Programming Assignment

Improving the performance TCP Reno algorithm

Due date : Dec. 12, 2025

Assignment Overview

- Goal
  - Refactor/Modify TCP Reno, which is one of the TCP Congestion algorithms implemented in the Linux Kernel and test the performance of the modified TCP Reno.

- Assignment Environment
  - Ubuntu 20.04 LTS (VM Installment **Contents 0,1**)
  - Network Environment(Network Environment setup **Contents 2,3,4**)
    - 10 Connections, Throughput : 1Mbps, RTT : 200ms Environment Setup -> Fairness, Link Utilization, Congestion Control Analysis

Assignment Explanation

- Specified Assignment Goal
  - Analyze problems in Reno congestion control algorithm under specific network environments, then modify/refactor the Reno algorithm to conduct experiments and analyze the results

- Assignment Process
  1. Network Environment setup
  2. Measure performance of Reno in the setup from step 1
  3. Modify Reno congestion control algorithm
  4. Measure performance of modified Reno from step 3
  5. Analyze results and write report

** When completing the assignment, it is recommended to write code and improve performance using the provided reno_custom.c. This is because the Reno congestion control code included in the Linux kernel source, tcp_cong.c, does not contain code related to the custom module. However, it is also acceptable to directly modify /net/ipv4/tcp_cong.c for the assignment.

Assignment Explanation

- Example of Various Network Conditions(freely configurable)
  - 10~1000 large number of concurrent TCP connections present
  - Network with high bandwidth and high latency
  - Network with high packet loss due to link errors
  - Network with significant delay variations (jitter)
- It is sufficient to compare and analyze the modified algorithm with the existing Reno under a single network condition. However, additional points can be earned by analyzing the robustness of the modified algorithm across multiple network conditions.
- To confirm fairness, experiments are required under conditions with more than five TCP connections.

Assignment Explanation

- TCP Performance Evaluation Metrics
  - The following performance metrics must be assessed, and various other network performance evaluation metrics can also be utilized
    - Link Utilization
    - Fairness
    - Latency
  - It is not necessary for all evaluation metrics to show improvement.

Assignment Explanation

- Submission
  - 1) Assignment Report (Intermediate)
    - Free structure, Should include description of the progress made up to the submission date (Not strictly graded), submitted in PDF format
  - 2) Assignment Report (Final)
    - Minimum of 6 pages (no limit on additional pages), written in Word or Hancom, and submitted in PDF format.
    - Report Structure
      1. Analysis of the problem situation you set up and the experimental results of Reno in that situation
      2. Problem solving strategy
      3. Explanation of modification done to TCP Reno
      4. Performance evaluation and analysis
  - 3) C code (Final)
    - Custom TCP congestion algorithm that improves the performance of /net/ipv4/tcp_cong.c (refer to 36p)

Assignment Explanation

- Assignment Deadline
  - Intermediate (Report only) : **~ 2025/10/31 23:59:00**
  - Final (Report + Code) : **~ 2025/12/12 23:59:00**
  - In the case of late submission, **25% deduct per day,** after 3 days it will be considered no submission
    - Ex) For student with 80 points

      12/13 submission -> 80*0.75=60

      12/14 submission -> 80*0.5=40

      12/15 submission -> not 0, considered as no submission

Assignment Explanation

- Assignment Explanation Table of Contents
  - **0.Ubuntu image Download(20.04), 1.VM Installation** refer for Ubuntu 20.04 environment setup
  - **2.Mininet Introduction, 3. Installation, 4. Demo** refer for building an actual network topology through Mininet emulator
    - Through provided demo example, simple socket communication can be performed, and the network topology implemented using Mininet will be utilized for measuring performance of TCP congestion algortihm
  - **5.Building Custom TCP Network Module** refer for confirming whether a simple custom TCP congestion algorithm example can be inserted as a module into actual Ubuntu OS.
    - Through this section, it will be possible to deploy an improvised version of custom implemented TCP_RENO congestion control algorithm as a module into an actual OS.
    - The performance of the custom implemented congestion control algorithm can be verified and analyzed in the previously built network topology for assignment completion.

0. Ubuntu image Download(20.04)

1. VM Installation

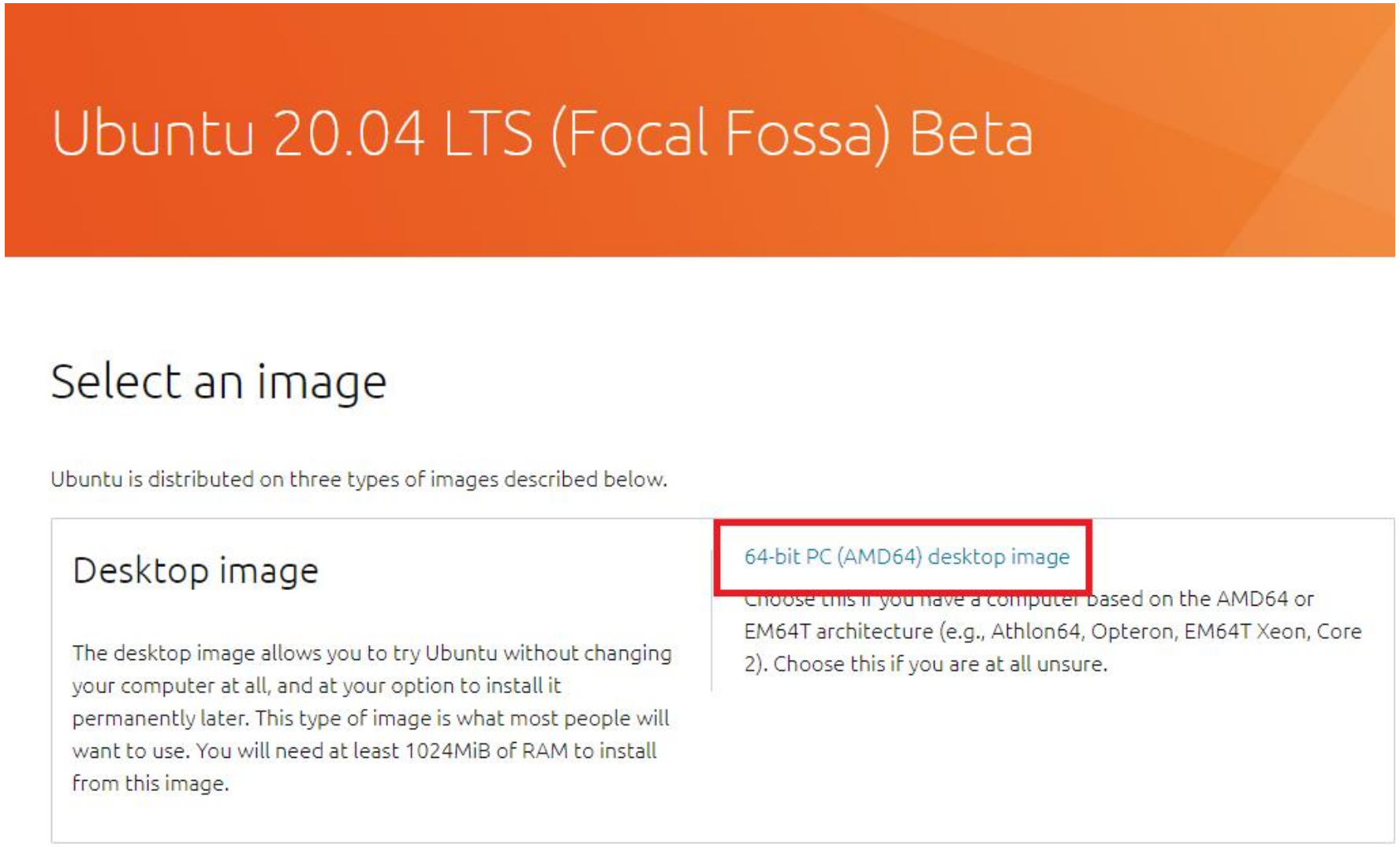
2. Mininet Introduction
3. Mininet Installation
4. Mininet Demo

5. Building Custom TCP Network Module

6. Reference Material
Appendix – TCP Reno Code Analysis

0. Ubuntu image Download(20.04)

http://old-releases.ubuntu.com/releases/focal/ - You must use this link version of Ubuntu 20.04

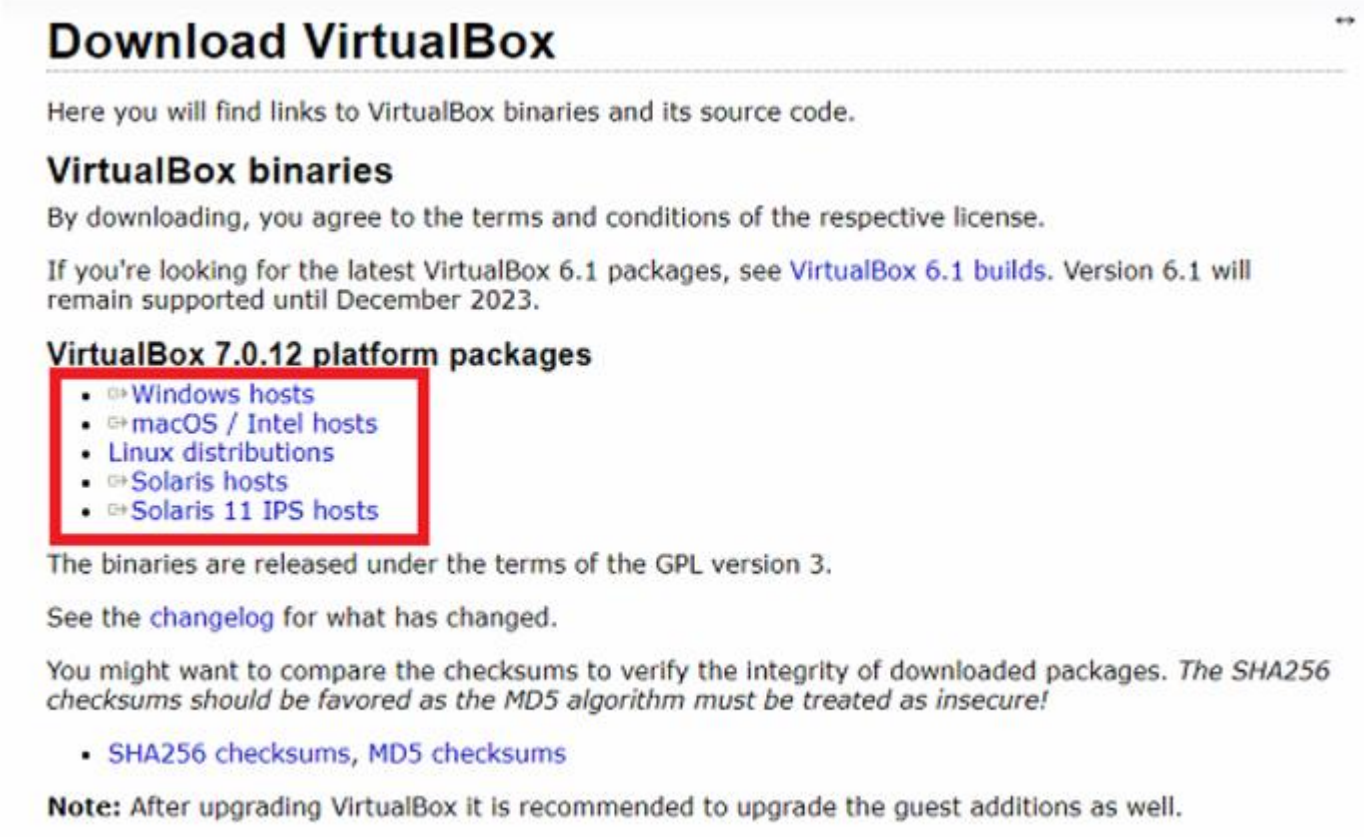


1. VM Installation (Virtual Box)

https://www.virtualbox.org/ Visit

1. VM Installation (Virtual Box)

Download according to your OS



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

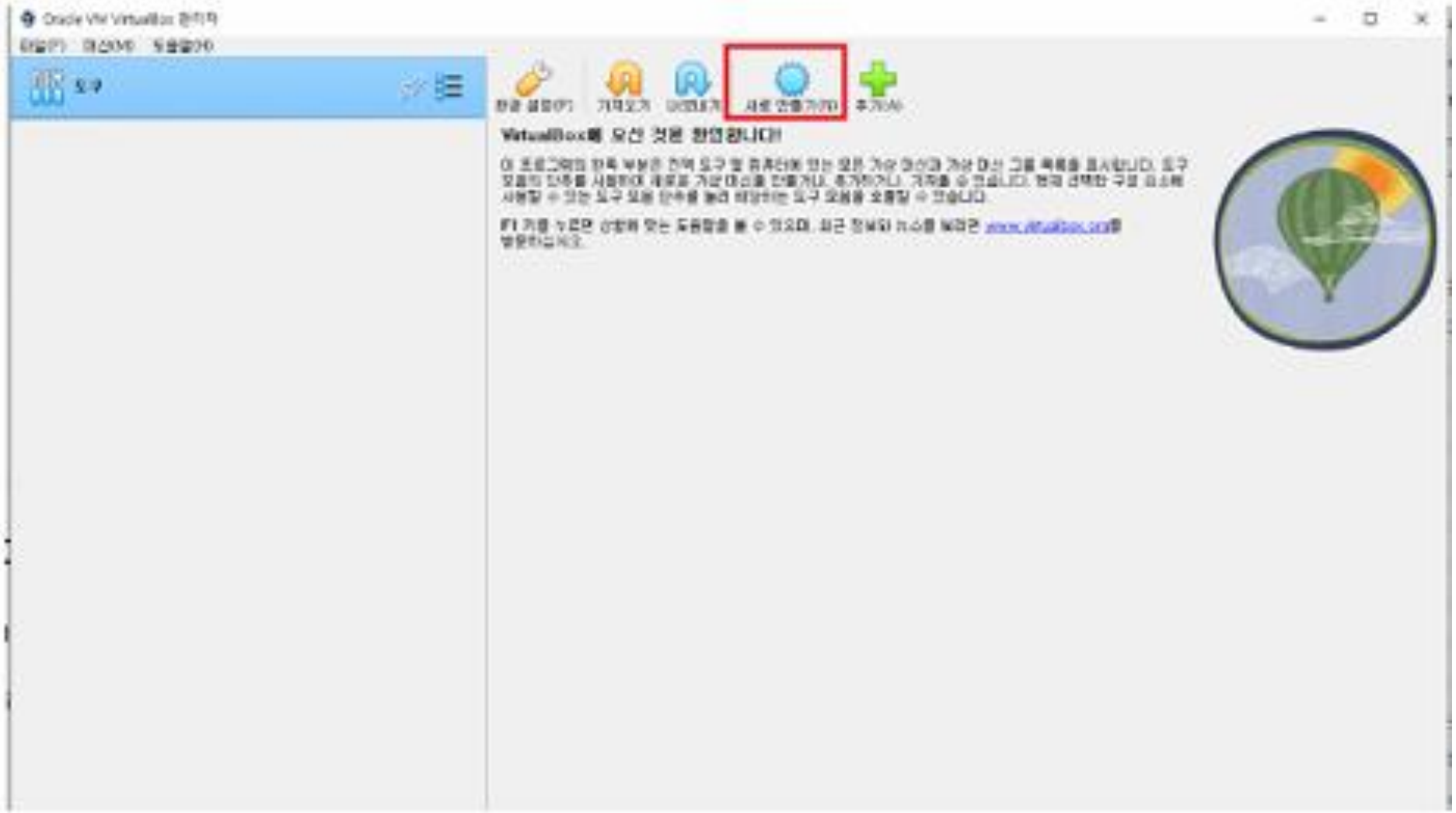
Note: After upgrading VirtualBox it is recommended to upgrade the guest additions as well.

1. VM Installation (Virtual Box)

Download- press next, ok, yes, install in order(default settings install)

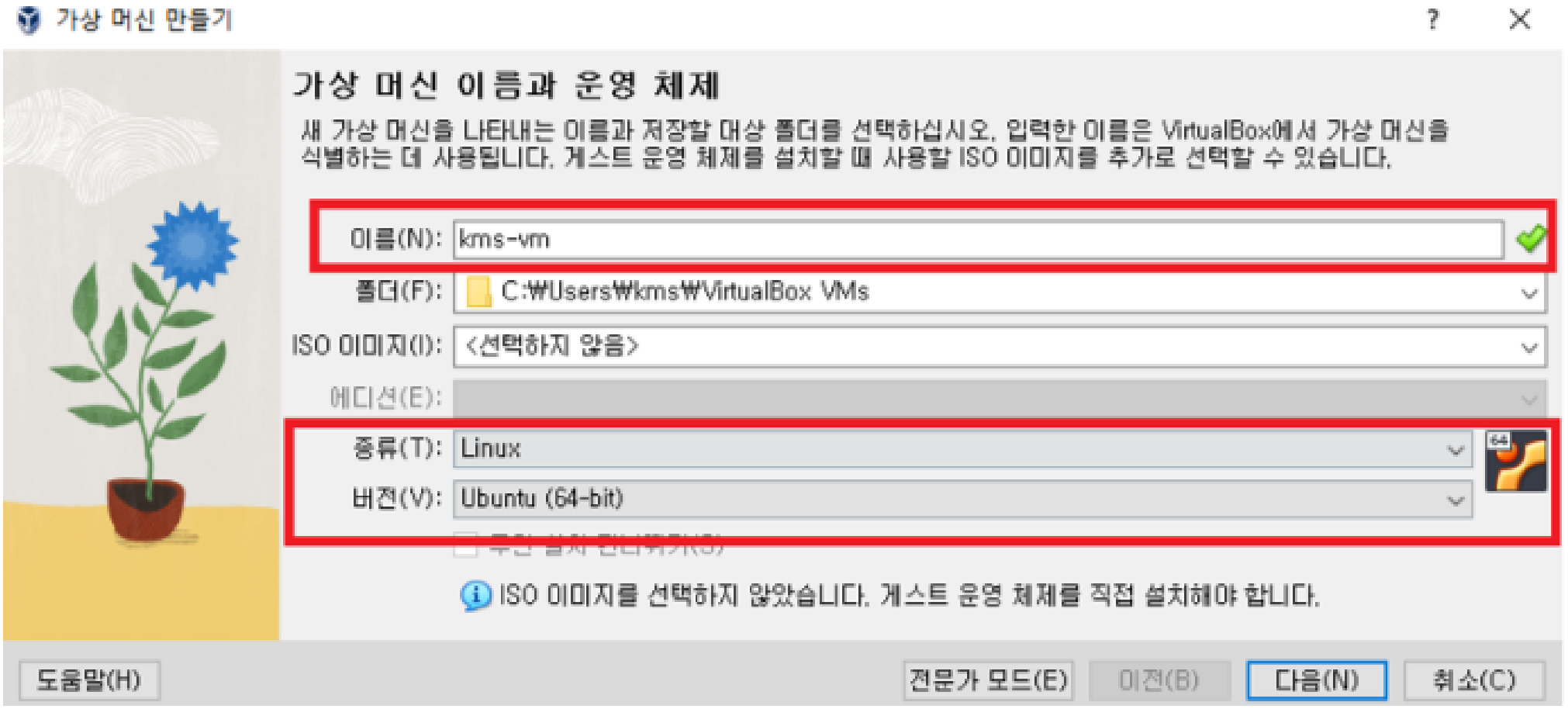
1. VM Installation (Virtual Box)

Create New Virtual Machine



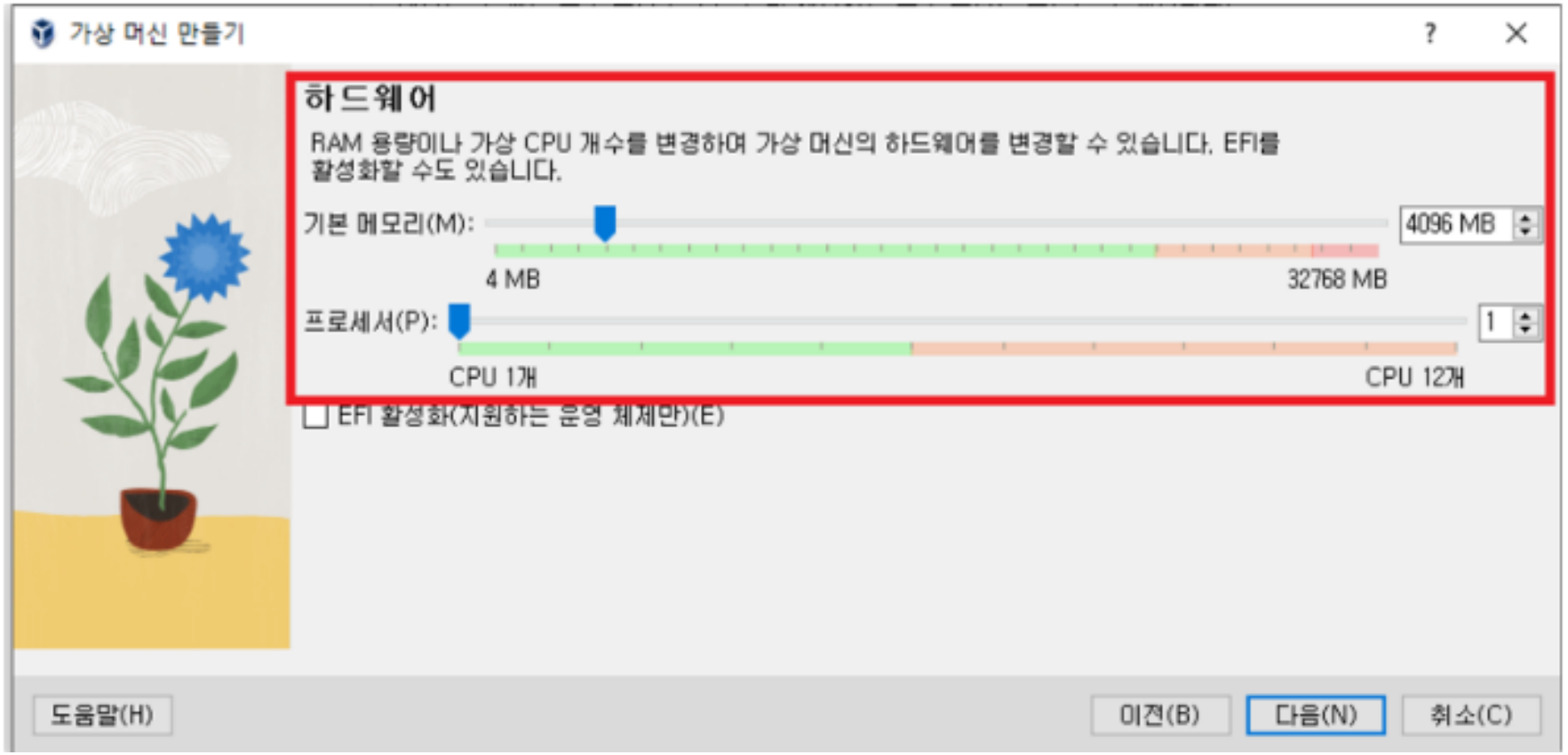
1. VM Installation (Virtual Box)

-Fill in Name, Type(Linux), Version(Ubuntu)
-ISO image should be <not selected>



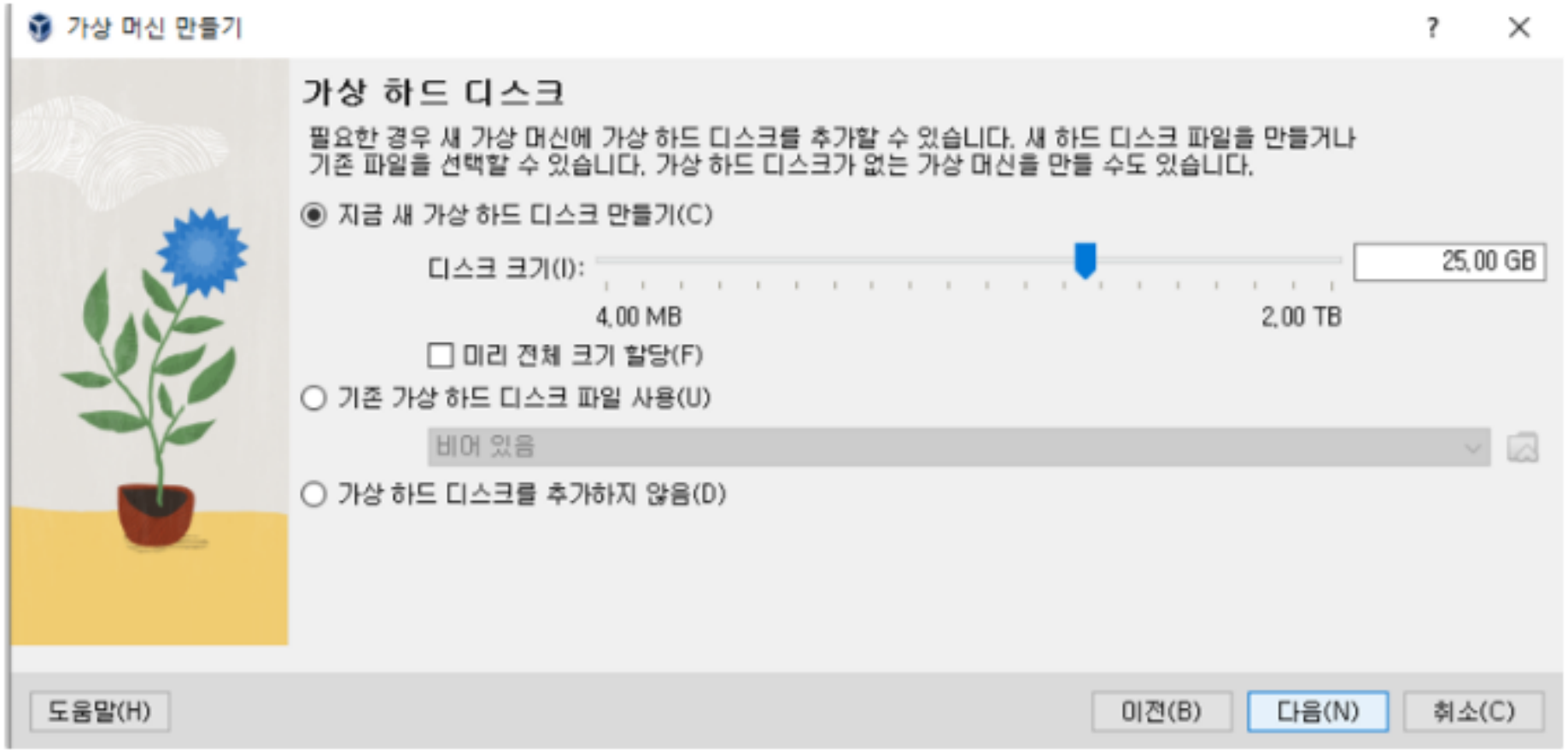
1. VM Installation (Virtual Box)

The hardware specs should be set according to your PC specs



1. VM Installation (Virtual Box)

Continue with default settings

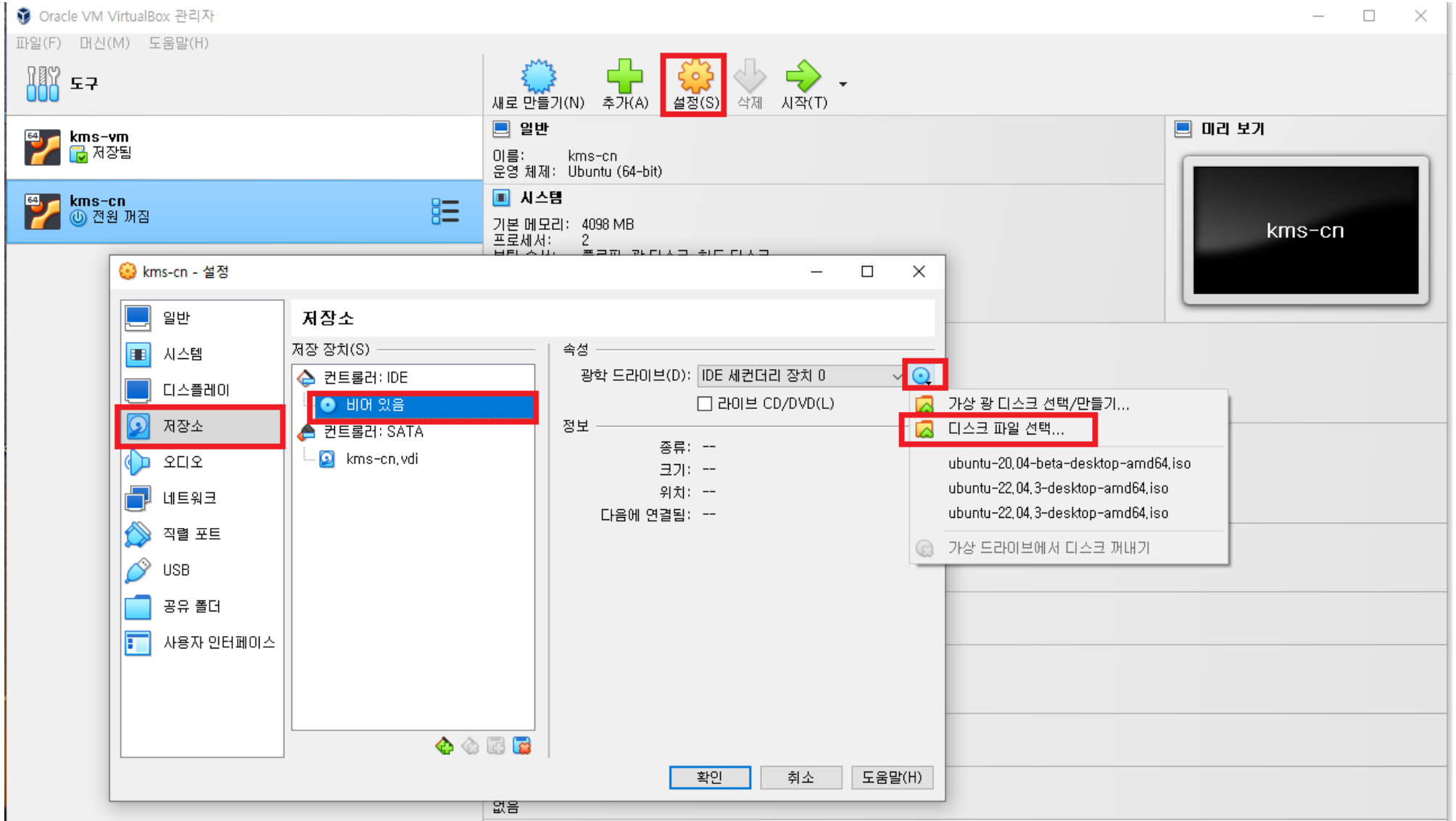


1. VM Installation (Virtual Box)

Configure the Virtual Machine

1. VM Installation (Virtual Box)

VM settings - ISO file setting (select the downloaded Ubuntu image)

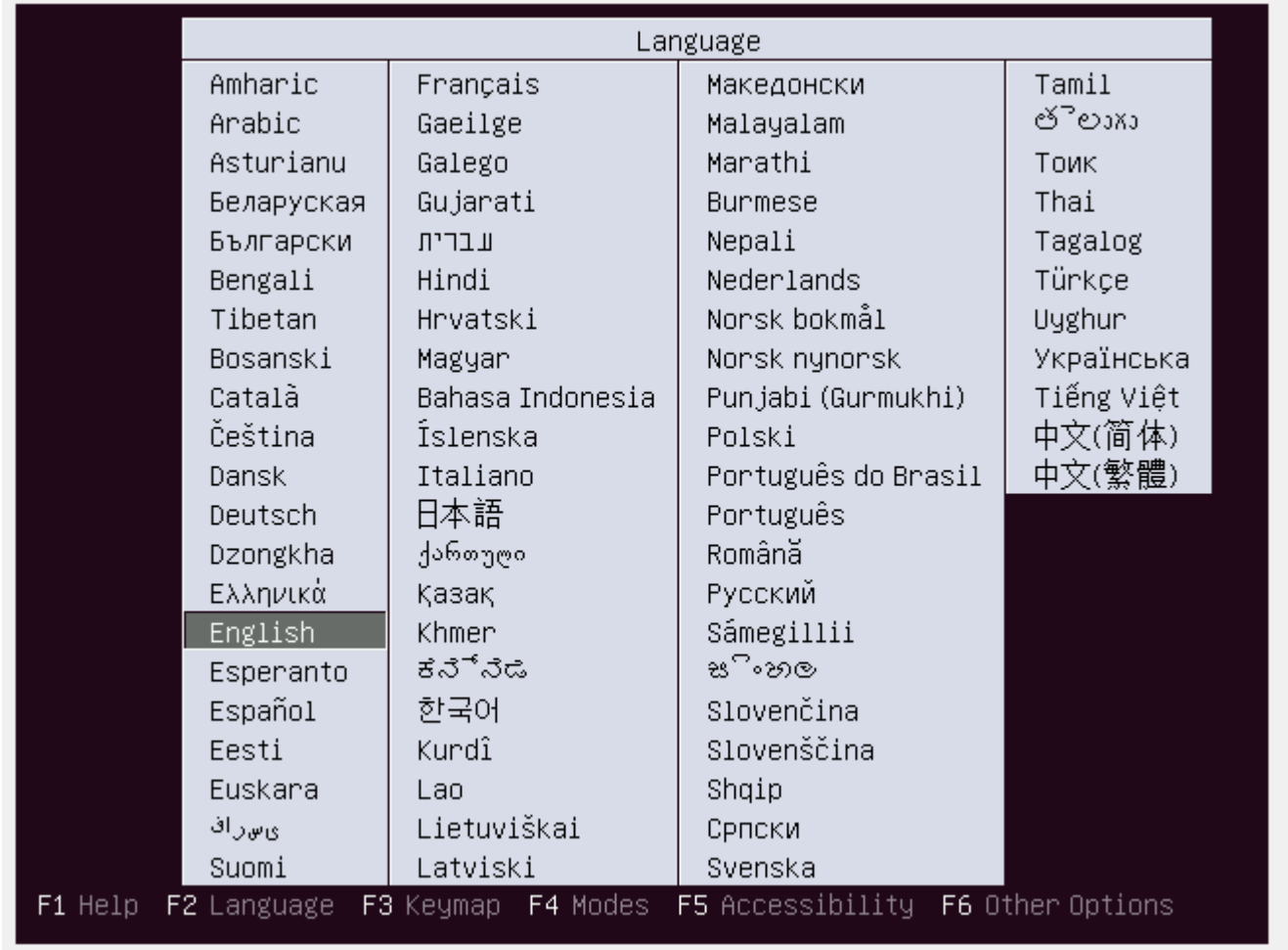


1. VM Installation (Virtual Box)

Start the Virtual Machine

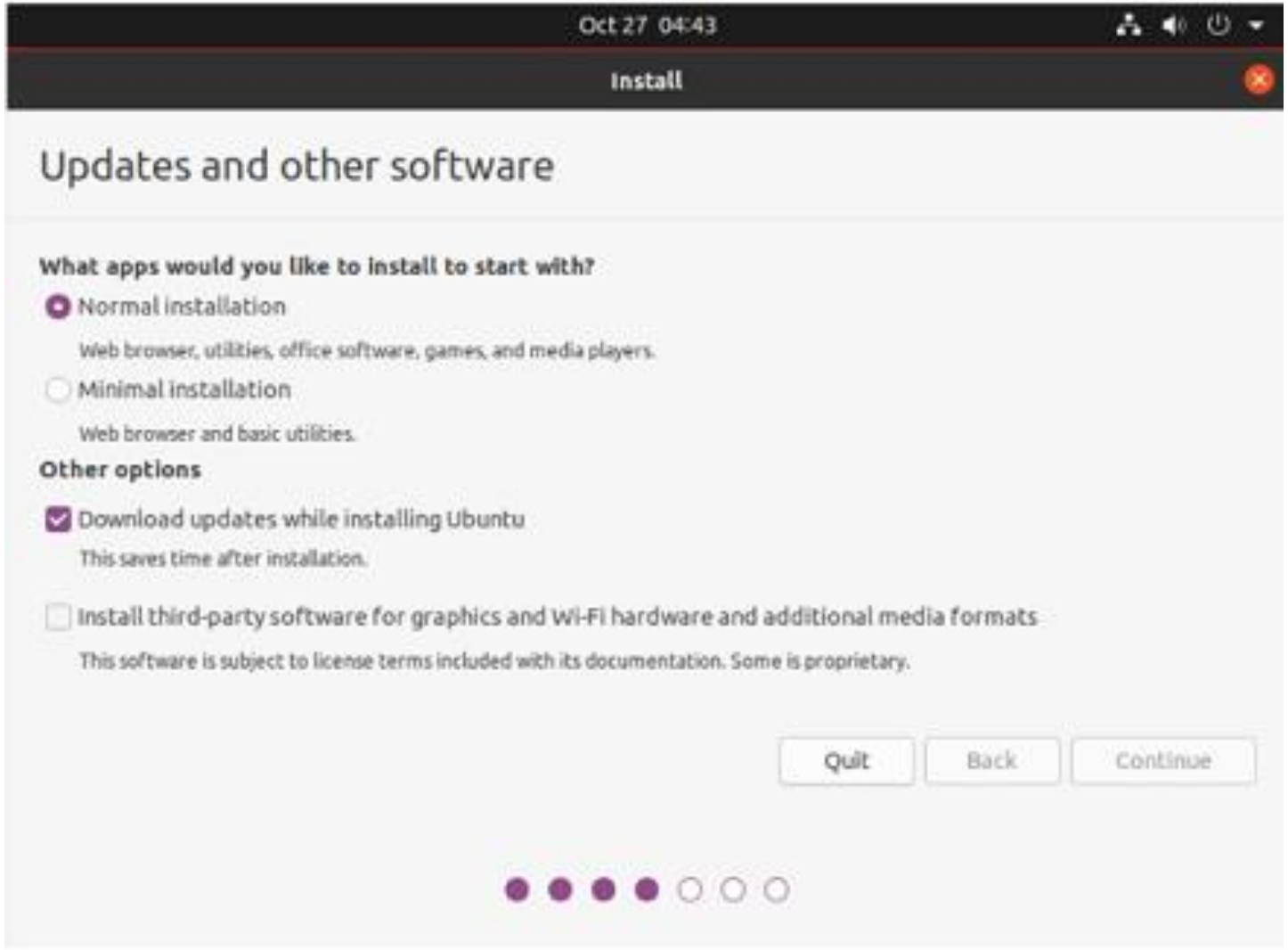
1. VM Installation (Virtual Box)

Ubuntu Installation
- Select English and press Enter, then select Install Ubuntu



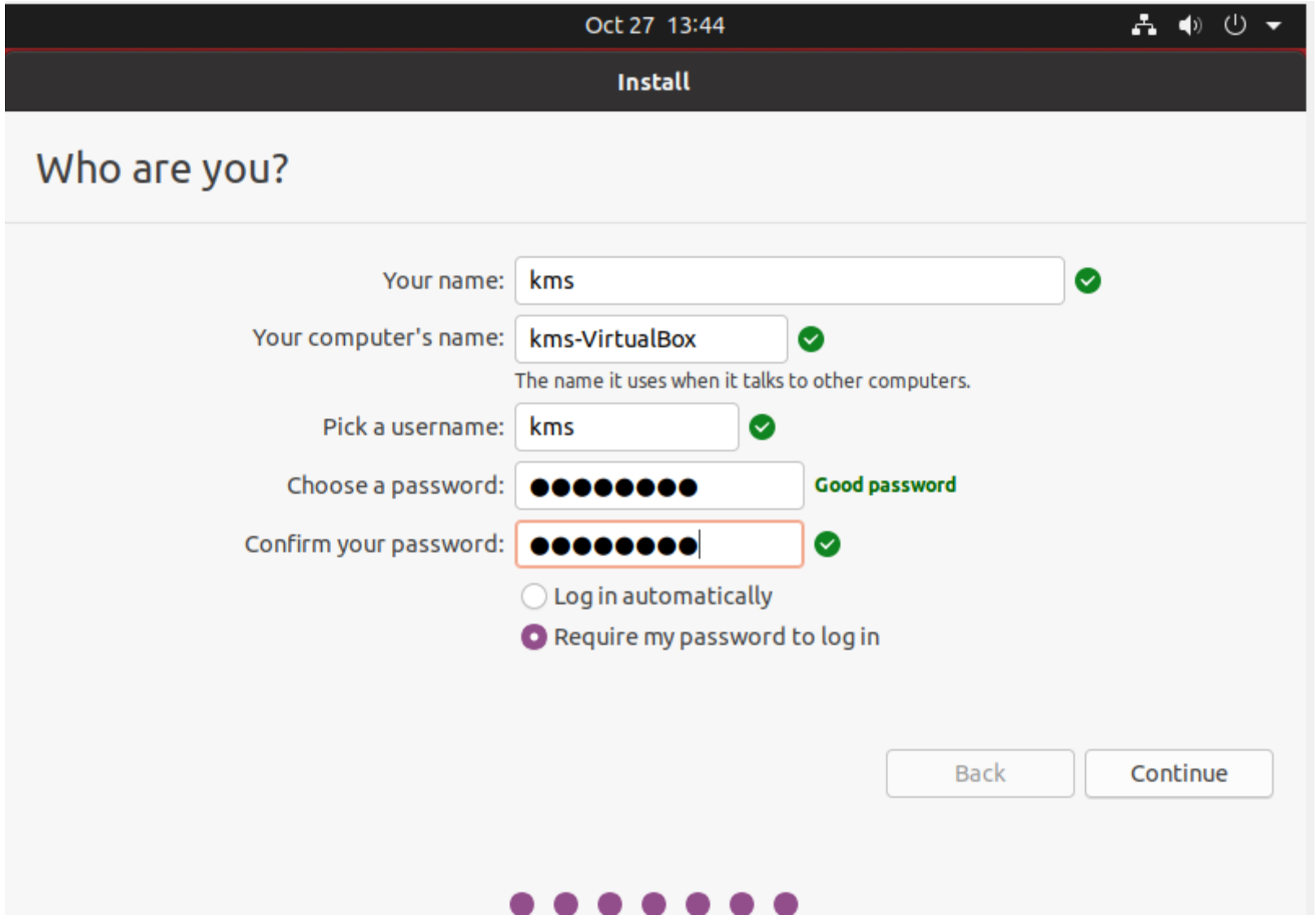
1. VM Installation (Virtual Box)

Ubuntu Installation
- Continue with default settings (Next, Yes, Install, Continue)



1. VM Installation (Virtual Box)

Ubuntu Installation
- Fill in Name, Password freely



1. VM Installation (Virtual Box)

Ubuntu Installation
- Continue with default settings (Next, Yes, Install, Continue)

End!

Linux Configuration

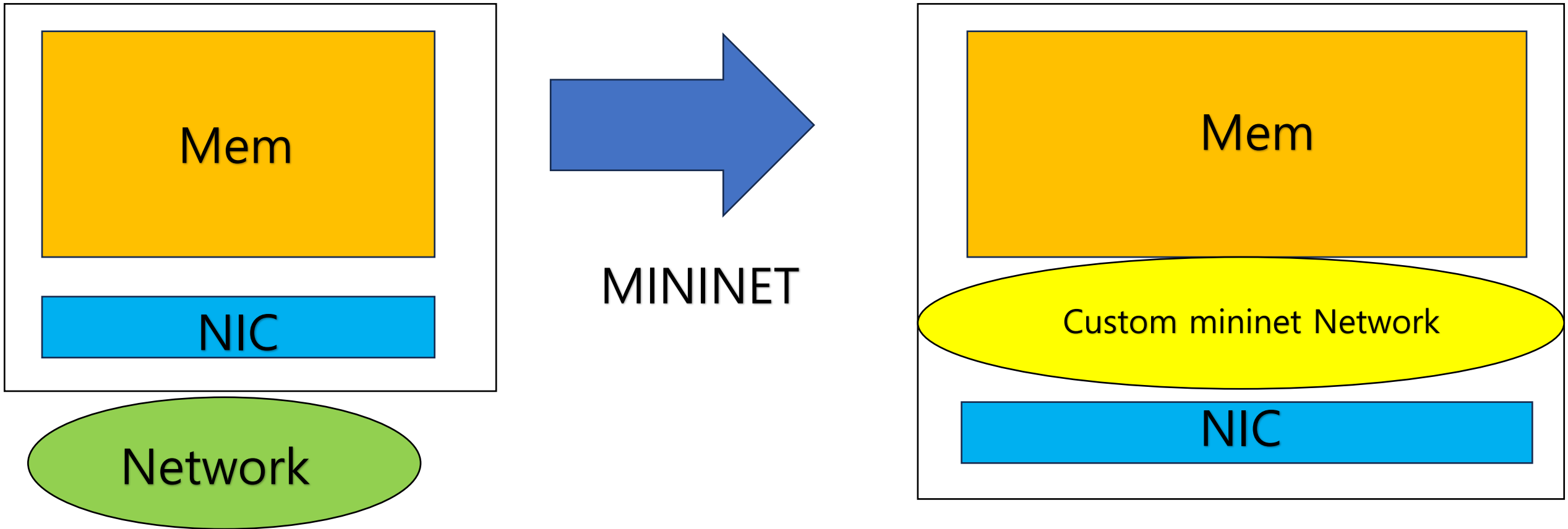
Open Linux terminal

$ sudo passwd <= set root password

$ 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

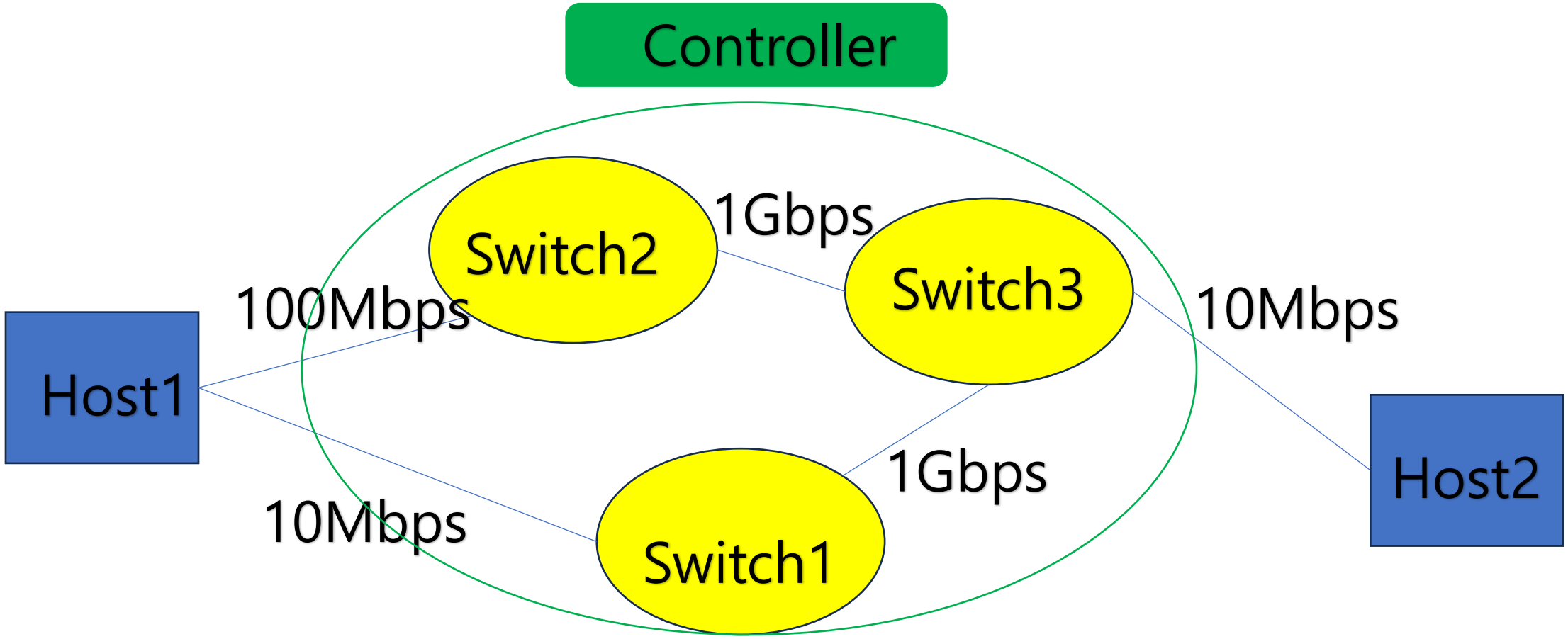
2. Mininet Introduction

- Network Emulator
- Sets up a virtual network environment within a single machine.



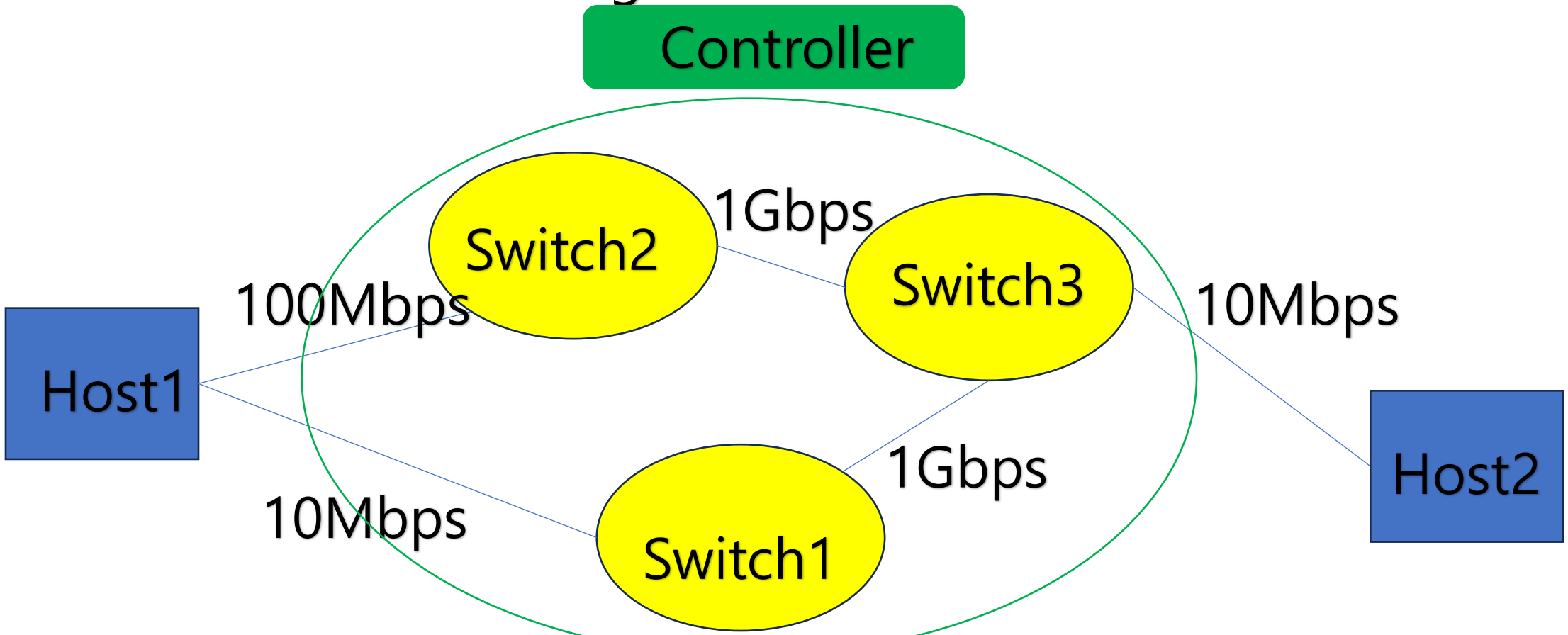
2. Mininet Introduction

- Can configure a custom virtual network topology



2. Mininet Introduction

- Host: packet send/receive + Node that executes application
- Switch
- Controller: switch management



3. Mininet Installation

https://github.com/mininet/mininet/blob/master/INSTALL refer to 3

$ 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 Installation

Confirm Downloaded Files

```
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$
```

Confirm Python Version (version 3.x)

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

3. Mininet Installation

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 Installation

Mininet Installation Confirm

$ 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

Download Example.py from ICampus

```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

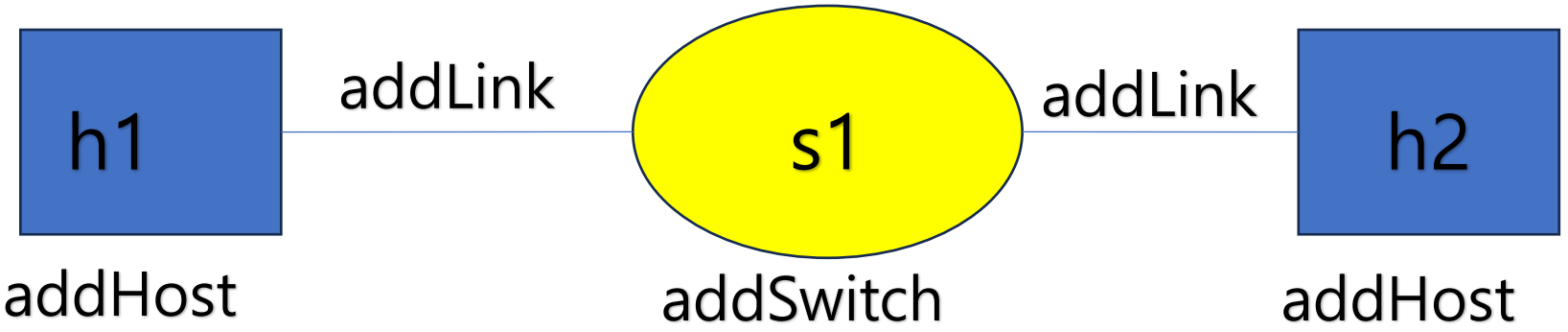
Example.py run
$ sudo python3 Example.py

```
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. Declaration of network topology of user choice

```
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)
```

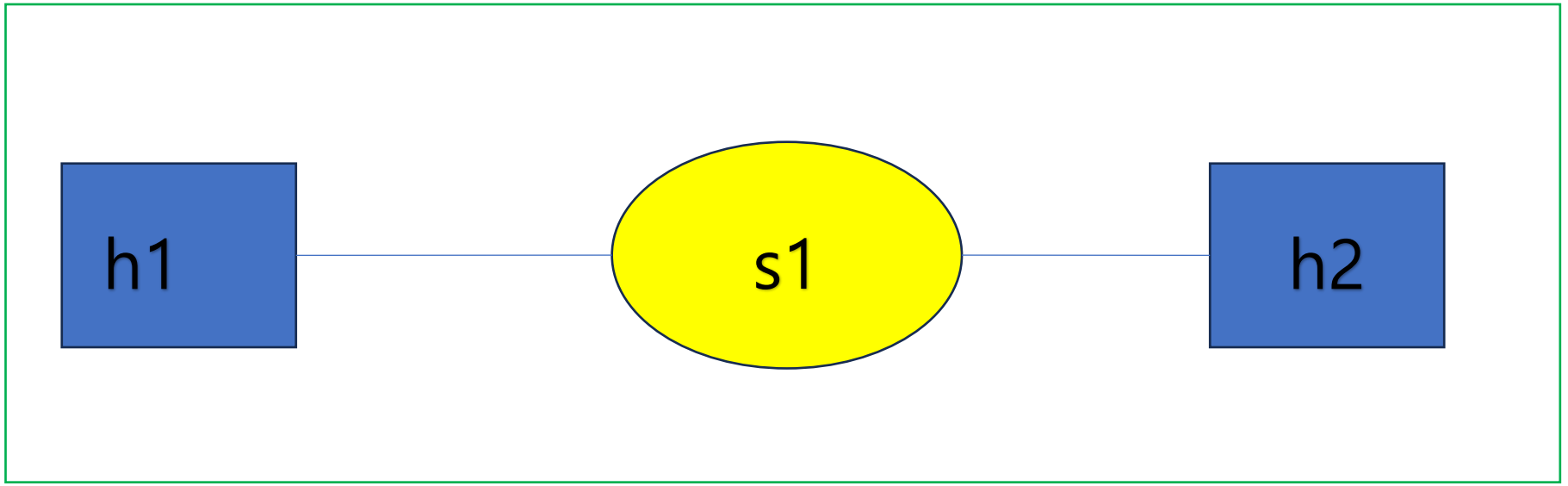


4. Mininet Demo1

2. Topology declaration / net declaration

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

net



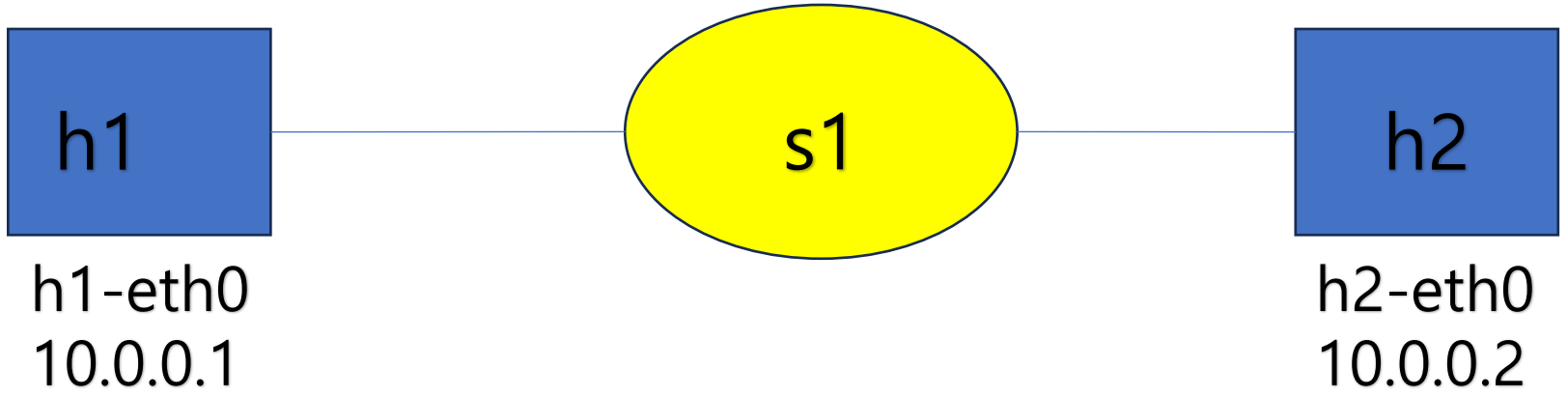
4. Mininet Demo1

3. host ip, interface name configuration

```
    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")
```



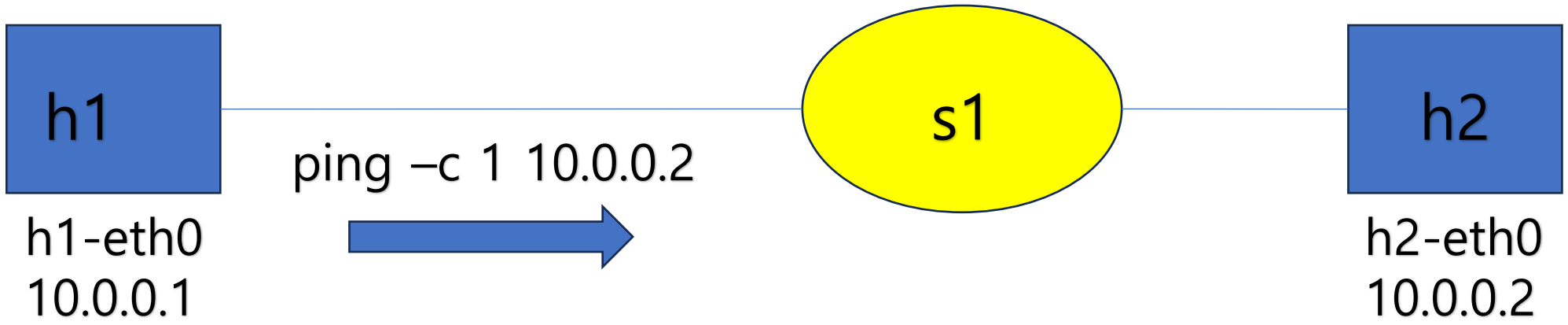
4. Mininet Demo1

4. Command execution on Mininet

```
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. Result Check

```
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. Download udp_client.py, udp_server.py, udp_socket.py from ICampus

```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. Result Check

```
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

Download Example_1.py from ICampus

```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

Link attribute configuration (bandwidth, delay, loss ..)

```
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 execute and print result

```
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

Result Check

```
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 options”
6.Reference Material 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 Usage example
- https://github.com/mininet/mininet/tree/master/examples

Mininet Official Document
- http://mininet.org/

dmesg options
- 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

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

```
/* 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);
}
```

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.