이션
  - 문서와 함께 있는 nam.avi 를 참고하시기 바랍니다.
- awk 명령과 xgraph 를 통한 결과 분석 및 평가



- 결과 분석
  - 출발지 0 번 노드에서 목적지 5 번 노드로 가는 tcp 패킷 중에 5 번 노드에 도착한 패킷의 trace 를 추출했다.
  - 그 trace 에서 awk 명령을 이용해서 packet id 에 따라, 다음 packet id 의 패킷이 도착하는 데 걸리는 평균 시간 간격을 그래프로 나타낸 것이다.
  - Packet id 가 약 600 번이 되기 전까지 다음 패킷이 도착하는 데 평균적으로 걸리는 시간의 변동이 매우 큼을 알 수 있다.
  - 즉, 이것은 전체 네트워크에서 초기에는 0 -> 5 로의 이벤트가 불안정하게 발생하다가 일정 시간이 지나면 대략 일정한 간격으로 이벤트가 발생하게 된다는 것을 의미한다.
- 시뮬레이션 평가
  - 시뮬레이션 상의 문제와 그 문제점을 개선할 방법은 무엇인가?
    - 라우팅에 관한 내용을 구현하지 못하였다. 이 점은 앞으로 라우팅 프로토콜에 대해서 디폴트 기능(유니캐스트)가 아니고 추가적인 요소가 필요한 경우에 더 공부해서 구현할 것이다.
    - 네트워크 표준에 따라 구현하지 못하였다. LAN 은 IEEE 802.3 표준에 대해 공부하고 NS2 에 미리 구현된 LAN 을 공부하여 쉽게 구현할 수 있는데, 아직 표준에 대해서 공부하려면 더 많은 네트워크 지식이 필요하여 구현하지 못하였다.

5. 결론

A. 향후 NS2 를 이용한 연구 방향 제시

1) 쉘 스크립팅 또는 프로그래밍을 통한 네트워크 시뮬레이션 및 네트워크 분석 툴 개발

A) 네트워크 분석 툴은 여러 개가 있지만, 어떤 한 용도에 대한 툴은 많지 않다. 즉, 사용자에게 편리한 툴들이 많이 있지 않기 때문에, 불편 사항들을 기반으로 한 간단한 스크립트나 실행파일을 만드는 것도 의의가 있다고 생각한다.

B) NS2 의 최적 환경이 UNIX 기반 OS 이기 때문에 NS2 의 분석 툴을 개발하려면 역시 쉘 스크립팅 또는 프로그래밍으로 개발해야 한다.

C) 준비정도

(1) 방대한 양의 데이터를 다루기 때문에 속도적인 측면도 고려해야해서, 최대한 내부 명령어를 사용하기 위해 리눅스 내부명령어와 리눅스 쉘스크립팅에 대해서 공부 중인 단계이다.

(2) 이 연구방향에 동기가 된 소스 코드 예제이다

```
cat nstest1_out.tr |grep "0 1 cbr" |grep ^r|awk '{print $2,$11}' |awk '{dif=$2-old2;if(dif==0) dif=0;if(dif>0) printf("%d₩t5f₩n",$2,($1-old1)/dif); old1=$1}; old1=$1; old2=$2}' > example.txt
```

2) 네트워크 해킹 공격에 대한 자세한 시뮬레이션을 제공하는 툴로써 사용

A) 계획하고 있는 네트워크 해킹 시나리오를 NS2 를 통해서 시뮬레이션하면, 어느 시뮬레이션 툴보다도 신뢰성있는 시뮬레이션 데이터를 추출할 수 있고, 또한 보조 툴들을 통해서 결과 자료를 원하는 형태로 분석할 수 있다는 장점이 있다..

3) 연구 계획

A) 연구해 볼 네트워크 해킹 공격에 대한 공격 시나리오를 구성해본다. 또한 네트워크 시뮬레이션 방법론에 대해 조금 더 공부해서 원하는 시나리오에 부합한 시뮬레이션을 할 수 있도록 준비한다.

B) 여러 시나리오에 대한 개략적인 내용을 내부 세미나나 포럼을 이용하여, 타인으로부터 검증을 받는다.

C) 검증된 시나리오들을 시뮬레이션 작성 절차에 따라서 시뮬레이션 한다.

6. 부록

A. 참고 문헌

1) 배성수,한종수, "네트워크 시뮬레이터(NS2 기초와 활용)", 세화 출판

2) Teerawat Issariyakul, Ekram Hossain, "Introduction to Network Simulator NS2", Springer

3) Kevin Fall, "The ns Manual – The VINT Project", A Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC.

4) 논문 검색 사이트 : http://www.dbpia.co.kr/

B. 소스 코드 : LAN.tcl

```
#option setting
set opt(tr)       "MyLAN.tr"
set opt(namtr)   "MyLAN.nam"

set opt(start)   0.0
set opt(stop)    5.0
set opt(node)    19

set opt(qType)   DropTail

set opt(tcp)     TCP/Reno
set opt(sink)    TCPSink
set opt(connection) 10

set opt(app)     FTP



#call for turning tracing on
proc create-trace {} {
        global ns opt
        global tf nf

        set tf [open $opt(tr) w]
        $ns trace-all $tf
```

```
        set nf [open $opt(namtr) w]
        $ns namtrace-all $nf
}


#call for create topology
proc create-topology {} {
        global ns opt node_

        for {set i 0} {$i < $opt(node)} {incr i} {
                set node_($i) [$ns node]
        }

        $ns duplex-link $node_(0) $node_(1) 2Mb 20ms $opt(qType)
        $ns duplex-link $node_(0) $node_(2) 2Mb 20ms $opt(qType)
        $ns duplex-link $node_(0) $node_(3) 2Mb 20ms $opt(qType)
        $ns duplex-link $node_(0) $node_(4) 2Mb 20ms $opt(qType)
        $ns duplex-link $node_(1) $node_(2) 2Mb 20ms $opt(qType)
        $ns duplex-link $node_(1) $node_(3) 2Mb 20ms $opt(qType)
        $ns duplex-link $node_(1) $node_(4) 2Mb 20ms $opt(qType)
        $ns duplex-link $node_(2) $node_(3) 2Mb 20ms $opt(qType)

        $ns duplex-link $node_(2) $node_(4) 2Mb 20ms $opt(qType)
        $ns duplex-link $node_(3) $node_(4) 2Mb 20ms $opt(qType)

        $ns duplex-link $node_(0) $node_(5) 2Mb 2ms $opt(qType)
        $ns duplex-link $node_(5) $node_(6) 2Mb 2ms $opt(qType)
        $ns duplex-link $node_(6) $node_(7) 2Mb 2ms $opt(qType)
        $ns duplex-link $node_(6) $node_(8) 2Mb 2ms $opt(qType)

        $ns duplex-link $node_(1) $node_(9) 2Mb 2ms $opt(qType)
        $ns duplex-link $node_(1) $node_(10) 2Mb 2ms $opt(qType)
        $ns duplex-link $node_(1) $node_(11) 2Mb 2ms $opt(qType)

        $ns duplex-link $node_(3) $node_(12) 2Mb 2ms $opt(qType)
        $ns duplex-link $node_(12) $node_(13) 2Mb 2ms $opt(qType)
```

```
        $ns duplex-link $node_(12) $node_(14) 2Mb 2ms $opt(qType)

        $ns duplex-link $node_(4) $node_(15) 2Mb 2ms $opt(qType)
        $ns duplex-link $node_(15) $node_(16) 2Mb 2ms $opt(qType)
        $ns duplex-link $node_(15) $node_(17) 2Mb 2ms $opt(qType)
        $ns duplex-link $node_(15) $node_(18) 2Mb 2ms $opt(qType)
}
#call for set scenario
proc set-scenario {} {
        global ns opt node_ tcp ftp

        set tcp(0) [$ns create-connection $opt(tcp) $node_(7) $opt(sink) $node_(11) 0]

        set tcp(1) [$ns create-connection $opt(tcp) $node_(8) $opt(sink) $node_(13) 1]
        set tcp(2) [$ns create-connection $opt(tcp) $node_(9) $opt(sink) $node_(14) 2]
        set tcp(3) [$ns create-connection $opt(tcp) $node_(10) $opt(sink) $node_(16) 3]
        set tcp(4) [$ns create-connection $opt(tcp) $node_(11) $opt(sink) $node_(17) 4]
        set tcp(5) [$ns create-connection $opt(tcp) $node_(13) $opt(sink) $node_(18) 5]
        set tcp(6) [$ns create-connection $opt(tcp) $node_(14) $opt(sink) $node_(7) 6]
        set tcp(7) [$ns create-connection $opt(tcp) $node_(16) $opt(sink) $node_(8) 7]
        set tcp(8) [$ns create-connection $opt(tcp) $node_(17) $opt(sink) $node_(9) 8]
        set tcp(9) [$ns create-connection $opt(tcp) $node_(18) $opt(sink) $node_(10) 9]
        for {set i 0} {$i < $opt(connection)} {incr i} {
                set ftp($i) [$tcp($i) attach-app $opt(app)]
        }
}


#call at start
proc Start {} {
        global ns ftp opt
        for {set i 0} {$i < $opt(connection)} {incr i} {
                $ns at $opt(start) "$ftp($i) start"
        }
}
```

```
#call at finish
proc finish {} {
        global ns opt tf nf

        $ns flush-trace
        close $tf
        close $nf
        exec nam $opt(namtr)

        exit 0
}

## MAIN ##

set ns [new Simulator]



set tf, nf
create-trace



set node_(0) 0
create-topology



set tcp(0) 0
set ftp(0) 0
set-scenario



$ns at $opt(start) "Start"

$ns at $opt(stop) "finish"

$ns run
```