NS-3 Basics

CSE 322 - Computer Networking Sessional

Special Credits:

1. Syed Md Mukit Rashid, Assistant Professor (on leave), CSE, BUET. PhD Student @ Pennsylvania State University.
2. Mashiat Mustaq, Lecturer (retd), CSE, BUET. PhD Student @ Purdue University.

NS3 installation steps

- Version to be used - **3.43**
- *https://www.nsnam.org/docs/release/3.43/installation/html/quick-start.html*.
- You may check out this for a video walkthrough.
- Follow the exact steps mentioned in the link. Install the **prerequisites** first.
- After following the steps, run the following command :

```
$ ./ns3 run hello-simulator
```

- If it outputs "`Hello Simulator`", then it was installed correctly.

NS3 installation steps

- *Some modules may not build due to missing dependency,* which won't be a problem. You may solve this error by installing the missing dependencies if you wish.

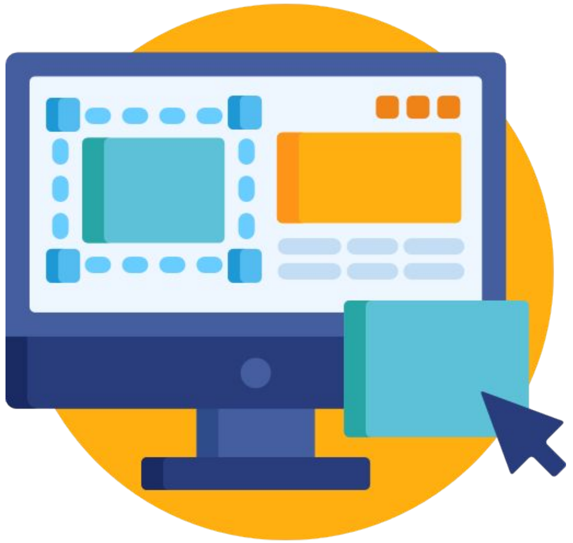
- Python bindings are not required for this course.

NS3 - A Network Simulator

- **Resources:**
    - **Official Website :** https://www.nsnam.org/
    - **Tutorial** : https://www.nsnam.org/docs/release/3.43/tutorial/ns-3-tutorial.pdf
        - Useful chapters : 5-9
    - **Models:** https://www.nsnam.org/docs/release/3.43/models/html/index.html
        - Description of models are provided here. Helpful for understanding the concepts.
    - **Doxygen API documentation:**
        - https://www.nsnam.org/docs/release/3.43/doxygen/index.html
    - **IDEs :** VSCode, Jetbrains CLion etc (Install the necessary plugins)
    - **Google group :** https://groups.google.com/g/ns-3-users
    - **YouTube Playlist :** https://youtu.be/bjUNbXBmA2c?feature=shared

Conceptual Overview - Node

- A basic computing device abstraction, e.g : Computer
- A class defined in C++
- Purpose :
    - Adding functionality such as applications, protocol stacks, peripheral cards etc



Conceptual Overview - Application

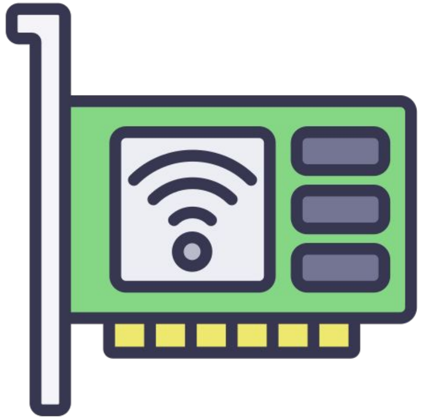
- Representation of user-level software applications
- A class defined in C++
- Purpose :
  - Runs on nodes to to run different types of simulations
- Examples:
  - `UDPEchoServer/ClientApplication`, `BulkSendApplication`, `OnOffApplication`, `PacketSinkApplication`etc
  - built- in applications directory : `src/application/models`

Conceptual Overview - Channel

- Abstraction of the media through which data flows in the network
- Connect Node to a channel
- A class defined in C++
- Types :
  - `CsmaChannel` (Ethernet)
  - `PointToPointChannel`
  - `WifiChannel`
  - `WimaxChannel` etc

Conceptual Overview - Net Device

- Abstraction of both software driver and Network Interface Card used to connect a Node to a network
- A net device is "installed" in a *Node* to enable the *Node* to communicate with other Nodes via *Channels*.
- A *Node* may be connected to more than one *Channel* via multiple *NetDevices*.
- Types :
  - `CsmaNetDevice` (Ethernet)
  - `PointToPointNetDevice`
  - `WifiNetDevice`



Conceptual Overview - Topology Helpers

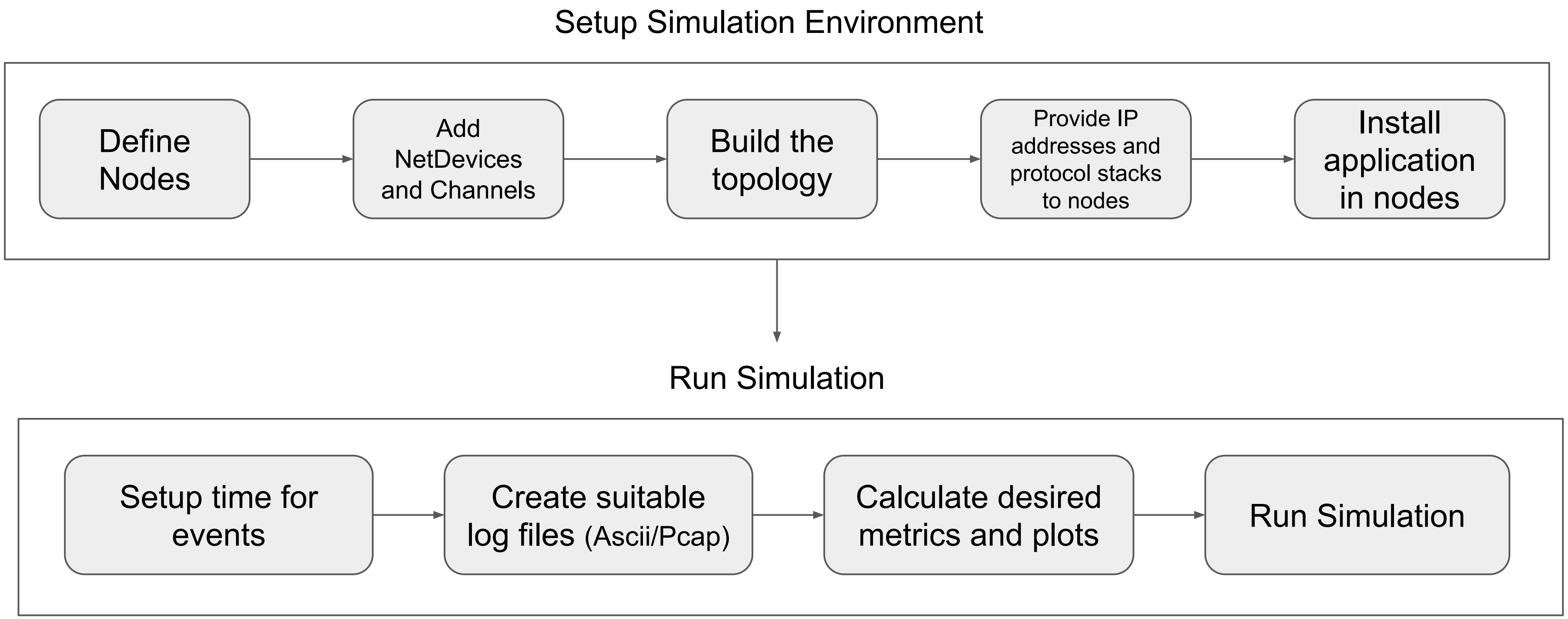
In each simulation, simplify common tasks such as:

- Connecting Nodes to NetDevices
- Connecting NetDevices to Channels
- Assigning IP addresses etc.

combine those many distinct operations into an easy to use model for your convenience

Examples: `PointToPointHelper`, `InternetStackHelper`, `UDPEchoServerHelper`etc.

Simulation Overview


Setup Simulation Environment
Define Nodes
Add NetDevices and Channels
Build the topology
Provide IP addresses and protocol stacks to nodes
Install application in nodes
Run Simulation
Setup time for events
Create suitable log files (Ascii/Pcap)
Calculate desired metrics and plots
Run Simulation

Run first.cc

```
cp examples/tutorial/first.cc
scratch/

./ns3 build

./ns3 run scratch/first
```

When the simulator will stop?

- No further events are in the event queue.
- A special Stop event is found.
    - `Simulator::Stop(stopTime)`
    - Necessary when there are recurring events (WiFi)
    - *Important to call* `Simulator::Stop` *before calling* `Simulator::Run`

Logging Overview

Log Type	Purpose	Macro
LOG_ERROR	error messages	NS_LOG_ERROR
LOG_WARN	warning messages	NS_LOG_WARN
LOG_DEBUG	relatively rare, ad-hoc debugging messages	NS_LOG_DEBUG
LOG_INFO	informational messages about program progress	NS_LOG_INFO
LOG_FUNCTION	a message describing each function called	NS_LOG_FUNCTION - member func. NS_LOG_FUNCTION_NOARGS - static func.
LOG_LOGIC	messages describing logical flow within a function	NS_LOG_LOGIC
LOG_ALL	Log everything mentioned above	no associated macro

Logging Overview

- **LOG_LEVEL_TYPE:** Enables logging of all the levels above it.
    - **Ex : LOG_LEVEL_INFO :** Enable logging for ERROR, WARN, DEBUG, INFO types.
- **NS_LOG_UNCOND** – Log the associated message unconditionally (no associated log level).

Logging Overview

- Using the shell environment variable -> NS_LOG
  - increase the logging level without changing the script

    -
      ```
      export NS_LOG=UdpEchoClientApplication=level_all
      ```
  - Enable two logging components together - Colon separated

    -
      ```
      export
      'NS_LOG=UdpEchoClientApplication=level_all|prefix_func:UdpE
      choServerApplication=level_all'
      ```

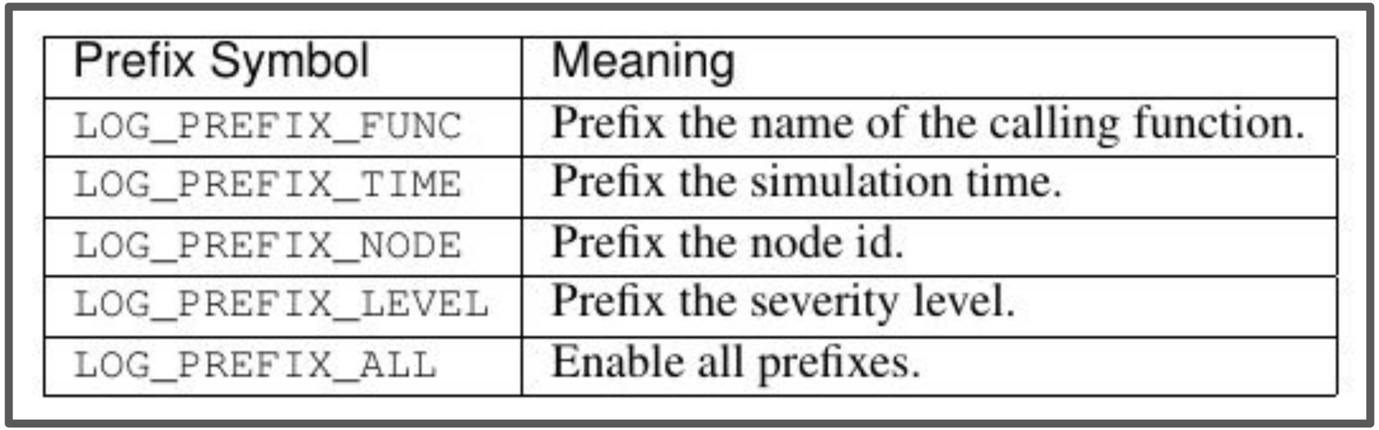
Logging Overview

- Using the shell environment variable -> NS_LOG
  - Distinguish which method generates a log message - ORing

    -
      ```
      export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_func'
      ```
  - See the simulation time

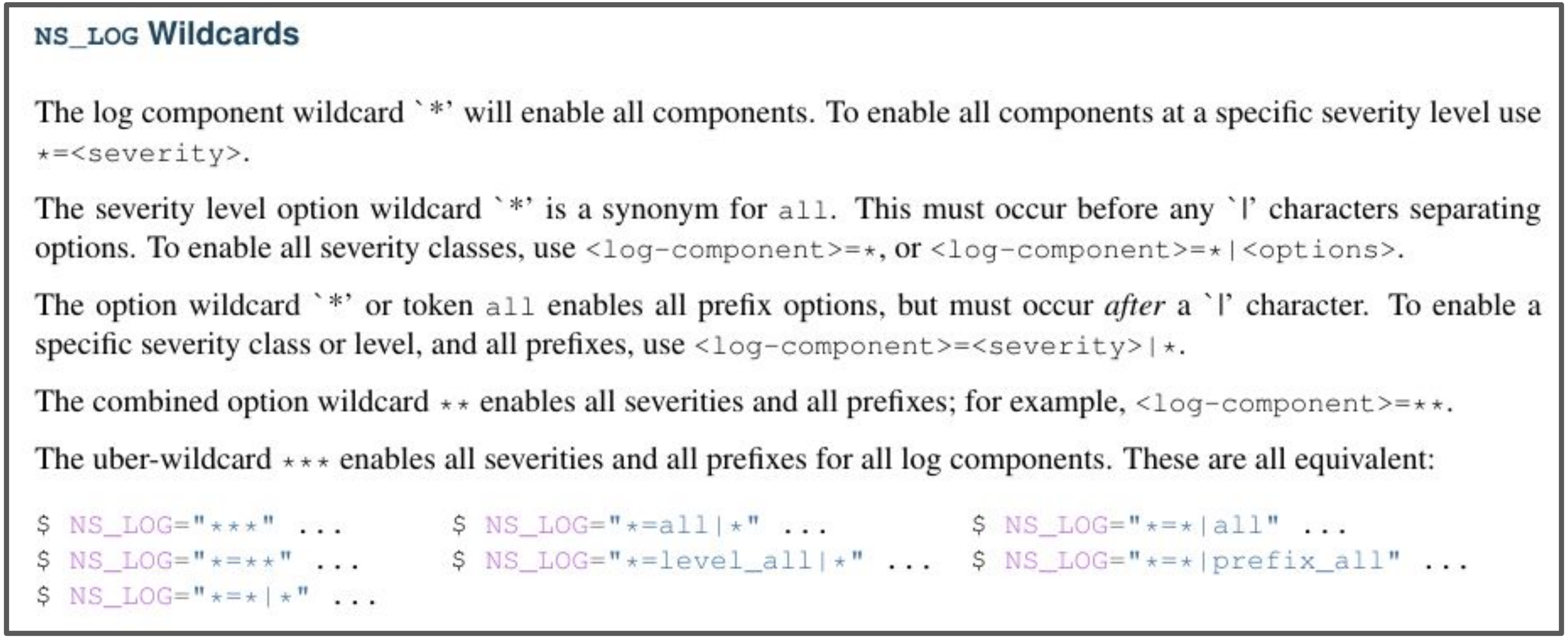
    -
      ```
      export 'NS_LOG=UdpEchoClientApplication=level_all|prefix_time'
      ```

Logging Overview



Prefix Symbol	Meaning
`LOG_PREFIX_FUNC`	Prefix the name of the calling function.
`LOG_PREFIX_TIME`	Prefix the simulation time.
`LOG_PREFIX_NODE`	Prefix the node id.
`LOG_PREFIX_LEVEL`	Prefix the severity level.
`LOG_PREFIX_ALL`	Enable all prefixes.

Logging Overview



NS_LOG Wildcards

The log component wildcard `*’ will enable all components. To enable all components at a specific severity level use `*=<severity>`.

The severity level option wildcard `*’ is a synonym for `all`. This must occur before any `|’ characters separating options. To enable all severity classes, use `<log-component>=*`, or `<log-component>=*|<options>`.

The option wildcard `*’ or token `all` enables all prefix options, but must occur *after* a `|’ character. To enable a specific severity class or level, and all prefixes, use `<log-component>=<severity>|*`.

The combined option wildcard `**` enables all severities and all prefixes; for example, `<log-component>=**`.

The uber-wildcard `***` enables all severities and all prefixes for all log components. These are all equivalent:

```
$ NS_LOG="***" ...          $ NS_LOG="*=all|*" ...          $ NS_LOG="*=*|all" ...
$ NS_LOG="*=**" ...         $ NS_LOG="*=level_all|*" ...    $ NS_LOG="*=*|prefix_all" ...
$ NS_LOG="*=*|*" ...
```

Logging Overview

- Turn off logging previously enabled
  - `export NS_LOG=""`

- Enable logging in code
  - `export NS_LOG=FirstScriptExample=info`

Using Command Line Arguments

- Declare command line parser.
- Show general arguments for a program.

  -
    ```
    ./ns3 run "scratch/first --PrintHelp"
    ```
- Provide new command line argument

  -
    ```
    ./ns3 run "scratch/first
    --ns3::PointToPointNetDevice::DataRate=32Kbps"
    ```
- Provide multiple command line arguments

  -
    ```
    ./ns3 run "scratch/first
    --ns3::PointToPointNetDevice::DataRate=32Kbps
    --ns3::PointToPointChannel::Delay=2ms"
    ```

Using User Defined Command Line Arguments

```
int
main(int argc, char *argv[])
{
uint32_t nPackets = 1;
CommandLine cmd;
cmd.AddValue("nPackets", "Number of packets to echo", nPackets);
cmd.Parse(argc, argv);
… … …
```

ASCII Tracing

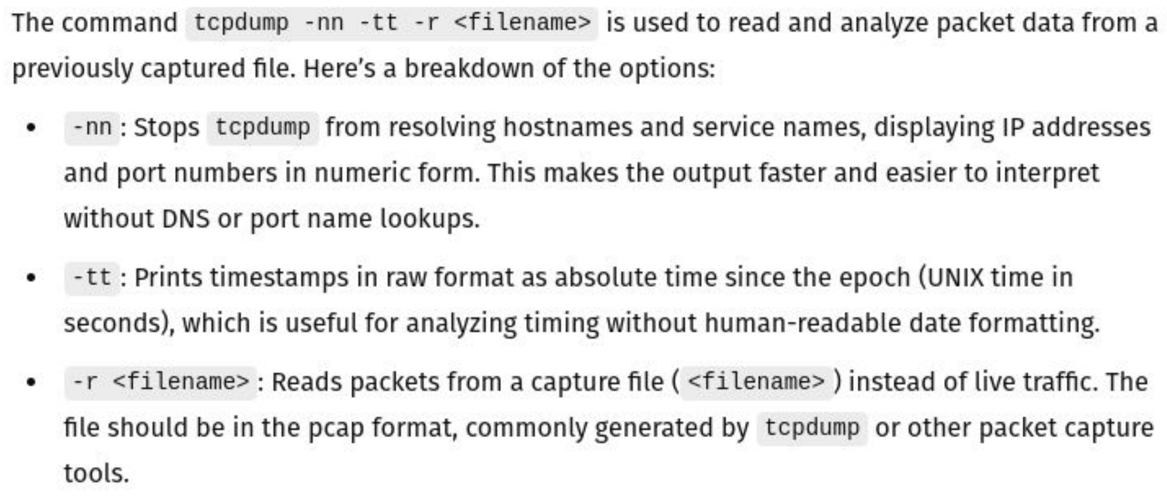
+	An enqueue operation occurred on the device queue
-	A dequeue operation occurred on the device queue
d	A packet was dropped, typically because the queue was full
r	A packet was received by the netdevice

ASCII Tracing

+	Enqueue
2	Time (Seconds)
/NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Enqueue	Trace source origin
ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052 10.1.1.1 > 10.1.1.2) ns3::UdpHeader (length: 1032 49153 > 9) Payload (size=1024)	Packet information

Pcap Tracing

- Wireshark
- Tcpdump
  - tcpdump -nn -tt -r filename.pcap

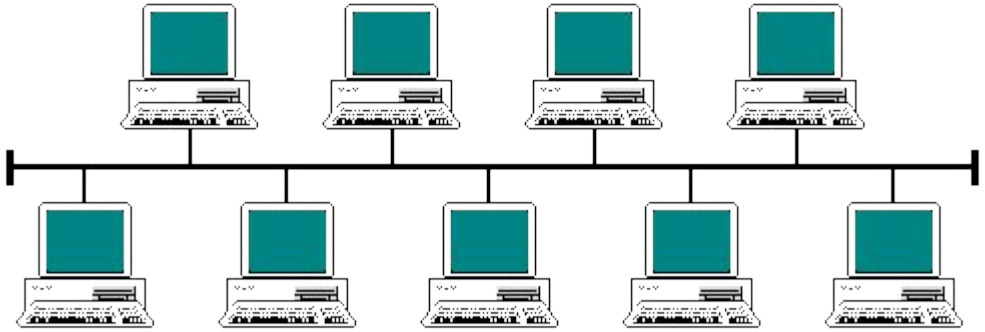


The command `tcpdump -nn -tt -r <filename>` is used to read and analyze packet data from a previously captured file. Here's a breakdown of the options:

- `-nn`: Stops `tcpdump` from resolving hostnames and service names, displaying IP addresses and port numbers in numeric form. This makes the output faster and easier to interpret without DNS or port name lookups.
- `-tt`: Prints timestamps in raw format as absolute time since the epoch (UNIX time in seconds), which is useful for analyzing timing without human-readable date formatting.
- `-r <filename>`: Reads packets from a capture file (`<filename>`) instead of live traffic. The file should be in the pcap format, commonly generated by `tcpdump` or other packet capture tools.

ChatGPT

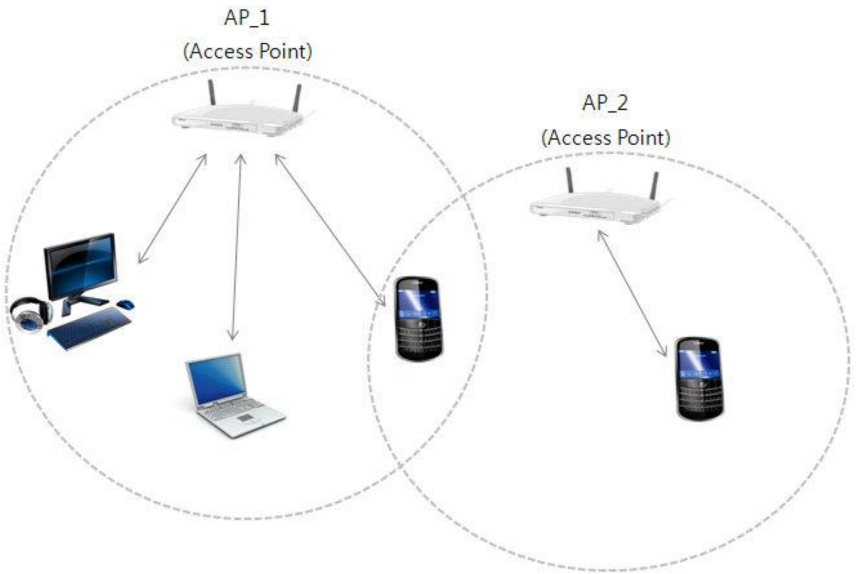
Ethernet (Bus Network)



- CSMA NetDevice and channel
- Promiscuous mode - allows a network device to intercept and read each packet.
- ARP (Address Resolution Protocol) - retrieves the receiver’s MAC address.

Wireless Network

- AP - Access Point
- AP generates beacons continuously
- Beacon - regular transmissions from access points (APs)
    - purpose to inform user devices (clients) about available Wi-Fi services and nearby access points


AP_1
(Access Point)
AP_2
(Access Point)