ITE4004 Network Protocol course

Practice Report

2016026080 한다진

* **Experiment environments**

OS Window 7

Tools OPNet

* **OPNet**

OPNet 은 C언어로 설계된 이벤트 기반 GUI simulator 이다. Network, Node, Process, DES 등을 구성하여, 우리가 테스트하고자 하는 네트워크 망을 설계할 수 있고, 이를 테스트 하여 네트워크 망의 성능이 어떻게 나오는 지를 확인할 수 있다. 또한 TCP\_manager 단의 application source code를 수정하여 내부 패킷에 따라 네트워크 망이 어떤식으로 동작할 지 구체적으로 지정해 줄 수 있다. 전반적인 사용 방식은 아래와 같다.

1. 네트워크 모델을 구성하고,
2. 시뮬레이터를 통해 확인하고자 하는 값을 DES를 통해 설정한 다음,
3. 시뮬레이터를 돌리고
4. 그 결과를 확인한다.

사용중인 노트북이 mac os 이여서, Aws의 window 7 환경에서 테스트를 진행하려 했으나, OPNet을 돌릴 수 있도록 환경설정을 하는 도중에 에러가 많이 발생하였고, 프로그램을 돌린 뒤에도 버벅거리는 현상이 계속 발생하여, 조교님이 주신 vmware 환경 파일을 통해, 가상환경 내에서 실습을 진행했다.

전체 프로젝트는 여러 개의 시나리오로 구성이 된다. 한 프로젝트에는 유사한 구조의 네트워크 모델을 구성한다. 그리고 시나리오별로 네트워크 모델 내부 오브젝트들의 특성을 조금씩 바꿔가며 시뮬레이션을 한다. 시나리오 별 시뮬레이션 결과의 차이를 확인하며 실험결과를 확인할 수 있다.

자주 등장하는 용어는 아래와 같다.

* AP 기지국에 해당된다.
* STA 단말기, 혹은 클라이언트에 해당된다
* Server 데이터의 소스포인트인 서버에 해당된다.
* Attribute 네트워크 모델을 구성하는 오브젝트에 특성을 부여할 때 쓴다.
* DES 시뮬레이션을 돌릴 때, 결과값으로 확인할 요소를 설정할 때 쓴다.

**1st week – OPNet tutorial**

첫번째 실습은 OPNet을 사용하는 방식을 배우는 튜토리얼이다. 프로젝트를 생성할 때에는, 어떤 네트워크 환경에서 돌아갈 것인지 설정을 하고, 사용할 기술들을 라이브러리 추가하듯이 체크하여 프로젝트에서 사용할 수 있게끔 해준다.

프로젝트를 office network 환경으로 새로 만든다. 이번 실습에서는 SM\_Int\_Model\_List 라이브러리의 오브젝트들을 사용하여 네트워크 모델을 만들 것이다. 그래서 SM\_Int\_Model\_List 라이브러리를 추가해준다.

네트워크 모델을 만드는 방식은 rapid configuration 기능을 통해 만드는 방식과 palette를 통해 만드는 방식이 있다. Rapid configuration 기능은 보다 더 규칙적인 모델을 만드는데 편하고, 팔레트는 직관적으로 모델을 구성하는 것을 눈으로 확인하기 용이하다는 장점이 있다.

**Scenario 1 – star, ethernet**

스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

1번 시나리오에서는 star 구조의 네트워크 망에서 이더넷을 통해 데이터가 송수신 하는 것을 시뮬레이션 했다. Global DES 를 수정하여 모든 오브젝트의 DES를 수정했다. 이더넷을 통해 데이터가 서버 -> 기지국 -> 수신서버로 유선망으로 전송되는 것을 확인했다. 그리고 결과를 확인하는 방식에 대해서 간단히 배웠다.

**Scenario 2 – star, ethernet, router**

스크린샷이(가) 표시된 사진

자동 생성된 설명스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

두번째 시나리오는 첫번째 시나리오를 복사하여 그 위에 새로운 네트워크 모델 오브젝트를 추가하는 방식으로 진행된다. 새로운 star 구조의 네트워크를 생성하고, 라우터를 만들어서 두 star 구조의 네트워크망을 이어준다. 이후 결과를 확인했을 때, 새로 생성한 네트워크 망에서 데이터로드가 더 많은 부하(더 많은 데이터 전송)가 생기는 것을 확인할 수 있었다. 첫번째 star 구조의 네트워크에서보다 두번째 star 구조의 네트워크는 서버에 통신하려는 노드의 수가 더 많다. 그래서 서버에 걸리는 부하가 커진다.

**2nd week – Wireless**

Wlan 환경에서의 네트워크 구조를 시뮬레이션 해보는 실습이다. 이전 실습과 마찬가지로 office 환경의 네트워크를 선택하고, wireless\_lan, link, ethernet 라이브러리를 활용하여 네트워크 모델을 구성한다.

**Scenario 1 – wlan**

스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

첫번째 시나리오는 단순히 wlan 으로 연결이 되어있는지 확인하는 시나리오다. 클라이언트에서 데이터를 보낸 것을 서버에서 확인한다. 서버와 클라이언트가 전송한 데이터와 서버측에서 수신한 데이터가 같다. 클라이언트는 tcp, http, mac 방식으로 데이터를 각각 전송하고, 서버는 이를 수신하는데, 수신된 데이터의 개형이 모두 동일하다. 이것으로 모든 방식을 통해 데이터가 잘 전송된 것을 확인할 수 있고 계층간 저해요소가 없는 것을 확인했다.

Server-wired 구조의 경우 delay 가 발생하지 않지만, 전파를 전송매체로 사용한다면 delay가 많이 발생하게 된다. 통신할때 방해요소가 많아지기 때문에 딜레이가 발생하는 것을 확인할 수 있다.

**Scenario 2 – RTS/CTS, IFS fragmentation**

스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

시나리오2는 시나리오1번에서 트래픽을 추가하여 시뮬레이션을 돌린 것이다. OPNet에서의 트래픽은, 시뮬레이션을 돌릴 때 수행해야 할 task를 부여하는 것이다. Task를 부여한 후, 여러 상황에 대해서 성능변화가 어떻게 일어나는 지를 확인한다. 우리가 확인하고자 하는 것은 같은 task를 처리할 때, RTS/CTS 기법과 IFS 기법의 적용상태에 따라서 전체 성능이 얼마나 변화하는가 이다.

네트워크 상에서 패킷이 같은 경로상으로 갈 때, 패킷끼리 충돌해서 데이터가 변형 및 손상되는 경우가 있다. 그래서 현재 시간간격이 누구의 통신주기인지, 누가 통신을 어떤 것으로 하고 있는지 등을 눈치싸움을 하면서 진행이 된다. 그 대표적인 방식이 IFS 와 RTS/CTS 이다.

* IFS 란 프레임을 전송할 때 패킷간 충돌을 피하기 위해 프레임 사이에 대기하는 시간 간격을 뜻한다. 즉 fragmenation 데이터를 전송할 때, 프레임 전송 사이사이에 대기를 하며 데이터를 전송하는 방식이다.
* RTS/CTS는 채널 선점을 하여 데이터를 송수신하는 일종의 three-hand-shaking 기법과 유사한 기법이다. 송신부가 rts 를 보내고 수신부로부터 cts 를 받게 된 다음에 정보전송을 시작하는 방식이다. CSMA/CA 를 구현하는 방법 중 하나다. TCP 성능저하를 발생시킨다는 단점이 있지만 hidden terminal 문제를 해결할 수 있는 방법이다.

실습은 아래의 경우에 대해서 각각 시뮬레이션을 돌린 후 결과를 비교하며 진행한다.

1. 일반 연결상태,
2. RTS/CTS 적용상태,
3. IFS 적용 상태,
4. RTS/CTS & IFS 적용 상태

아무런 기법도 적용하지 않은 것이 가장 높은 성능을 보였고, 기법을 적용할 때 마다 성능이 하락하는 것을 확인했다. 연결상태에 아무런 문제가 일어나지 않는 이상적인 상황에서는 RTS/CTS 나 IFS 기법을 사용하지 않는 것이 효과적이라는 것을 확인했다.

**3rd week – Jamming and Hidden Terminal**

이번에는 전파 송수신에 방해가 되는 요소가 존재하는 경우에, 데이터 전송 측면에서 성능변화가 어떻게 일어나는 지를 확인한다.

**Scenario 1 – Jamming**

스크린샷이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

Jamming 이란, 특정 주파수의 전파의 송수신을 방해하기 위해 같은 주파수의 신호를 보내는 것을 의미한다. 시나리오1 에서는 재밍 요소를 넣어주어서 데이터 송수신을 방해하고, 재밍이 발생하는 구간에서 데이터 송수신 성능이 얼마나 떨어지는 지를 확인한다. TCP 프로토콜은 내부적으로 동작하는 기능이 많기 때문에, jamming 에 따른 성능하락을 확인하기 위해 udp 프로토콜로 실습을 진행하였다. 오른쪽 그림을 보면 재밍이 발생한 시점에서 성능이 대략 1/3 로 떨어지는 것을 확인할 수 있었다.

**Scenario 2 – Jamming - IFS**

스크린샷이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

IFS 기법을 적용한 경우 성능저하폭이 크게 감소하는 것을 확인할 수 있었다. IFS 기법을 적용한 경우, 일반적인 상황에서는 성능이 더 나쁘지만, Jamming 이 적용된 구간에서는 오히려 더 좋은 성능을 보였다.

**Scenario 3 – Hidden Terminal Problem**

스크린샷이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

Hidden Terminal Problem 이란 상대방이 어떤 채널을 사용하는 지 몰라서 우연히 동일한 채널을 겹쳐서 사용하게 되는 경우를 뜻한다. 이 경우 동일한 채널을 사용하여 데이터를 전송하려 하기 때문에 jamming 과 유사하게 데이터 전송 성능이 떨어지는 모습을 보인다.

STA\_0이 이동성이 있도록 생성했고, 시간에 따라 sta 와 sta\_0 이 서로 인식할 수 없는 거리가 되었을 때 전체 네트워크 성능이 떨어지도록 네트워크 모델을 구성했다. 시뮬레이션 결과 sta간 거리가 서로 인식할 수 없을 만큼 멀어진 시점부터 데이터 전송 성능이 떨어지는 것을 확인했다. 이는 동일한 채널을 사용하려고 하기 때문에 발생한 것이다.

**Scenario 4 – Hidden Terminal Problem – RTS**

스크린샷이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

RTS/CTS 기법을 적용한 경우 성능저하폭이 크게 감소하는 것을 확인할 수 있었다. RTS/CTS 기법을 적용한 경우, 일반적인 상황에서는 성능이 더 나쁘지만, Hidden Terminal Problem 이 적용된 구간에서는 오히려 더 좋은 성능을 보였다.

**4th week**

이번 실습은 OPNet 에서 실제 패킷을 다루는 코드를 수정하며 실습을 했습니다. 특히 tcp 단에서 동작하는 코드를 수정하며 시뮬레이션을 해 보았습니다.

이번 실습의 경우, open child process model 을 하면서 실습이 진행이 되었는데, 전반적으로 실습을 진행하기 힘든 상황이었습니다. 초반 wlan\_mac에서의 노드들의 색깔이 실습 자료에 나와있는 것과 달랐고, 이후 컴파일이 적용되지 않는 문제가 발생했고, 시뮬레이션을 돌렸는데 target packet을 찾지 못한다는 에러가 반복적으로 발생했습니다. 그래서 2주차 실습 코드를 다시 복사해서 실습을 진행하려했습니다만, 이전까지 돌렸던 모든 프로젝트가 동작하지 않는 현상이 발생했습니다. 결국, 과거에 백업으로 만들어 놨던 소스 코드를 압축을 풀어 다시 실행했으나 마찬가지로 동작하지 않았습니다. Vmware로 가상환경을 다시 설정하고, 백업으로 만들어 놨던 소스코드를 압축 해제 후 실행시키니, 그제서야 정상적으로 동작하는 것을 확인했습니다.

이후 다시 차근차근 옵션을 재설정하며 시나리오 1의 경우는 정상적으로 동작은 시켰으나, 이것이 정확히 어떤 의미를 가지고 있는 실습인지는 미쳐 파악하지 못했습니다. 따라가기 급급한 나머지 실습의 핵심적인 내용을 이해하지 못했습니다. 패킷의 사이즈를 확인하는 실습이었습니다.

**Scenario 1**

스크린샷, 컴퓨터이(가) 표시된 사진

자동 생성된 설명스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명스크린샷, 모니터, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

1번 시나리오는 무선랜의 성능 개선을 위해 tcp 단에서 back off 에 쓰이는 시간을 줄이는 것이었습니다. 그래서 back off 를 진행하는 부분의 코드를 수정하며 결과를 확인하고자 했습니다. 여기서 back off 란 데이터 전송 중 문제가 생기는 것을 대비해서 버퍼처럼 데이터를 잠시 담아두는 구간을 의미합니다. 다만 실험 결과 전반적으로 성능이 하락하는 결과가 나왔습니다.

**Scenario 2**

스크린샷, 컴퓨터이(가) 표시된 사진

자동 생성된 설명스크린샷, 컴퓨터, 모니터이(가) 표시된 사진

자동 생성된 설명스크린샷, 컴퓨터, 모니터, 책상이(가) 표시된 사진

자동 생성된 설명

2번 시나리오는 수신 패킷의 사이즈 변수를 추가하는 것입니다. OPNet에는 패킷의 바디 사이즈를 기본적인 DES를 통해 확인할 수 없습니다. 그래서 직접 패킷사이즈를 관찰할 수 있는 변수를 생성하고 이를 통해 시뮬레이션 돌렸을 때 패킷의 사이즈를 관찰하고자 했습니다. 변수를 정의하고 핸들변수를 등록하여 DES에서 사용할 수 있게끔 했습니다.

시나리오 2번에서는, 우선 노드의 색이 실습 자료의 색과 달라서 전반적으로 진행하기 힘들었습니다. 색깔은 상관이 없다는 말씀을 들었지만, 1번 시나리오에서도 초반 노드색이 달랐을 때에는 진행이 불가했다가, 초기 세팅을 다 처음부터 다시하고 노드 색이 실습 피피티에 나오는 노드색과 동일해졌을 때 비로소 시뮬레이션이 정상적으로 동작하는 것을 확인할 수 있었습니다. 2번 시나리오에서는 컴파일까지는 정상적으로 진행했으나, DES에 설정해준 ReceivedFrameBody를 찾지 못해서 에러가 발생했습니다.