Programming Assignment

Improving the performance TCP Reno algorithm

Due date : Dec. 8, 2023

과제 설명

- 목표
  - 리눅스 커널에 구현된 TCP Congestion 알고리즘 중 하나인 TCP Reno 알고리즘을 개선한 후 직접 모듈을 올려 성능 분석을 시행함.

- 수행 환경
  - Ubuntu 20.04 LTS (VM 설치 관련은 **목차 0,1** 참조)
  - 예시 네트워크 환경 (네트워크 환경 구축은 **목차 2,3,4** 참조)
    - 10개의 Connection, Throughput : 1Mbps, RTT : 200ms, 에서 통신시 Fairness, Link Utilization, Congestion Control 측면에 대해 분석

과제 설명

- 과제 상세 목표
  - 특정 네트워크 조건에서 Reno 혼잡 제어의 문제점을 분석 후 Reno 혼잡 제어 알고리즘을 수정하여 그 결과를 실험을 통해 확인하고 실험 결과를 분석한다.

- 과제 수행 과정
  1. 네트워크 조건 설정
  2. 설정한 네트워크 조건에서 Reno 성능 분석
  3. Linux의 Reno 혼잡 제어를 수정
  4. 수정한 Reno를 설정한 네트워크 조건에서 동작시킨 후 결과 확인
  5. 실험 결과 분석 후 보고서 작성

** 과제를 수행시, 주어진 **reno_custom.c** 를 활용해 코드 작성 및 성능 개선을 권장함.
리눅스 커널 소스에 포함된 Reno 혼잡제어 코드인 tcp_cong.c에는 custom 모듈에 관한 코드가 없기 때문이며, /net/ipv4/tcp_cong.c를 직접 수정하여 과제 수행해도 무방함.

과제 설명

- 다양한 네트워크 조건의 예시 (자유롭게 설정 가능)
  - 10~1000개 이상의 다수의 Concurrent TCP Connection이 있을 때
  - 고대역, 높은 지연 특성을 가진 네트워크
  - Link Error로 인한 높은 패킷 손실이 있는 환경
  - 지연의 편차가 큰(Jitter) 네트워크 환경
- 하나의 네트워크 조건에 대해서만 기존 Reno와 비교분석하면 되나 수정한 알고리즘의 강건성(Robustness)를 보이기 위해 다수의 네트워크 조건에서 분석 시 가산점 획득
- Fairness 측면의 확인하기 위해서 5개 이상의 TCP Connection이 있는 조건에서 실험이 요구됨

과제 설명

- TCP 성능 평가 지표
  - 필수적으로 아래의 성능을 확인하여야 하며 이외에도 다양한 네트워크 성능 평가 지표를 활용할 수 있음
    - Link Utilization
    - Fairness
    - Latency
  - 모든 평가 지표가 개선될 필요는 없음

과제 설명

- 제출물
  - 1) C code
    - /net/ipv4/tcp_cong.c (36p 참조)의 기능을 개선한 본인의 Custom TCP congestion 알고리즘 코드
  - 2) 과제 보고서
    - 6 페이지 이상(초과 분량 제한 없음), Word, 한글 자유롭게 작성 후 PDF로 제출
    - 보고서 구성
      1. 설정한 문제 상황과 해당 상황에서 Reno의 실험 결과 분석
      2. 문제 해결 전략
      3. TCP Reno 코드 수정 사항 설명
      4. 성능 분석 수행 및 분석

과제 설명

- 과제 기한
  - **~ 2023/12/08 23:59:00**
  - 지연 제출시, 하루에 **25% 감점**. 3일 후 제출시 **미제출 처리**
    - Ex) 과제 점수 80점인 학생이,
      12/9일에 제출한 경우 -> 80*0.75=60
      12/10일에 제출한 경우 -> 80*0.5=40
      12/11일에 제출한 경우 -> 0점이 아닌 미제출 처리

과제 설명

- 과제 설명 문서 참고
  - **0.Ubuntu image 다운로드(20.04), 1.VM 설치**를 참고하여 ubuntu 20.04 환경 세팅
  - **2.Mininet 소개, 3. 설치, 4. Demo**를 참고하여 Mininet 에뮬레이터를 통한 실제 네트워크 토폴로지 구축.
    - 해당 Demo 예제를 통해 간단한 소켓 통신을 수행할 수 있으며, Mininet을 통해 구현한 네트워크 토폴로지로 차후 TCP Congestion 알고리즘 성능 측정에 활용
  - **5.Building Custom TCP Network Module**를 참고하여 간단한 custom TCP Congestion algorithm 예제를 실제 Ubuntu OS에 모듈로 삽입해 적용되는지 확인.
    - 해당 목차 학습을 통해 추후 구현한 TCP_Reno 혼잡제어 알고리즘의 개량버전을 실제 OS에 모듈로 올릴 수 있음.
    - 그후 사전에 구축한 네트워크 토폴로지에서 혼잡제어 알고리즘의 동작을 확인 및 분석 가능해 과제 수행.

0. Ubuntu image 다운로드(20.04)

1. VM 설치

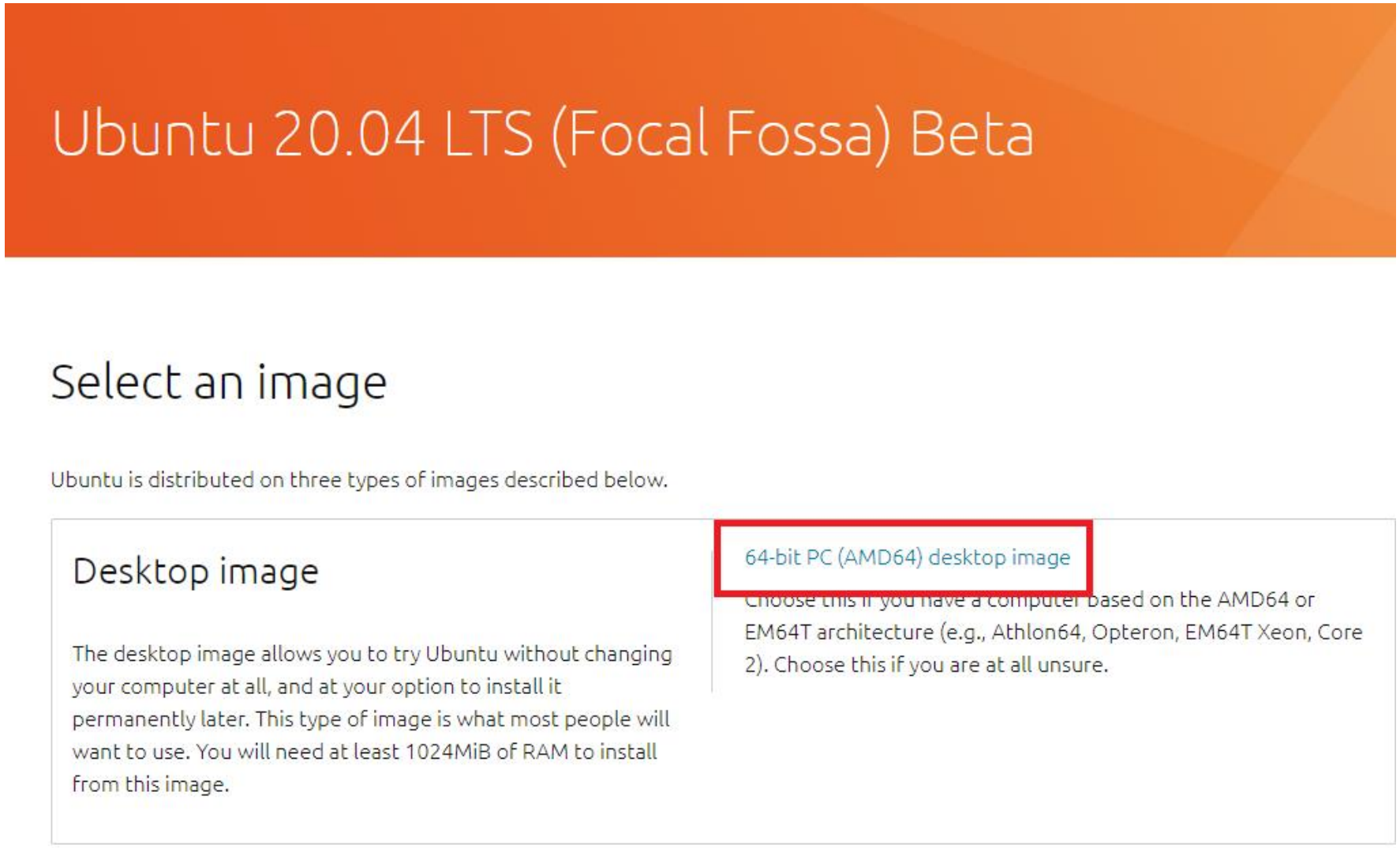
2. Mininet 소개
3. Mininet 설치
4. Mininet Demo

5. Building Custom TCP Network Module

6. 참고자료
Appendix – TCP Reno Code Analysis

0. Ubuntu image 다운로드(20.04)

http://old-releases.ubuntu.com/releases/focal/ - 반드시 해당 링크의 20.04 버전의 Ubuntu로 수행

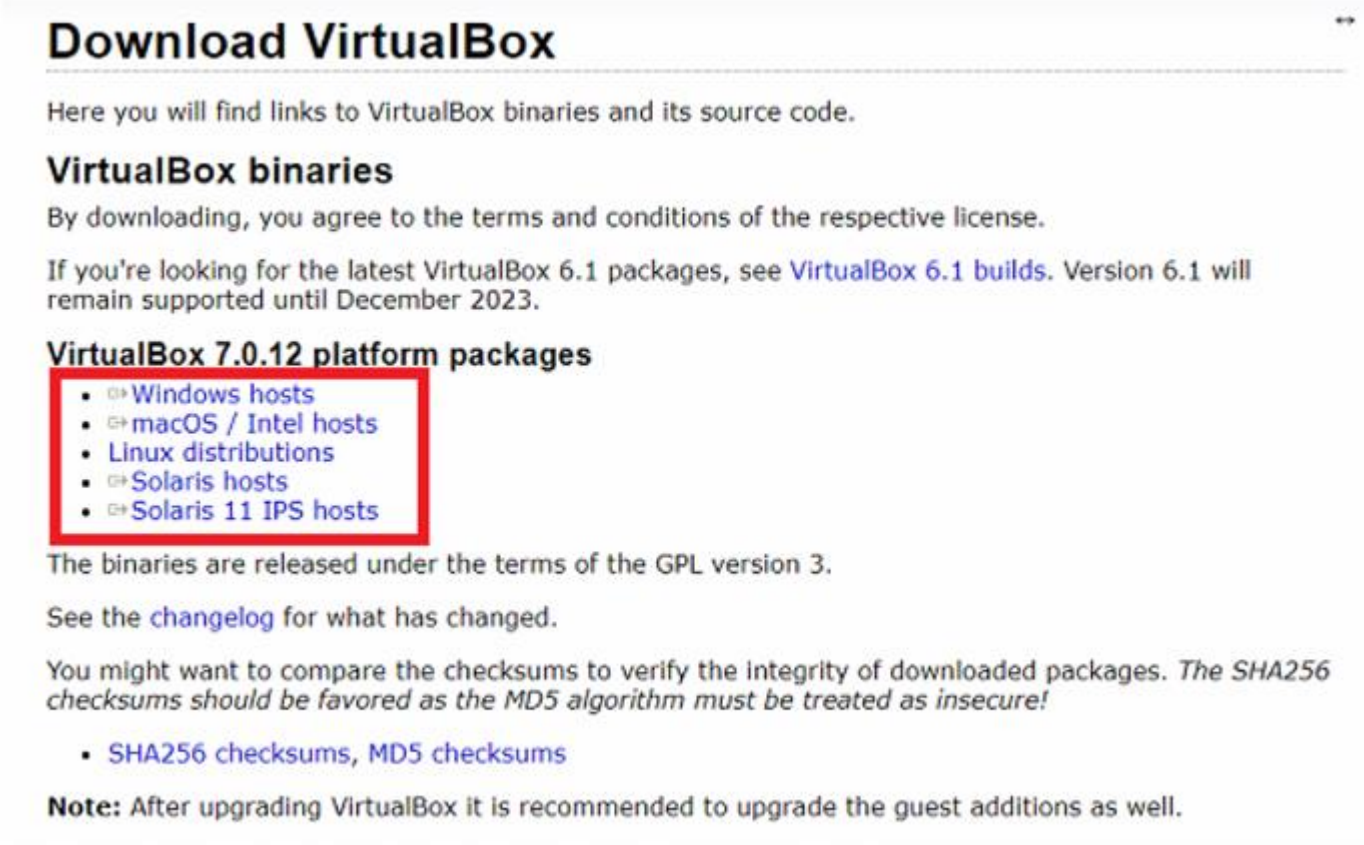


1. VM 설치(Virtual Box)

https://www.virtualbox.org/ 접속

1. VM 설치(Virtual Box)

OS에 맞게 다운로드



Download VirtualBox

Here you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

By downloading, you agree to the terms and conditions of the respective license.

If you're looking for the latest VirtualBox 6.1 packages, see VirtualBox 6.1 builds. Version 6.1 will remain supported until December 2023.

VirtualBox 7.0.12 platform packages

- Windows hosts
- macOS / Intel hosts
- Linux distributions
- Solaris hosts
- Solaris 11 IPS hosts

The binaries are released under the terms of the GPL version 3.

See the changelog for what has changed.

You might want to compare the checksums to verify the integrity of downloaded packages. *The SHA256 checksums should be favored as the MD5 algorithm must be treated as insecure!*

- SHA256 checksums, MD5 checksums

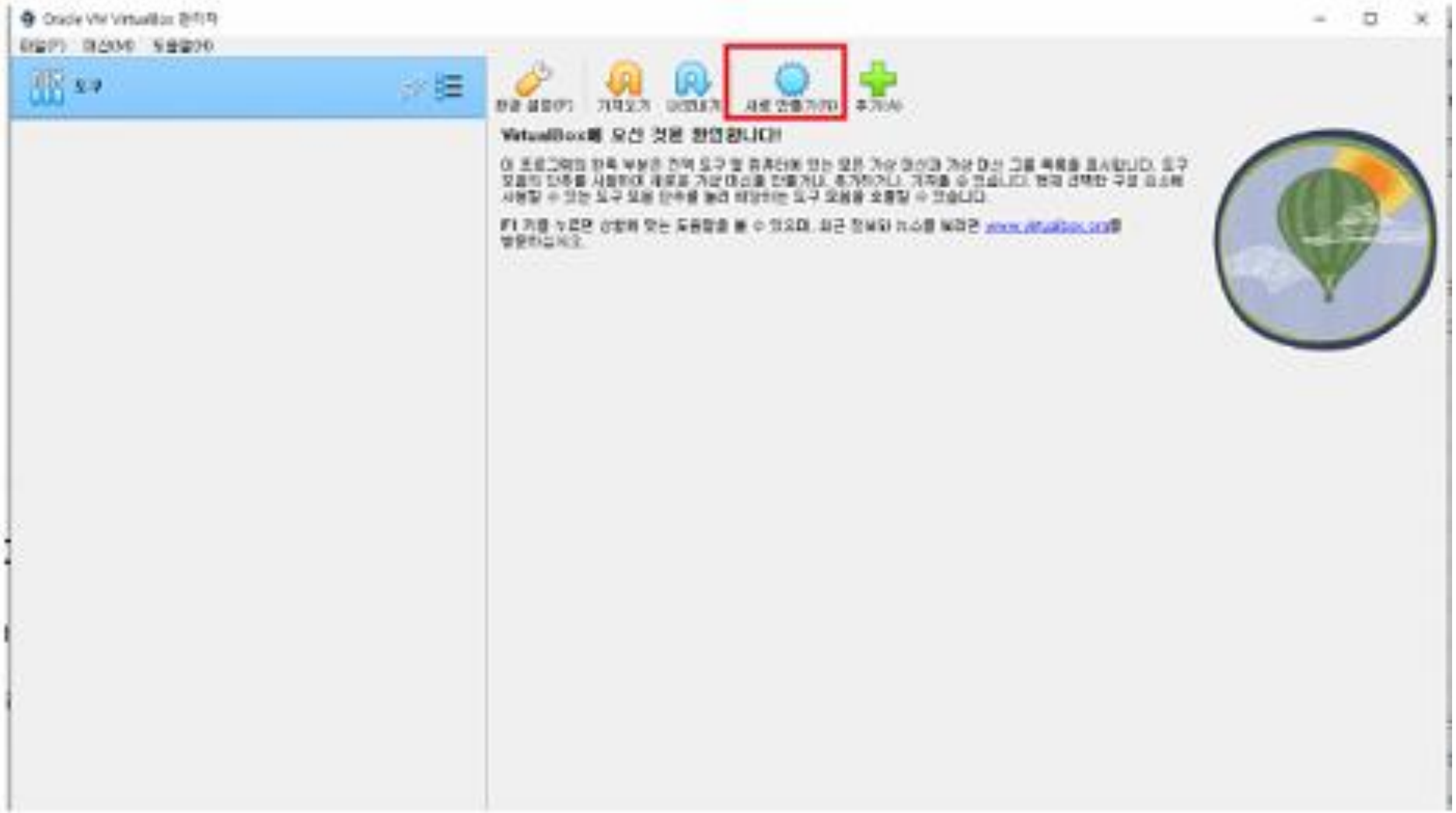
Note: After upgrading VirtualBox it is recommended to upgrade the guest additions as well.

1. VM 설치(Virtual Box)

다운로드- 계속 next, ok, yes, install 으로 진행(기본 설정 다운로드)

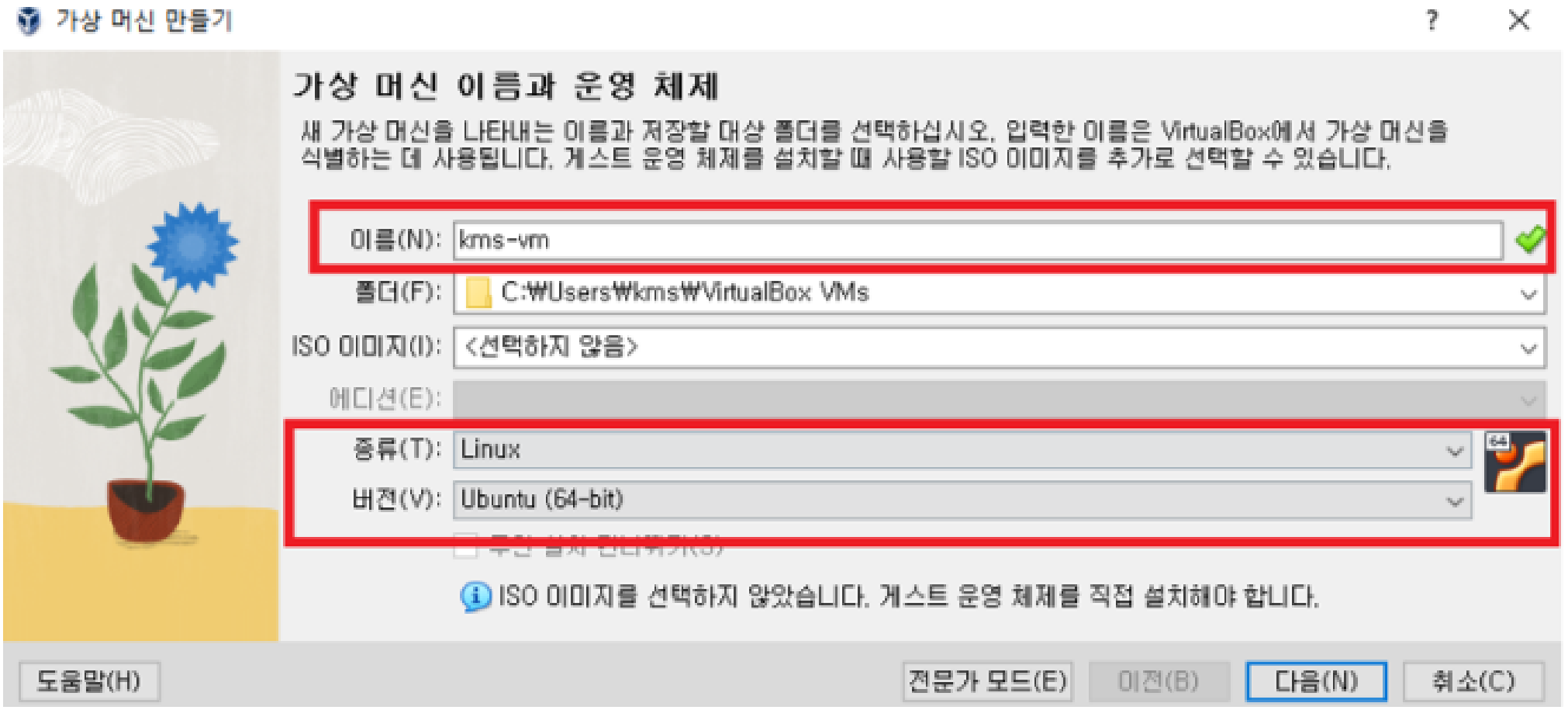
1. VM 설치(Virtual Box)

가상머신 새로 만들기



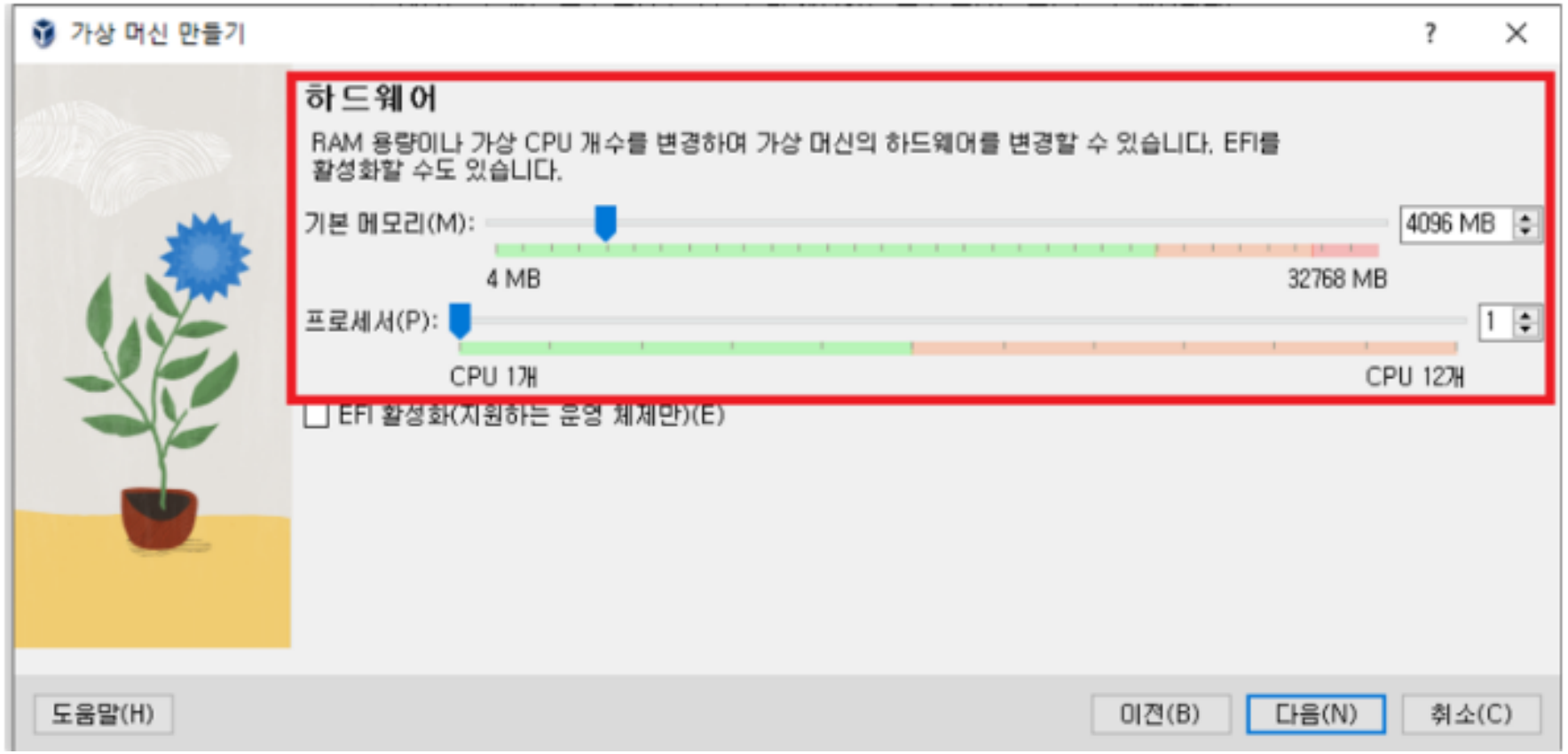
1. VM 설치(Virtual Box)

-이름, 종류(Linux), 버전(Ubuntu) 기입
-ISO 이미지는 <선택하지 않음>



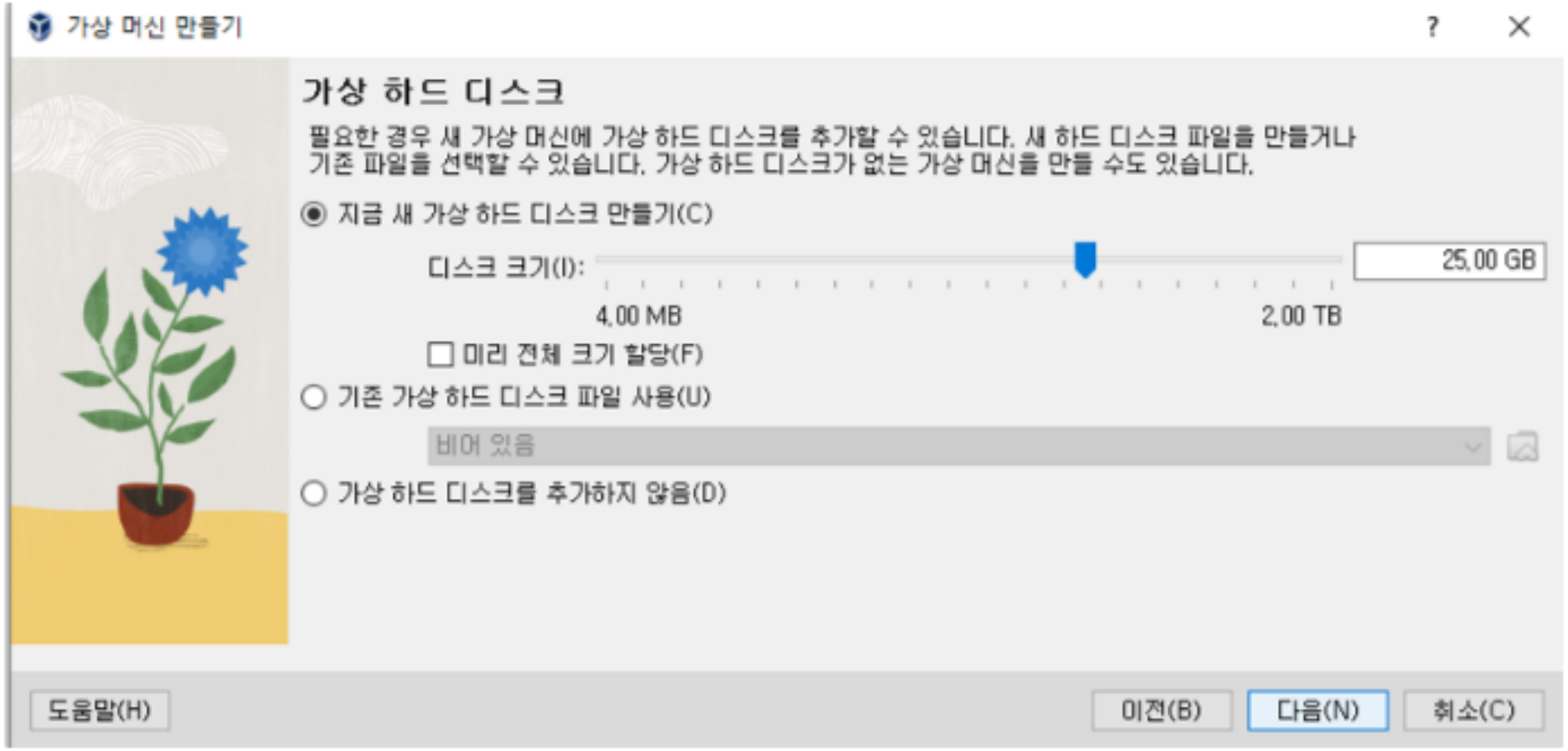
1. VM 설치(Virtual Box)

자신의 PC 사양에 맞게 적절히 설정



1. VM 설치(Virtual Box)

기본 설정으로 진행

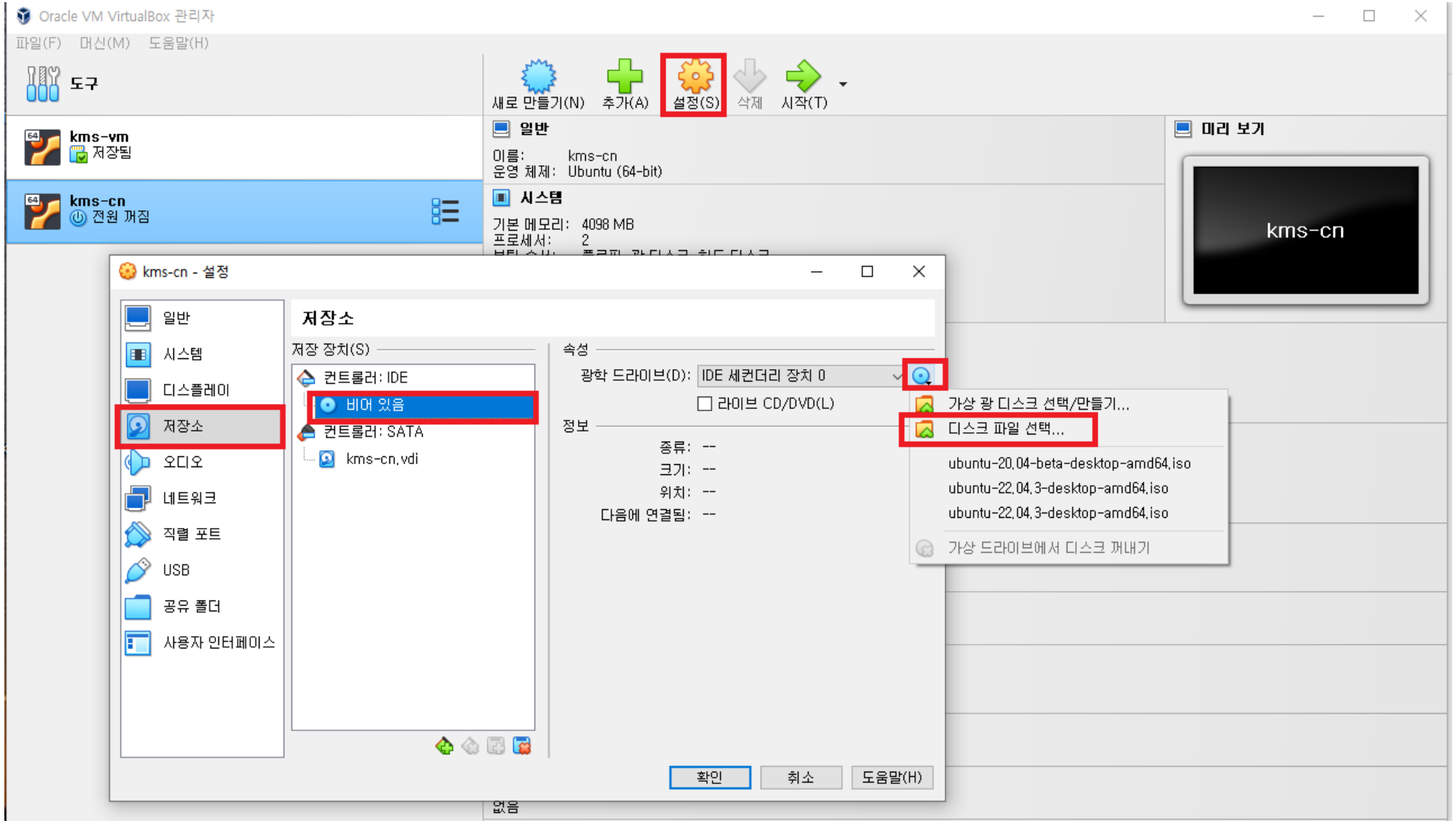


1. VM 설치(Virtual Box)

가상머신 설정

1. VM 설치(Virtual Box)

가상머신 설정- ISO 파일 설정 (다운받은 ISO 20.04 버전 파일 선택)

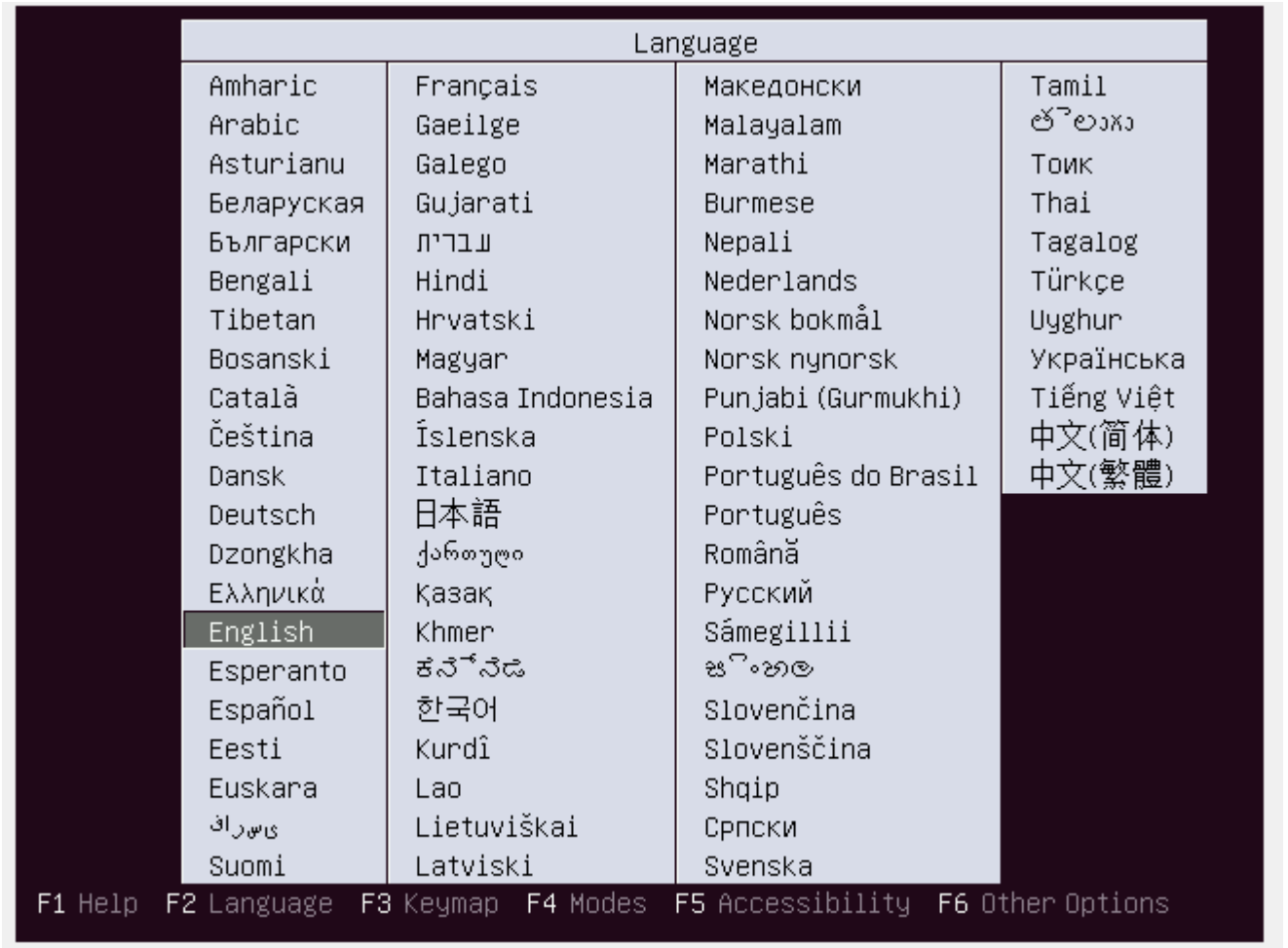


1. VM 설치(Virtual Box)

가상머신 시작

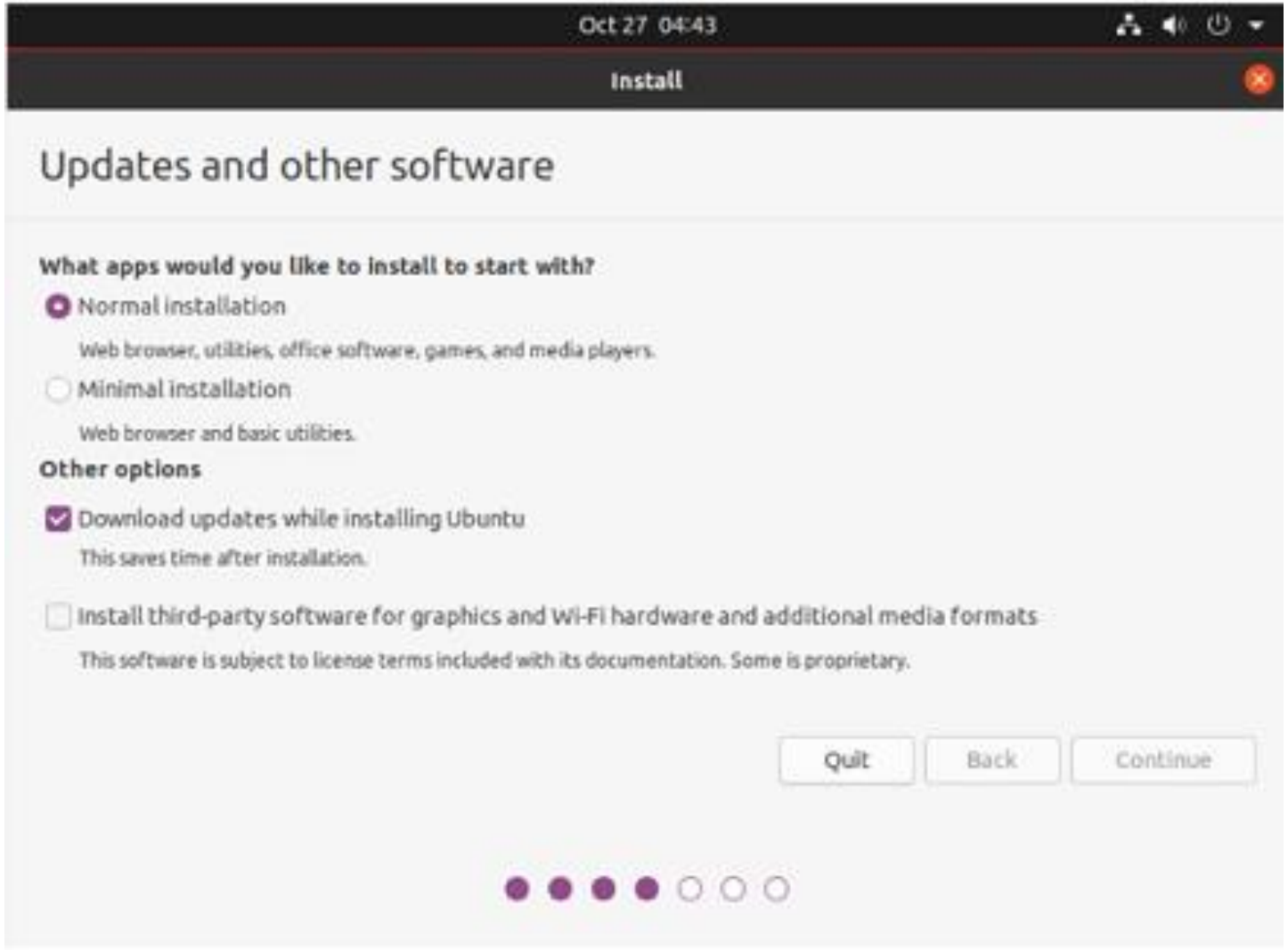
1. VM 설치(Virtual Box)

Ubuntu 설치
- English 로 설정하고 Enter 누른 후 메뉴에서 Install Ubuntu 선택



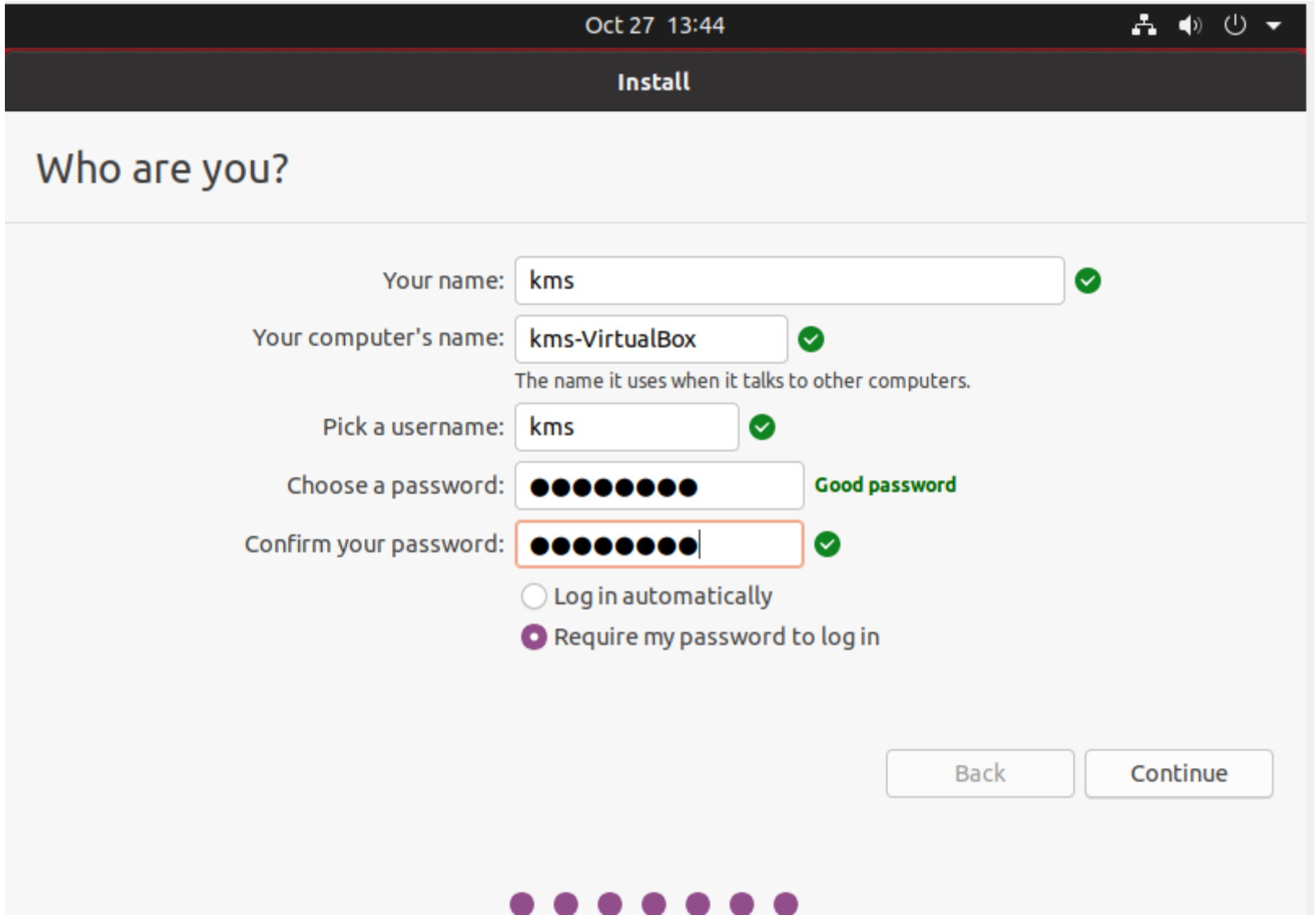
1. VM 설치(Virtual Box)

Ubuntu 설치
-별다른 설정 변경 없이 기본 설정으로 설치 진행(Next, Yes, Install, Continue)



1. VM 설치(Virtual Box)

Ubuntu 설치
-이름, 비밀번호 자유롭게 설정



1. VM 설치(Virtual Box)

Ubuntu 설치
-별다른 설정 변경 없이 기본 설정으로 설치 진행(Next, Yes, Install, Continue)

끝!

Linux 설정

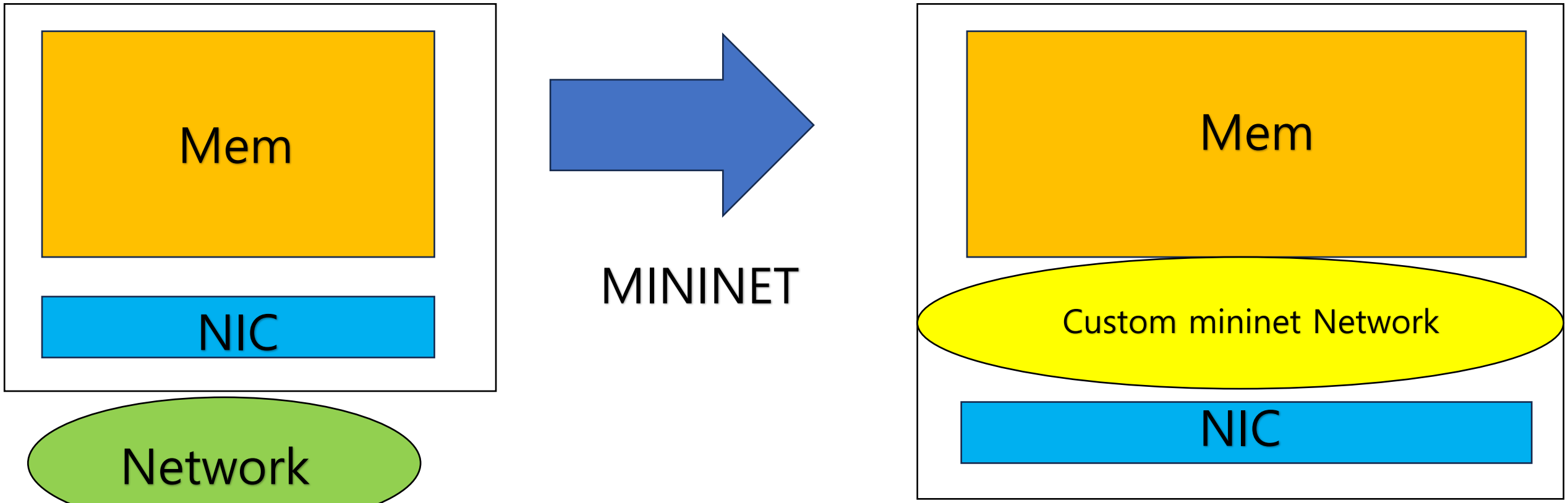
Linux 터미널 열고

$ sudo passwd <= root 비밀번호 설정

$ sudo apt update
$ sudo apt upgrade
$ sudo apt install build-essential
$ sudo apt install vim
$ sudo apt install git
$ sudo apt install python3
$ sudo apt install python-is-python3

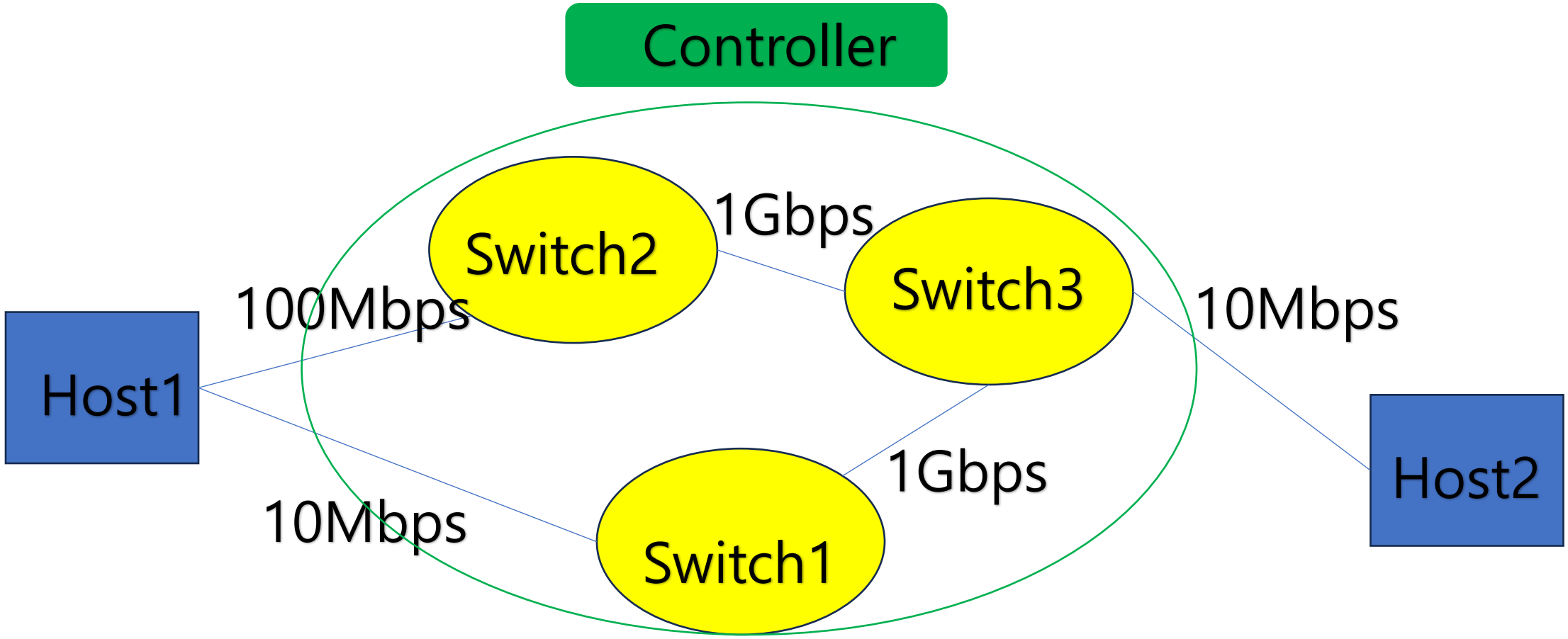
2. Mininet 소개

- Network Emulator
- 하나의 machine 내에서 가상 네트워크 환경 구성 가능



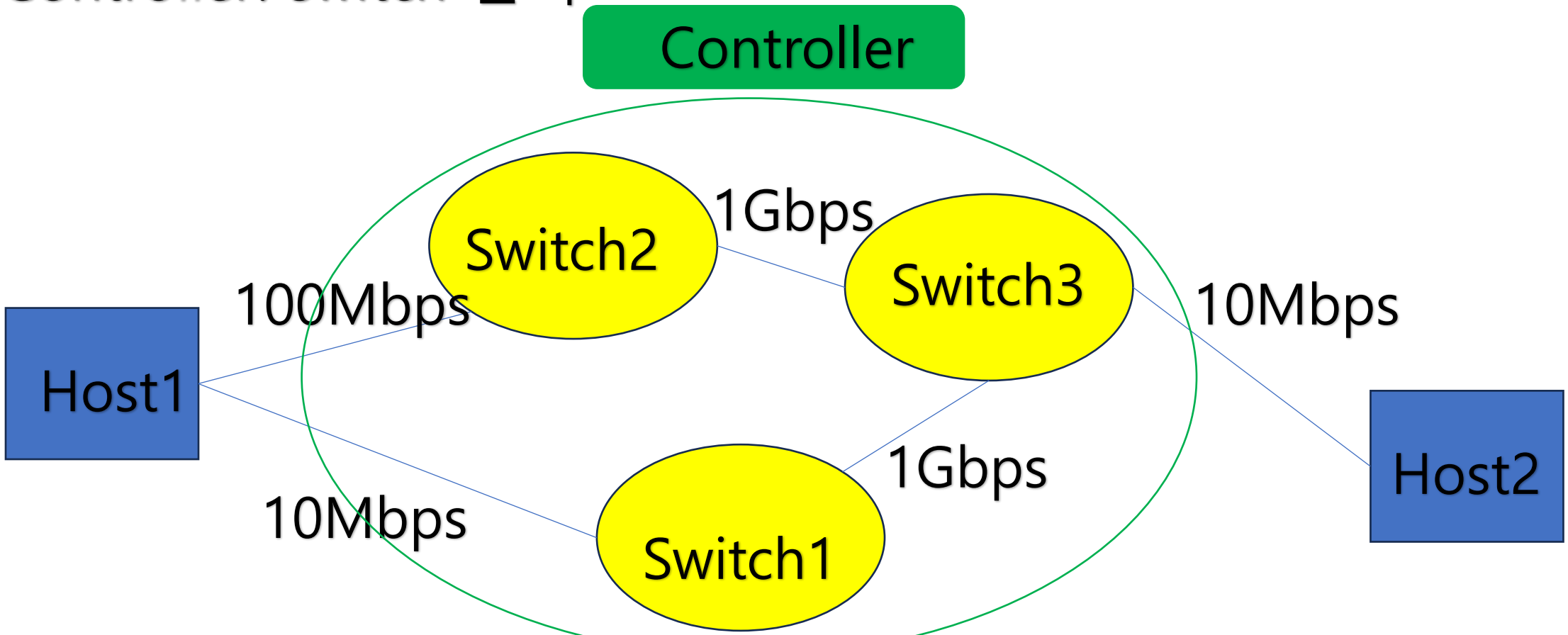
2. Mininet 소개

- Custom한 가상 network topology 구성 가능



2. Mininet 소개

- Host: 패킷 전송, 수신 및 application이 실행되는 Node
- Switch
- Controller: switch 관리



3. Mininet 설치

https://github.com/mininet/mininet/blob/master/INSTALL 의 3번 참고

$ git clone https://github.com/mininet/mininet.git

```
kms@kms-VirtualBox:~/cn$ git clone https://github.com/mininet/mininet.git
Cloning into 'mininet'...
remote: Enumerating objects: 10388, done.
remote: Counting objects: 100% (234/234), done.
remote: Compressing objects: 100% (141/141), done.
remote: Total 10388 (delta 129), reused 175 (delta 91), pack-reused 10154
Receiving objects: 100% (10388/10388), 3.36 MiB | 19.10 MiB/s, done.
Resolving deltas: 100% (6910/6910), done.
```

3. Mininet 설치

다운받은 파일 확인

```
kms@kms-VirtualBox:~/cn$ ls
mininet
kms@kms-VirtualBox:~/cn$ cd mininet/
kms@kms-VirtualBox:~/cn/mininet$ l
bin/          custom/  doc/       INSTALL  Makefile  mnexec.c   setup.py
CONTRIBUTORS  debian/  examples/  LICENSE  mininet/  README.md  util/
kms@kms-VirtualBox:~/cn/mininet$
```

Python 버전 확인(3.x 버전)

```
kms@kms-VirtualBox:~/cn/mininet$ python -V
Python 3.8.10
```

3. Mininet 설치

Mininet 설치

$ PYTHON=python3 util/install.sh -fnv

```
kms@kms-VirtualBox:~/cn/mininet$ PYTHON=python3 util/install.sh -fnv
Detected Linux distribution: Ubuntu 20.04 focal amd64
sys.version_info(major=3, minor=8, micro=10, releaselevel='final', serial=0)
Detected Python (python3) version 3
Installing OpenFlow reference implementation...
Reading package lists...
Building dependency tree...
Reading state information...
gcc is already the newest version (4:9.3.0-1ubuntu2).
gcc set to manually installed.
make is already the newest version (4.2.1-1.2).
make set to manually installed.
The following packages were automatically installed and are no longer required:
  gir1.2-goa-1.0 libfprint-2-tod1 libfwupdplugin1 libllvm9 libxmlb1
  ubuntu-system-service
```

3. Mininet 설치

Mininet 설치 확인

$ sudo mn

```
kms@kms-VirtualBox:~/cn/mininet$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

4. Mininet Demo1

아이캠퍼스에서 Example.py 파일 다운로드

```python
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.node import OVSKernelSwitch
from mininet.cli import CLI
from mininet.node import Host
from mininet.log import setLogLevel, info
import socket
import time


class SimpleTopology(Topo):
    def build(self):
        h1 = self.addHost('h1', cls=Host, defaultRoute=None)
        h2 = self.addHost('h2', cls=Host, defaultRoute=None)
        h3 = self.addHost('h3', cls=Host, defaultRoute=None)
        s1 = self.addSwitch( 's1', cls=OVSKernelSwitch, failMode='standalone')
        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(h3, s1)



def main():
    topo = SimpleTopology()
    net = Mininet( topo=topo, autoSetMacs=True, build=False, ipBase="10.0.0.0/24")

#    net.build()
    net.start()

    h1 = net.getNodeByName('h1')
    h2 = net.getNodeByName('h2')
    h3 = net.getNodeByName('h3')

    h1.setIP(intf="h1-eth0", ip="10.0.0.1/24")
    h2.setIP(intf="h2-eth0", ip="10.0.0.2/24")
    h3.setIP(intf="h3-eth0", ip="10.0.0.3/24")

    result = h1.cmd('ping -c 1', "10.0.0.2")

    # Measure the latency from the 'ping' command output
    lines = result.split('\n')
    print(lines[-2])




    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    main()
```

4. Mininet Demo1

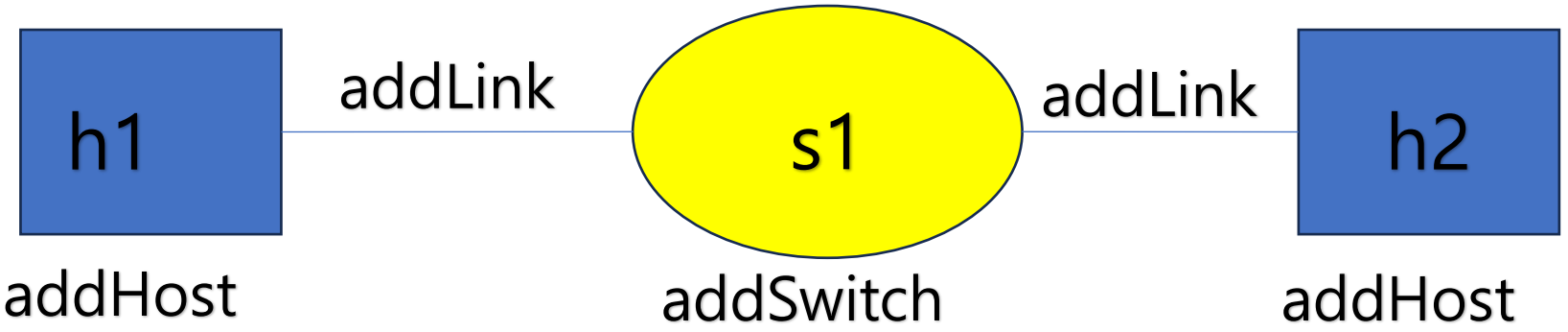
Example.py 실행
$ sudo python3 Example.py

```
kms@kms-VirtualBox:~/cn$ sudo python3 Example.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
rtt min/avg/max/mdev = 1.816/1.816/1.816/0.000 ms
*** Starting CLI:
mininet>
```

4. Mininet Demo1

1. 자신이 구성하고자 하는 network topology 정의

```
class SimpleTopology(Topo):
    def build(self):
        h1 = self.addHost('h1', cls=Host, defaultRoute=None)
        h2 = self.addHost('h2', cls=Host, defaultRoute=None)
        s1 = self.addSwitch( 's1', cls=OVSKernelSwitch, failMode='standalone')
        self.addLink(h1, s1)
        self.addLink(h2, s1)
```

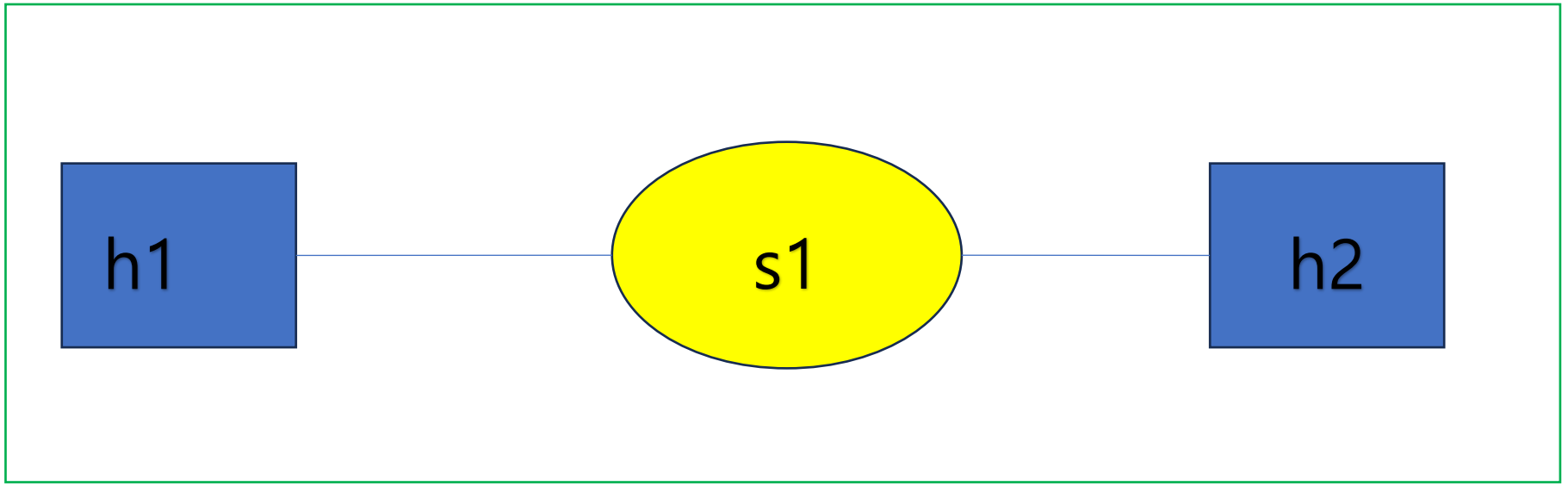


4. Mininet Demo1

2. Topology 선언 및 net 선언

```
def main():
    topo = SimpleTopology()
    net = Mininet( topo=topo, autoSetMacs=True, build=False, ipBase="10.0.0.0/24")
```

net



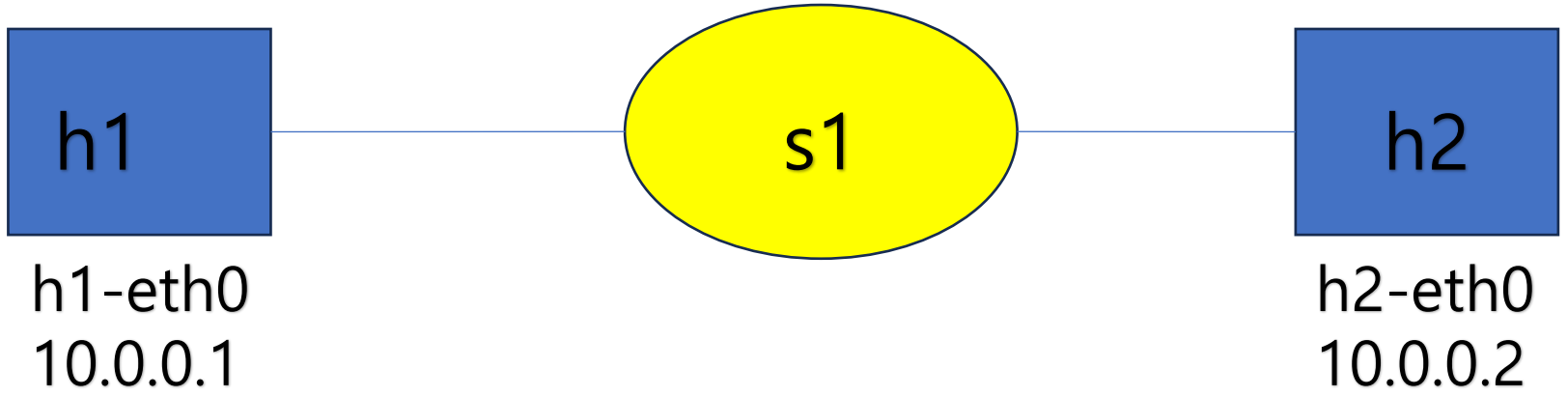
4. Mininet Demo1

3. 각 host의 ip, interface name 설정

```
    net.start()

    h1 = net.getNodeByName('h1')
    h2 = net.getNodeByName('h2')

    h1.setIP(intf="h1-eth0", ip="10.0.0.1/24")
    h2.setIP(intf="h2-eth0", ip="10.0.0.2/24")
```



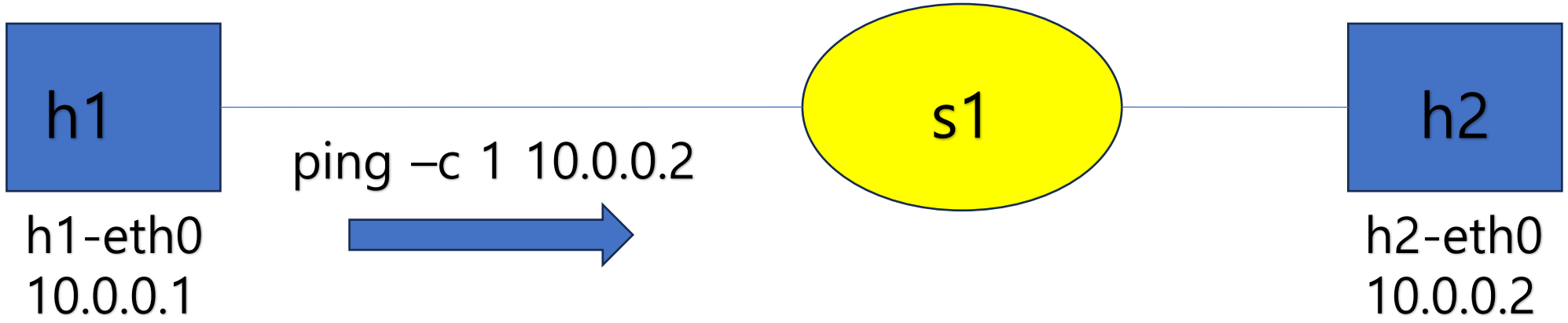
4. Mininet Demo1

4. Mininet상에서 명령어 수행

```
result = h1.cmd('ping -c 1', "10.0.0.2")

# Measure the latency from the 'ping' command output
lines = result.split('\n')
print("The result of ping is ", lines[-2])

CLI(net)
net.stop()
```



4. Mininet Demo1

5. 결과 확인

```
kms@kms-VirtualBox:~/cn$ sudo python3 Example.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
The result of ping is  rtt min/avg/max/mdev = 30.172/30.172/30.172/0.000 ms
*** Starting CLI:
mininet>
```

4. Mininet Demo2

1. 아이캠퍼스에서 udp_client.py, udp_server.py, udp_socket.py 파일 다운로드

```python
import socket
import sys

def udp_client(server_ip):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.sendto('Hello i am client'.encode('utf-8'), (server_ip, 12345))
    data, addr = s.recvfrom(1024)
    print("The message from server:", data.decode('utf-8'))
    s.close()

udp_client(sys.argv[1])
~
```

```python
import socket

def udp_server():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind(('10.0.0.2', 12345))
    data, addr = s.recvfrom(1024)
    s.sendto("Hi I'm server!".encode('utf-8'), addr)
    s.close()


udp_server()
~
~
```

```python
class SimpleTopology(Topo):
    def build(self):
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        s1 = self.addSwitch('s1')
        self.addLink(h1, s1)
        self.addLink(h2, s1)



def main():
    topo = SimpleTopology()
    net = Mininet( topo=topo)

    net.start()

    h1 = net.get('h1')
    h2 = net.get('h2')

    h2.cmd('python3 udp_server.py &')

    time.sleep(1)

    start_time = time.time()

    result = h1.cmd('python3 udp_client.py 10.0.0.2')


    end_time = time.time()


    rtt = (end_time - start_time) * 1000
    print("Round-trip time: {0:.2f} ms".format(rtt))
    print(result)




    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    main()
```

4. Mininet Demo2

2. 결과 확인

```
kms@kms-VirtualBox:~/cn$ sudo python3 udp_socket.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
Round-trip time: 24.68 ms
The message from server: Hi I'm server!

*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
```

4. Mininet Demo3

아이캠퍼스에서 Example_1.py 파일 다운로드

```python
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.node import OVSKernelSwitch
from mininet.cli import CLI
from mininet.node import Host
from mininet.log import setLogLevel, info
from mininet.link import Link, TCLink
import socket
import time


class SimpleTopology(Topo):
    def build(self):
        h1 = self.addHost('h1', cls=Host, defaultRoute=None)
        h2 = self.addHost('h2', cls=Host, defaultRoute=None)
        s1 = self.addSwitch( 's1', cls=OVSKernelSwitch, failMode='standalone')
        self.addLink(h1, s1, cls=TCLink, bw=10000000, delay='0.1ms', loss=0.001)

        self.addLink(h2, s1, cls=TCLink, bw=10000000, delay='0.1ms')
        #self.addLink(h2, s1)



def main():
    topo = SimpleTopology()
    net = Mininet( topo=topo, autoSetMacs=True, build=False, ipBase="10.0.0.0/24")

#    net.build()
    net.start()

    h1 = net.getNodeByName('h1')
    h2 = net.getNodeByName('h2')

    h1.setIP(intf="h1-eth0", ip="10.0.0.1/24")
    h2.setIP(intf="h2-eth0", ip="10.0.0.2/24")
    src, dst = net.hosts[0], net.hosts[1]
    s_bw, c_bw = net.iperf([src, dst], seconds=10)
    info(s_bw)
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    main()
```

4. Mininet Demo3

Link의 속성 설정(bandwidth, delay, loss 등)

```
self.addLink(h1, s1, cls=TCLink, bw=10000000, delay='0.1ms', loss=0.001)

self.addLink(h2, s1, cls=TCLink, bw=10000000, delay='0.1ms')
```

iperf 수행 후 결과 출력

```
h2.setIP(intf="h2-eth0", ip="10.0.0.2/24")
src, dst = net.hosts[0], net.hosts[1]
s_bw, c_bw = net.iperf([src, dst], seconds=10)
info(s_bw)
```

4. Mininet Demo3

결과확인

```
kms@kms-VirtualBox:~/cn$ sudo python3 Example_1.py
[sudo] password for kms:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
Bandwidth limit 10000000 is outside supported range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay 0.00100% loss) Bandwidth limit 10000000 is outside supp
orted range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay 0.00100% loss) (h1, s1) Bandwidth limit 10000000 is out
side supported range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay) Bandwidth limit 10000000 is outside supported range 0.
.1000 - ignoring
(10000000.00Mbit 0.1ms delay) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...Bandwidth limit 10000000 is outside supported range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay 0.00100% loss) Bandwidth limit 10000000 is outside supp
orted range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay)
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['11.1 Gbits/sec', '11.1 Gbits/sec']
11.1 Gbits/sec*** Starting CLI:
mininet>
```

5. Building Custom TCP Network Module

- Prerequisites:
  $ sudo apt update
  $ sudo apt install build-essential
  $ sudo apt install linux-headers-$(uname-r)
  $ sudo apt install linux-source

*** When downloaded into system directory, files are stored in /usr/src/linux-source-5.4.0/ (for example).

```
nehtw@ubuntu:~$ cd /usr/src/linux-source-5.4.0/
nehtw@ubuntu:/usr/src/linux-source-5.4.0$ cd net/ipv4/
nehtw@ubuntu:/usr/src/linux-source-5.4.0/net/ipv4$ ls
af_inet.c        fou.c                  ip_gre.c          netfilter.c        tcp_cdg.c        tcp_minisocks.c  udp.c
ah4.c            gre_demux.c            ip_input.c        netlink.c          tcp_cong.c       tcp_nv.c         udp_diag.c
arp.c            gre_offload.c          ipip.c            nexthop.c          tcp_cubic.c      tcp_offload.c    udp_impl.h
bpfilter         icmp.c                 ipmr_base.c       ping.c             tcp_dctcp.c      tcp_output.c     udplite.c
cipso_ipv4.c     igmp.c                 ipmr.c            proc.c             tcp_dctcp.h      tcp_rate.c       udp_offload.c
datagram.c       inet_connection_sock.c ip_options.c      protocol.c         tcp_diag.c       tcp_recovery.c   udp_tunnel.c
devinet.c        inet_diag.c            ip_output.c       raw.c              tcp_fastopen.c   tcp_scalable.c   xfrm4_input.c
esp4.c           inet_fragment.c        ip_sockglue.c     raw_diag.c         tcp_highspeed.c  tcp_timer.c      xfrm4_output.c
esp4_offload.c   inet_hashtables.c      ip_tunnel.c       route.c            tcp_htcp.c       tcp_ulp.c        xfrm4_policy.c
fib_frontend.c   inetpeer.c             ip_tunnel_core.c  syncookies.c       tcp_hybla.c      tcp_vegas.c      xfrm4_protocol.c
fib_lookup.h     inet_timewait_sock.c   ip_vti.c          sysctl_net_ipv4.c  tcp_illinois.c   tcp_vegas.h      xfrm4_state.c
fib_notifier.c   ipcomp.c               Kconfig           tcp_bbr.c          tcp_input.c      tcp_veno.c       xfrm4_tunnel.c
fib_rules.c      ipconfig.c             Makefile          tcp_bic.c          tcp_ipv4.c       tcp_westwood.c
fib_semantics.c  ip_forward.c           metrics.c         tcp_bpf.c          tcp_lp.c         tcp_yeah.c
fib_trie.c       ip_fragment.c          netfilter         tcp.c              tcp_metrics.c    tunnel4.c
```

5. Building Custom TCP Network Module

- net/ipv4/tcp_cong.c : TCP Reno congestion algorithm:

```
/*
 * TCP Reno congestion control
 * This is special case used for fallback as well.
 */
/* This is Jacobson's slow start and congestion avoidance.
 * SIGCOMM '88, p. 328.
 */
void tcp_reno_cong_avoid(struct sock *sk, u32 ack, u32 acked)
{
        struct tcp_sock *tp = tcp_sk(sk);

        if (!tcp_is_cwnd_limited(sk))
                return;

        /* In "safe" area, increase. */
        if (tcp_in_slow_start(tp)) {
                acked = tcp_slow_start(tp, acked);
                if (!acked)
                        return;
        }
        /* In dangerous area, increase slowly. */
        tcp_cong_avoid_ai(tp, tp->snd_cwnd, acked);
}
EXPORT_SYMBOL_GPL(tcp_reno_cong_avoid);

/* Slow start threshold is half the congestion window (min 2) */
u32 tcp_reno_ssthresh(struct sock *sk)
{
        const struct tcp_sock *tp = tcp_sk(sk);

        return max(tp->snd_cwnd >> 1U, 2U);
}
EXPORT_SYMBOL_GPL(tcp_reno_ssthresh);
```

5. Building Custom TCP Network Module

- Preparing the Module: (**using uploaded file** in icampus)

```
$ mkdir tcp_simple_custom
$ cd tcp_simple_custom
$ vim reno_custom.c
```

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <net/tcp.h>

void tcp_reno_init(struct sock *sk)
{
    /* Initialize congestion control specific variables here */
    tcp_sk(sk)->snd_ssthresh = TCP_INFINITE_SSTHRESH; // Typically, this is a high value
    tcp_sk(sk)->snd_cwnd = 1; // Start with a congestion window of 1
}

u32 tcp_reno_ssthresh(struct sock *sk)
{
    /* Halve the congestion window, min 2 */
    const struct tcp_sock *tp = tcp_sk(sk);
    return max(tp->snd_cwnd >> 1U, 2U);
}

void tcp_reno_cong_avoid(struct sock *sk, u32 ack, u32 acked)
{
    struct tcp_sock *tp = tcp_sk(sk);
    printk(KERN_INFO "tp->snd_cwnd is %d\n", tp->snd_cwnd);

    if (!tcp_is_cwnd_limited(sk))
        return;

    if (tp->snd_cwnd <= tp->snd_ssthresh) {
        /* In "slow start", cwnd is increased by the number of ACKed packets */
        acked = tcp_slow_start(tp, acked);
        if (!acked)
            return;
    } else {
        /* In "congestion avoidance", cwnd is increased by 1 full packet
         * per round-trip time (RTT), which is approximated here by the number of
         * ACKed packets divided by the current congestion window. */
        tcp_cong_avoid_ai(tp, tp->snd_cwnd, acked);
    }

    /* Ensure that cwnd does not exceed the maximum allowed value */
    tp->snd_cwnd = min(tp->snd_cwnd, tp->snd_cwnd_clamp);
}
```

For Debug Message,
refer to "dmesg 명령어 옵션"
6.참고자료 54p

5. Building Custom TCP Network Module

- Creating the Makefile:

```
obj-m += reno_custom.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

obj-m += reno_custom.o

all:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

```
nehtw@ubuntu:~/tcp_simple_custom$ make
make -C /lib/modules/5.15.0-88-generic/build M=/home/nehtw/tcp_simple_custom modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-88-generic'
  CC [M]  /home/nehtw/tcp_simple_custom/reno_custom.o
  MODPOST /home/nehtw/tcp_simple_custom/Module.symvers
  CC [M]  /home/nehtw/tcp_simple_custom/reno_custom.mod.o
  LD [M]  /home/nehtw/tcp_simple_custom/reno_custom.ko
  BTF [M] /home/nehtw/tcp_simple_custom/reno_custom.ko
Skipping BTF generation for /home/nehtw/tcp_simple_custom/reno_custom.ko due to unavailability of vmlinux
```

5. Building Custom TCP Network Module

- Inserting/Loading the Module
 $ sudo insmod reno_custom.ko
 $ lsmod | grep reno_custom

- Checking the Current Congestion Control Algorithms:
 $ cat /proc/sys/net/ipv4/tcp_congestion_control
- Checking the Available Congestion Control Algorithms:
 $ cat /proc/sys/net/ipv4/tcp_available_congestion_control

```
nehtw@ubuntu:~/tcp_simple_custom$ sudo insmod reno_custom.ko
nehtw@ubuntu:~/tcp_simple_custom$ lsmod | grep reno_custom
reno_custom            16384  0
nehtw@ubuntu:~/tcp_simple_custom$ cat /proc/sys/net/ipv4/tcp_congestion_control
cubic
nehtw@ubuntu:~/tcp_simple_custom$ cat /proc/sys/net/ipv4/tcp_available_congestion_control
reno cubic reno_custom
```

5. Building Custom TCP Network Module

- Setting the Congestion Control Algorithms:
  $ sudo sysctl net.ipv4.tcp_congestion_control=reno_custom

```
nehtw@ubuntu:~/tcp_simple_custom$ sudo sysctl net.ipv4.tcp_congestion_control=reno_custom
net.ipv4.tcp_congestion_control = reno_custom
```

- For the purposes of testing, to artificially introduce congestion,
  1) Use Iperf3 tool.
     open two terminal,
      terminal 1) $ iperf3 -s
      terminal 2) $ iperf3 –c 127.0.0.1

  - Monitor with dmesg
    $ dmesg

5. Building Custom TCP Network Module

- result

```
nehtw@ubuntu:~/tcp_simple_custom$ iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 127.0.0.1, port 54264
[  5] local 127.0.0.1 port 5201 connected to 127.0.0.1 port 54270
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  3.66 GBytes  31.5 Gbits/sec
[  5]   1.00-2.00   sec  4.22 GBytes  36.3 Gbits/sec
[  5]   2.00-3.00   sec  4.43 GBytes  38.0 Gbits/sec
[  5]   3.00-4.00   sec  4.48 GBytes  38.4 Gbits/sec
[  5]   4.00-5.00   sec  4.11 GBytes  35.3 Gbits/sec
[  5]   5.00-6.00   sec  4.81 GBytes  41.3 Gbits/sec
[  5]   6.00-7.00   sec  4.81 GBytes  41.3 Gbits/sec
[  5]   7.00-8.00   sec  4.74 GBytes  40.7 Gbits/sec
[  5]   8.00-9.00   sec  4.55 GBytes  39.1 Gbits/sec
[  5]   9.00-10.00  sec  5.69 GBytes  48.9 Gbits/sec
[  5]  10.00-10.04  sec   221 MBytes  42.8 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.04  sec  45.7 GBytes  39.1 Gbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

```
nehtw@ubuntu:~/tcp_simple_custom$ iperf3 -c 127.0.0.1
Connecting to host 127.0.0.1, port 5201
[  5] local 127.0.0.1 port 54270 connected to 127.0.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  3.85 GBytes  33.1 Gbits/sec    0   1.21 GBytes

[  5]   1.00-2.00   sec  4.22 GBytes  36.3 Gbits/sec    0   -1071441273.00 By
tes
[  5]   2.00-3.00   sec  4.41 GBytes  37.9 Gbits/sec    0   1.51 GBytes

[  5]   3.00-4.00   sec  4.53 GBytes  38.9 Gbits/sec    0   15.5 MBytes

[  5]   4.00-5.00   sec  4.07 GBytes  35.0 Gbits/sec    0   1.65 GBytes

[  5]   5.00-6.00   sec  4.82 GBytes  41.4 Gbits/sec    0    645 MBytes

[  5]   6.00-7.00   sec  4.82 GBytes  41.4 Gbits/sec    0   -391815688.00 Byt
es
[  5]   7.00-8.00   sec  4.70 GBytes  40.3 Gbits/sec    0   -1733109595.00 By
tes
[  5]   8.00-9.00   sec  4.55 GBytes  39.1 Gbits/sec    0    870 MBytes

[  5]   9.00-10.00  sec  5.74 GBytes  49.3 Gbits/sec    0   -347759674.00 Byt
es
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  45.7 GBytes  39.3 Gbits/sec    0             sender
[  5]   0.00-10.04  sec  45.7 GBytes  39.1 Gbits/sec                  receive
r

iperf Done.
```

5. Building Custom TCP Network Module

- For the purposes of testing, to artificially introduce congestion,
  2) Use **Mininet demo3 in page 42**.
  $ sudo python3 Example_1.py

  - Monitor with dmesg
  $ dmesg

5. Building Custom TCP Network Module

- after packet send, execute “$ dmesg”

```
kms@kms-VirtualBox:~/cn$ vim Example_1.py
kms@kms-VirtualBox:~/cn$ sudo python3 Example_1.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
Bandwidth limit 10000000 is outside supported range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay 0.00100% loss) Bandwidth limit 10000000 is outside supp
orted range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay 0.00100% loss) (h1, s1) Bandwidth limit 10000000 is out
side supported range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay) Bandwidth limit 10000000 is outside supported range 0.
.1000 - ignoring
(10000000.00Mbit 0.1ms delay) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...Bandwidth limit 10000000 is outside supported range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay 0.00100% loss) Bandwidth limit 10000000 is outside supp
orted range 0..1000 - ignoring
(10000000.00Mbit 0.1ms delay)
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['14.2 Gbits/sec', '14.1 Gbits/sec']
14.2 Gbits/sec*** Starting CLI:
mininet>
```

```
[91387.065506] tp->snd_cwnd is 7074
[91387.072033] tp->snd_cwnd is 7074
[91387.072257] tp->snd_cwnd is 7074
[91387.076027] tp->snd_cwnd is 7074
[91387.078864] tp->snd_cwnd is 7074
[91387.079227] tp->snd_cwnd is 7074
[91387.080673] tp->snd_cwnd is 7074
[91387.080683] tp->snd_cwnd is 7074
[91387.085272] tp->snd_cwnd is 7074
[91387.085960] tp->snd_cwnd is 7074
[91387.085964] tp->snd_cwnd is 7074
[91387.085965] tp->snd_cwnd is 7074
[91387.089594] tp->snd_cwnd is 3537
[91387.091071] tp->snd_cwnd is 3537
[91387.091246] tp->snd_cwnd is 3537
[91387.091289] tp->snd_cwnd is 3537
[91387.091294] tp->snd_cwnd is 3537
[91387.091297] tp->snd_cwnd is 3537
[91387.091299] tp->snd_cwnd is 3537
[91387.091301] tp->snd_cwnd is 3537
[91387.091304] tp->snd_cwnd is 3537
[91387.091306] tp->snd_cwnd is 3537
[91387.091308] tp->snd_cwnd is 3537
[91387.091311] tp->snd_cwnd is 3537
[91387.091313] tp->snd_cwnd is 3537
[91387.091424] tp->snd_cwnd is 3537
[91387.091499] tp->snd_cwnd is 3537
[91387.137986] tp->snd_cwnd is 3537
[91387.138251] tp->snd_cwnd is 3537
kms@kms-VirtualBox:~/cn/module_ex$
```

6. 참고자료

Mininet 사용 example
- https://github.com/mininet/mininet/tree/master/examples

Mininet 공식 doc
- http://mininet.org/

dmesg 명령어 옵션
- https://jjeongil.tistory.com/1786

TCP's Congestion Control Implementation in Linux Kernel
- https://wiki.aalto.fi/download/attachments/69901948/TCP-CongestionControl.pdf

Appendix – TCP Reno code analysis

* Header and includes
  - standard C preprocessor directives and include statements.

  - define a format for logging messages prefixed with "TCP: ",

  - and include various headers for functionalities needed by the TCP code (like memory management, data types, and TCP specific functions).

```
#define pr_fmt(fmt) "TCP: " fmt

#include <linux/module.h>
#include <linux/mm.h>
#include <linux/types.h>
#include <linux/list.h>
#include <linux/gfp.h>
#include <linux/jhash.h>
#include <net/tcp.h>
```

Appendix – TCP Reno code analysis

* Global variables
  - define a spinlock and initialize a list head to manage TCP congestion control algorithms safely in a multi-threaded context.

  - This list is protected by a spinlock, ensuring that concurrent accesses do not cause race conditions.

```
static DEFINE_SPINLOCK(tcp_cong_list_lock);
static LIST_HEAD(tcp_cong_list);
```

Appendix – TCP Reno code analysis

* Finding a congestion control algorithm
  - searches for a TCP congestion control algorithm by its name or key in the global list of registered algorithms.

  - returns a pointer to the tcp_congestion_ops structure if found or NULL if not.

```c
/* Simple linear search, don't expect many entries! */
static struct tcp_congestion_ops *tcp_ca_find(const char *name)
{
	struct tcp_congestion_ops *e;

	list_for_each_entry_rcu(e, &tcp_cong_list, list) {
		if (strcmp(e->name, name) == 0)
			return e;
	}

	return NULL;
}
```

```c
/* Simple linear search, not much in here. */
struct tcp_congestion_ops *tcp_ca_find_key(u32 key)
{
	struct tcp_congestion_ops *e;

	list_for_each_entry_rcu(e, &tcp_cong_list, list) {
		if (e->key == key)
			return e;
	}

	return NULL;
}
```

Appendix – TCP Reno code analysis

* tcp_register_congestion_control
  - registers a new TCP congestion control algorithm, ensuring that it implements the required operations.
  - If the algorithm is already registered or has a non-unique key, it returns an error.

```c
/*
 * Attach new congestion control algorithm to the list
 * of available options.
 */
int tcp_register_congestion_control(struct tcp_congestion_ops *ca)
{
        int ret = 0;

        /* all algorithms must implement these */
        if (!ca->ssthresh || !ca->undo_cwnd ||
            !(ca->cong_avoid || ca->cong_control)) {
                pr_err("%s does not implement required ops\n", ca->name);
                return -EINVAL;
        }

        ca->key = jhash(ca->name, sizeof(ca->name), strlen(ca->name));

        spin_lock(&tcp_cong_list_lock);
        if (ca->key == TCP_CA_UNSPEC || tcp_ca_find_key(ca->key)) {
                pr_notice("%s already registered or non-unique key\n",
                          ca->name);
                ret = -EEXIST;
        } else {
                list_add_tail_rcu(&ca->list, &tcp_cong_list);
                pr_debug("%s registered\n", ca->name);
        }
        spin_unlock(&tcp_cong_list_lock);

        return ret;
}
```

Appendix – TCP Reno code analysis

* tcp_unregister_congestion_control
  - unregisters a TCP congestion control algorithm

  - makes sure that any ongoing operations using this algorithm complete by calling synchronize_rcu() after removal.

```
void tcp_unregister_congestion_control(struct tcp_congestion_ops *ca)
{
        spin_lock(&tcp_cong_list_lock);
        list_del_rcu(&ca->list);
        spin_unlock(&tcp_cong_list_lock);

        /* Wait for outstanding readers to complete before the
         * module gets removed entirely.
         *
         * A try_module_get() should fail by now as our module is
         * in "going" state since no refs are held anymore and
         * module_exit() handler being called.
         */
        synchronize_rcu();
}
EXPORT_SYMBOL_GPL(tcp_unregister_congestion_control);
```

Appendix – TCP Reno code analysis

* tcp_ca_get_key_by_name, _by_key
  - are used to retrieve the key of a TCP congestion control algorithm by its name and vice versa.

```
u32 tcp_ca_get_key_by_name(struct net *net, const char *name, bool *ecn_ca)
{
        const struct tcp_congestion_ops *ca;
        u32 key = TCP_CA_UNSPEC;

        might_sleep();

        rcu_read_lock();
        ca = tcp_ca_find_autoload(net, name);
        if (ca) {
                key = ca->key;
                *ecn_ca = ca->flags & TCP_CONG_NEEDS_ECN;
        }
        rcu_read_unlock();

        return key;
}
EXPORT_SYMBOL_GPL(tcp_ca_get_key_by_name);
```

```
char *tcp_ca_get_name_by_key(u32 key, char *buffer)
{
        const struct tcp_congestion_ops *ca;
        char *ret = NULL;

        rcu_read_lock();
        ca = tcp_ca_find_key(key);
        if (ca)
                ret = strncpy(buffer, ca->name,
                              TCP_CA_NAME_MAX);
        rcu_read_unlock();

        return ret;
}
EXPORT_SYMBOL_GPL(tcp_ca_get_name_by_key);
```

Appendix – TCP Reno code analysis

* tcp_assign_congestion_control, _init_
  - set up the congestion control algorithm for a new TCP socket.

  - If the chosen algorithm requires Explicit Congestion Notification (ECN), they configure the socket accordingly.

```
/* Assign choice of congestion control. */
void tcp_assign_congestion_control(struct sock *sk)
{
        struct net *net = sock_net(sk);
        struct inet_connection_sock *icsk = inet_csk(sk);
        const struct tcp_congestion_ops *ca;

        rcu_read_lock();
        ca = rcu_dereference(net->ipv4.tcp_congestion_control);
        if (unlikely(!try_module_get(ca->owner)))
                ca = &tcp_reno;
        icsk->icsk_ca_ops = ca;
        rcu_read_unlock();

        memset(icsk->icsk_ca_priv, 0, sizeof(icsk->icsk_ca_priv));
        if (ca->flags & TCP_CONG_NEEDS_ECN)
                INET_ECN_xmit(sk);
        else
                INET_ECN_dontxmit(sk);
}

void tcp_init_congestion_control(struct sock *sk)
{
        const struct inet_connection_sock *icsk = inet_csk(sk);

        tcp_sk(sk)->prior_ssthresh = 0;
        if (icsk->icsk_ca_ops->init)
                icsk->icsk_ca_ops->init(sk);
        if (tcp_ca_needs_ecn(sk))
                INET_ECN_xmit(sk);
        else
                INET_ECN_dontxmit(sk);
}
```

Appendix – TCP Reno code analysis

```
static void tcp_reinit_congestion_control(struct sock *sk,
                                          const struct tcp_congestion_ops *ca)
{
        struct inet_connection_sock *icsk = inet_csk(sk);

        tcp_cleanup_congestion_control(sk);
        icsk->icsk_ca_ops = ca;
        icsk->icsk_ca_setsockopt = 1;
        memset(icsk->icsk_ca_priv, 0, sizeof(icsk->icsk_ca_priv));

        if (ca->flags & TCP_CONG_NEEDS_ECN)
                INET_ECN_xmit(sk);
        else
                INET_ECN_dontxmit(sk);

        if (!((1 << sk->sk_state) & (TCPF_CLOSE | TCPF_LISTEN)))
                tcp_init_congestion_control(sk);
}
```

* tcp_reinit_congestion_control
  - re-initializes the congestion control for a socket if a different algorithm is chosen after the socket is already in use.

Appendix – TCP Reno code analysis

```
/* Manage refcounts on socket close. */
void tcp_cleanup_congestion_control(struct sock *sk)
{
        struct inet_connection_sock *icsk = inet_csk(sk);

        if (icsk->icsk_ca_ops->release)
                icsk->icsk_ca_ops->release(sk);
        module_put(icsk->icsk_ca_ops->owner);
}
```

* tcp_cleanup_congestion_control
  - cleans up any private data when a socket no longer uses a particular congestion control algorithm.

Appendix – TCP Reno code analysis

```
/* Used by sysctl to change default congestion control */
int tcp_set_default_congestion_control(struct net *net, const char *name)
{
```

```
/* Build string with list of available congestion control values */
void tcp_get_available_congestion_control(char *buf, size_t maxlen)
```

```
void tcp_get_default_congestion_control(struct net *net, char *name)
{
```

* related to sysctl operations
  -> tcp_set_default_congestion_control,
  tcp_get_available_congestion_control,
  tcp_get_default_congestion_control, etc..

  - interact with sysctl to allow user-space tools to query or set TCP congestion control algorithms.

Appendix – TCP Reno code analysis

* tcp_set_congestion_control
  - changes the congestion control algorithm for a given socket, handling module reference counts and permissions appropriately.

```c
/* Change congestion control for socket. If load is false, then it is the
 * responsibility of the caller to call tcp_init_congestion_control or
 * tcp_reinit_congestion_control (if the current congestion control was
 * already initialized.
 */
int tcp_set_congestion_control(struct sock *sk, const char *name, bool load,
                               bool reinit, bool cap_net_admin)
{
        struct inet_connection_sock *icsk = inet_csk(sk);
        const struct tcp_congestion_ops *ca;
        int err = 0;

        if (icsk->icsk_ca_dst_locked)
                return -EPERM;

        rcu_read_lock();
        if (!load)
                ca = tcp_ca_find(name);
        else
                ca = tcp_ca_find_autoload(sock_net(sk), name);

        /* No change asking for existing value */
        if (ca == icsk->icsk_ca_ops) {
                icsk->icsk_ca_setsockopt = 1;
                goto out;
        }

        if (!ca) {
                err = -ENOENT;
        } else if (!load) {
                const struct tcp_congestion_ops *old_ca = icsk->icsk_ca_ops;

                if (try_module_get(ca->owner)) {
                        if (reinit) {
                                tcp_reinit_congestion_control(sk, ca);
                        } else {
                                icsk->icsk_ca_ops = ca;
                                module_put(old_ca->owner);
                        }
                } else {
                        err = -EBUSY;
                }
        } else if (!((ca->flags & TCP_CONG_NON_RESTRICTED) || cap_net_admin)) {
                err = -EPERM;
        } else if (!try_module_get(ca->owner)) {
                err = -EBUSY;
        } else {
                tcp_reinit_congestion_control(sk, ca);
        }
 out:
        rcu_read_unlock();
        return err;
}
```

Appendix – TCP Reno code analysis

```
u32 tcp_slow_start(struct tcp_sock *tp, u32 acked)
{
        u32 cwnd = min(tp->snd_cwnd + acked, tp->snd_ssthresh);

        acked -= cwnd - tp->snd_cwnd;
        tp->snd_cwnd = min(cwnd, tp->snd_cwnd_clamp);

        return acked;
}
```

* tcp_slow_start
  - implements the TCP slow start algorithm, which increases the congestion window (cwnd) rapidly to find the network capacity.

Appendix – TCP Reno code analysis

* tcp_cong_avoid_ai
  - "tcp congestion avoidance increment," is part of the TCP congestion control algorithm.

  - adjusts the cwnd carefully to ensure that the connection probes for additional bandwidth without inducing congestion.

```
void tcp_cong_avoid_ai(struct tcp_sock *tp, u32 w, u32 acked)
{
        /* If credits accumulated at a higher w, apply them gently now. */
        if (tp->snd_cwnd_cnt >= w) {
                tp->snd_cwnd_cnt = 0;
                tp->snd_cwnd++;
        }

        tp->snd_cwnd_cnt += acked;
        if (tp->snd_cwnd_cnt >= w) {
                u32 delta = tp->snd_cwnd_cnt / w;

                tp->snd_cwnd_cnt -= delta * w;
                tp->snd_cwnd += delta;
        }
        tp->snd_cwnd = min(tp->snd_cwnd, tp->snd_cwnd_clamp);
}
EXPORT_SYMBOL_GPL(tcp_cong_avoid_ai);
```

Appendix – TCP Reno code analysis

```
void tcp_reno_cong_avoid(struct sock *sk, u32 ack, u32 acked)
{
        struct tcp_sock *tp = tcp_sk(sk);

        if (!tcp_is_cwnd_limited(sk))
                return;

        /* In "safe" area, increase. */
        if (tcp_in_slow_start(tp)) {
                acked = tcp_slow_start(tp, acked);
                if (!acked)
                        return;
        }
        /* In dangerous area, increase slowly. */
        tcp_cong_avoid_ai(tp, tp->snd_cwnd, acked);
}
EXPORT_SYMBOL_GPL(tcp_reno_cong_avoid);
```

* tcp_reno_cong_avoid:
  - invoked to perform a conservative increase of the congestion window during the congestion avoidance phase.

  - If a packet loss is detected through triple duplicate acknowledgments, TCP Reno will reduce the cwnd by half, which is the multiplicative decrease aspect of AIMD.

  - This function helps to strike a balance between throughput and network stability.

Appendix – TCP Reno code analysis

```
u32 tcp_reno_ssthresh(struct sock *sk)
{
        const struct tcp_sock *tp = tcp_sk(sk);

        return max(tp->snd_cwnd >> 1U, 2U);
}
EXPORT_SYMBOL_GPL(tcp_reno_ssthresh);
```

* tcp_reno_ssthresh:
  - refers to the function or routine that determines the slow start threshold (ssthresh) in the Reno version of TCP.

  - When congestion is detected (typically by a timeout or the receipt of duplicate ACKs), Reno reduces the congestion window size, and ssthresh is adjusted to be halfway between the current cwnd and the flow's maximum window size observed so far.

Appendix – TCP Reno code analysis

```
u32 tcp_reno_undo_cwnd(struct sock *sk)
{
        const struct tcp_sock *tp = tcp_sk(sk);

        return max(tp->snd_cwnd, tp->prior_cwnd);
}
EXPORT_SYMBOL_GPL(tcp_reno_undo_cwnd);
```

* tcp_reno__undo_cwnd:
  - "undo" mechanism in TCP Reno congestion control.

  - When TCP suspects that packet loss has occurred due to congestion (as inferred from multiple duplicate ACKs or a timeout), it will reduce the cwnd and ssthresh as part of its congestion control strategy.

  - The tcp_reno_undo_cwnd functionality is designed to "undo" the cwnd reduction if it is later revealed that the action was taken in error, thereby restoring the cwnd to its previous state before the presumed congestion event.