

## Personnel

# Lecture 1 Network Programming

- **Computer and Communication Networks**, class B (SE and CE).
- **Textbook: Computer Networks [CN]**, 4th Ed. Andrew S. Tanenbaum. Prentice Hall. (Buy the textbook!)
- **Reference book: TCP/IP Illustrated, Volume 1, The Protocols [TV1]**. W. Richard Stevens. Prentice Hall.

## Reference

- CN 1.1–1.4, 6.1.3
- Manual pages: `socket(2)`, `bind(2)`, `listen(2)`, `accept(2)`, `connect(2)`, `getsockopt(2)`, `socket(7)`, `ip(7)`, `tcp(7)`.

Network(0234B)

Network(0234B)-1.1

## Calling for help

If you have difficulties, you can do one of the followings:  
In the order of preference...

- Post a news article to our **newsgroup**, `hku.csis.CSIS0234B`. This is preferred: all of you can answer, and all of you benefit from the answer. Questions asked in any other means may be answered (only) in newsgroup, so **please keep watching the newsgroup**.
- Send us a **mail** at `c0234b@csis.hku.hk`. The mail will end up in the mailbox of me and all tutors.
- Contact us during **tutorials**.
- Contact us directly. Send a personal mail before visiting our office to make an **appointment**.  
This is the order in which I deal with them. I.e., news are first cleared, and only when all news are dealt with I'll deal with mails to `c0234b`, and only after that I'll look at personal mails.

Network(0234B)-1.2

Network(0234B)-1.3

## Lecture conduct

Lectures are free-form, relaxed but serious.

- **Talking** is allowed **only on course matters**.
- **Attend lectures** only if you are willing to learn.  
Nobody force you into lectures anyway.
- **Everything** about the lecture **needs to be understood**  
Preferably during the lecture, and absolutely no later than the end of the day.
- **Ask** whenever you find something you don't understand.  
Don't hesitate: what you don't understand is probably not understood by many others as well, and you're helping others by just asking.
- **Stop the lecture** if you need time to digest. This is **vitaly** important: it is usually the only way for me to know that you need more time.  
This is the way in which you can understand everything during the lecture.

Network(0234B)-1.4

Network(0234B)-1.5

## Mode of instruction

### Lectures

- 26 hours. **Voluntary**.
- Primarily uni-directional flow of information. I teach, you listen. There is a little room for you to participate, though.

### Small Group Tutorials

- Provides supervised lab experience in 13 hours. **Compulsory**.  
But many find that they overrun tutorials, so be prepared to have 26 hours here.
- **Study the reading material well** before going to tutorials.  
They will be available 2 days in advance of each tutorial.
- Indicate preference to the time for tutorial slot after this lecture.  
But since tutorials can't be large, expect to make compromises, i.e., you might get a slot on an otherwise off day.

## Assessment

### Assignments (30%)

- 4 or 5 programming/written assignments in groups of 1–2.

### Quiz (10%), To be announced

- 1 hr quiz of very easy questions. No problems if you have done the assignments and understands the tutorials well.  
This is perhaps a big if, though.

### Tutorial assessment (10%), 13 weeks starting from the next

- Subjective assessment by TAs and lecturer.
- No more than one absense allowed. Each further absense results in a loss of 1/12 of **all** 50% coursework marks.

### Exam (50%)

## Policy: late submission, plagiarism

### Late submission

- Requests for postponing deadlines must be made **at least 3 days** before deadline.
- Assignments late for at most **48 hours** will **score 80%** of the marks it would score if it was submitted on-time.
- Assignments late for more than 48 hours are marked as normal, but will not contribute to the final score.

### Plagiarism

- Any discussions within group are allowed.
- Discussions across group or to outside groups are limited to sharing of concepts. No actual code, text or numbers may be exchanged.
- Plagiarism is **punished as described by departmental guidelines**.

Network(0234B)-1.6

## Advice on learning

- **Learn the concepts first (most important)**. Implementation details are not as important as the concepts.  
We might show code for illustration purpose. But keep in mind that the concepts is more important and must be understood first.
- **Understand, not just memorize**. Find the underlying ideas that give rise to concepts. Find unifying themes.
- **Keep up with lectures**. Each lecture builds on top of the previous ones, so if you miss just one lecture, it will be significantly more difficult to catch up with upcoming lectures.
- **Do the assignments early**. Programming assignments are usually more difficult than what you first think. And assignments clear up your concepts for easy understanding of the coming lectures.
- **Read the book**. Lecture is too short to explain everything in the book, and the function of lectures is simply to guide you to read.

Network(0234B)-1.7

## What will we learn in this course?

- **Hardware** which allows communications.  
The fibre, the wire, the air.
- **Conceptual frameworks** in which we write network software.  
What application writers think the network is.
- How **network software** communicate using the hardware.  
The glue between the framework and the hardware.
- **Problems** that arises when real-world imperfect networks are built and grow, and how to **solve** them.  
What happen if somebody turn off one router for a while.
- How the **Internet** does all these.  
Internet becomes the only thing that count.

Network(0234B)-1.8

## Lecture schedule

<b>Part 1:</b>	Week 1: Admin, Sockets
<b>The two ends</b>	Week 2: Application and Physical layer Week 3a: Modulation and Multiplexing
<b>Part 2:</b>	Week 3b: Link control
<b>Data link</b>	Week 4: Flow control, Windowing protocols Week 5: Multiple Access
<b>Part 3:</b>	Week 6: Network layer, Routing algorithms
<b>Networking</b>	Week 7: Multicasting, Internetworking Week 8: IP Address translation, Quiz Week 9: Internet routing
<b>Part 4:</b>	Week 10: Transport, Connection
<b>Transport</b>	Week 11: Flow and Congestion control
<b>Part 5</b>	Week 12–14: Network security

Network(0234B)-1.9

## Why networking

- **Resources sharing**: processing time, disk space, programs, files, database, environment, equipment, ...
- **Communication**: E-mail, concurrent editing, video-conferencing, instant messaging, chat room, newsgroup, BBS, ...
- **E-commerce**: Advertising, distributing product information, checking stock, placing order, auction, ...
- **Media product distribution and "swapping"**: photos, songs, video, radio programmes, ...
- **Electronic game**: hide and seek, flight sim, ...
- ...

Network(0234B)-1.10

## Classification of network

Transmission media	Network size
• Electrical signal in wire Voltage level	• 1m: Personal area network Inter-device communication
• Light in fiber Presence of light	• 10m–1km (room to campus): local area network ("LAN") Single owner, single technology
• EM-wave Similar	• 1km–10km (city): Metropolitan area network ("MAN") Single owner, multiple technologies
<b>Transmission model</b>	• 100km up (country to planet): Wide area network ("WAN") Multiple owners, multiple technologies Number of owners and technologies determines underlying assumptions.
• Broadcast links one to all	
• Point to point links one to one	

Network(0234B)-1.11

## Software for network

From the mechanism that provide the basic communication channel, we need to work towards a model for programmers to write code.

- A program in one computer can use the communication channel to send and receive messages with another program in another computer.
- A program in a computer can act as a **server** waiting for someone to contact it.
- Programs in other computers (**clients**) can request service from it. ("client-server" model)
- Alternatively, both communicating parties may act as both client and server (**peer-to-peer**, P2P).
- Depending on who is communicating (Government, Business, Consumer), funny names are given to them (B2C, B2B, G2C, C2C).

Network(0234B)-1.12

## Socket interface

- Many programming interface to communicate between programs of different computers.
- The traditional interface used by Unix is "**Berkeley socket interface**", which other systems (e.g., Windows) is also modelled against.
- Works (a bit) like files: the kernel keeps **sockets**, or half-connection. They works as **file descriptors**.  
So once created and configured correctly, they can be *read*'ed and *write*'d.
- How to configure? Servers use system calls *bind*, *listen* and *accept* to provide an access point for others to connect.
- Clients use the system call *connect* to connect to a server.
- The server needs a "well-known port number" to allow others to connect. The client needs translating hostname to "IP address" to connect.

Network(0234B)-1.13

## Example server code

```
#include <sys/socket.h> // Socket calls
#include <netinet/in.h> // Use internet
#include <unistd.h>
#include <iostream>
int main() {
    int sock = socket(PF_INET, SOCK_STREAM, 0); // Create socket
    struct sockaddr_in addr; // The address
    int value = 1;
    setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &value, sizeof(int));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port = htons(5000); // Well known port number
    bind(sock, (struct sockaddr *)&addr, sizeof(struct sockaddr));
    listen(sock, 5);
    int newsock = accept(sock, 0, 0); // accept a connection
    write(newsock, "Hello\n", 6); // Write new socket
}
```

Network(0234B)-1.14

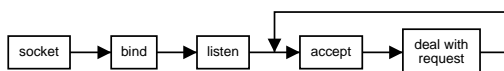
## Example client code

```
#include <sys/socket.h> // Socket calls
#include <netinet/in.h> // Use internet
#include <arpa/inet.h>
#include <unistd.h>
#include <stdio.h>
#include <iostream>
int main() {
    int sock = socket(PF_INET, SOCK_STREAM, 0); // Create socket
    struct sockaddr_in addr; // The address
    char msg[6];
    addr.sin_family = AF_INET;
    addr.sin_port = htons(5000);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    connect(sock, (struct sockaddr *)&addr, sizeof(addr));
    read(sock, msg, 6);
    write(STDOUT_FILENO, msg, 6);
}
```

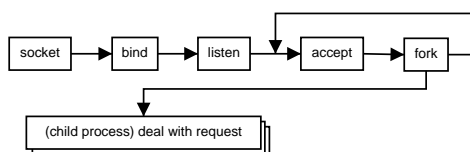
Network(0234B)-1.15

## Server cycle

- The server we show perform exactly one connection, and then die.
- Most servers won't work this way. Instead, it tries to repeatedly accept connections one after another.



- Even this is usually not good enough, since the server can only handle one request at a time. They create a process for each connection.



Network(0234B)-1.16

## Ease of use

- Although we know next to nothing about the network, we can successfully write programs that interacts through the network.
- If this is not easy enough, there are other language bindings (e.g., C++ or Python) making it even easier.
- Sometimes **coding read-write becomes too tedious**, and it is beneficial to **call remote functions** just like local functions, or **invoke remote object** just like local objects.
- Again, they are made easy by **libraries** for different languages.
- So **programming network application needs very little knowledge about the underlying network**.
- We have an easy to use application programming interface, we have a mechanism for data transfer. The question is how it is done.

Network(0234B)-1.17

## Complications

But wait... we are writing programs that transfer data between computers, but essentially we write our program to **expect no error**.

- What if, in the middle of a wireless communication, a big truck comes by blocking the line of sight?
- What if one of the computer in the middle goes down completely?
- What if the partner is currently busy executing another process?
- What if magnetic flux turned one of the bits we send from 0 to 1?
- What if other stations also want to send data at the same time, perhaps exceeding the capacity of the network?
- ...

Why we can still write programs that expects no error?

And, how to guard against someone "eavesdropping" the channel?

Network(0234B)-1.18

## Layering and protocol stack

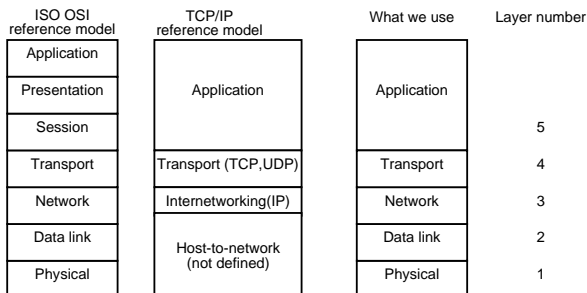
- There are a **lot of problems** sitting between the physical media and the programming interface.
- It is hopeless to solve all of them "in one shot" (by a single set of rules of transmission).
- We also want it modular. E.g., if we change the error detection, we want to reuse unaffected parts.
- Network software is developed in **layers**. Each layer is responsible for some needed functionalities.
- E.g., one of the lower layers may implement error detection, and upper layers will always assume that the data contains no undetected errors.  
What if the assumption is wrong? The lower layer might need modification, or the upper layer can add some more mechanism against it.
- This way designing complex protocol becomes more manageable.

Network(0234B)-1.19

## ISO OSI reference model vs. TCP/IP

Unluckily, for most functionalities, there is no reason why it must be placed in a particular layer, and how fine the layers need to be.

So there are multiple incompatible ways to divide...



Network(0234B)-1.20

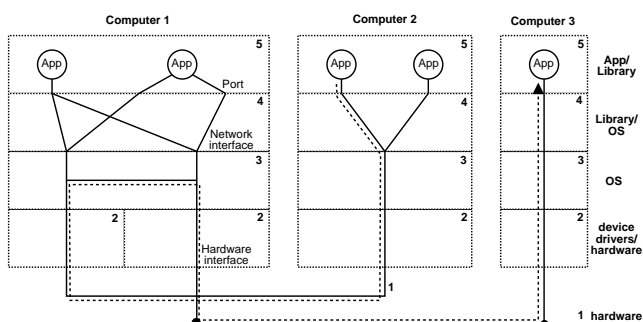
## Function of each layer

- **Physical:** Define how **signals** can be exchanged, e.g., voltage level, frequencies used, modulation, timing, maximum size of network, etc.  
This enables communication.
- **Data link:** Turn the signal stream into **data packet stream**. Error handling, framing (start and stop of frames), flow control, contention, etc.  
This makes communication uniform across technologies.
- **Network: Routing,** fragmentation and reassembly, congestion control.  
This turns connections into networks, allowing any pair to communicate.
- **Transport: Multiplexing** connections, connection management, flow and congestion control (again).  
Thus creating application interface for user programs.
- **Application: Specific** application requirements.  
All user programs, as well as some libraries, are in this layer.

Network(0234B)-1.21

## A schematics

With these functionalities defined, the network works this way:



Arrows shows how an application message is sent from computer 2 to computer 3.

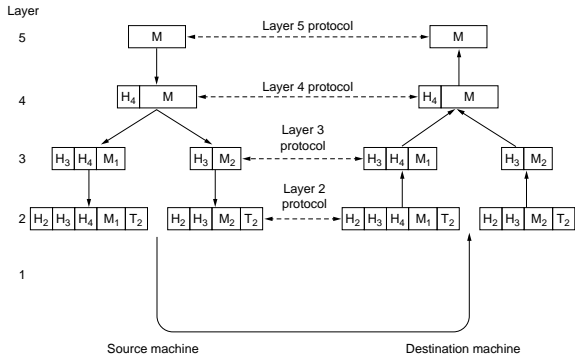
Network(0234B)-1.22

## Protocol design within layer

- When layer  $n$  sends data, it is sent to layer  $n - 1$  of its own computer.  
Or, "send data through the layer  $n - 1$  interface".
- It is expected that the same data will **re-emerge** in the other computer from layer  $n - 1$  to layer  $n$  towards the intended recipient.
- The net effect is like **layer  $n$  of the two computers are talking with each other**, even though in fact they just talk with layer  $n - 1$ .
- Upon receiving data from layer  $n$ , layer  $n - 1$  determines what to do to achieve the intended effect.  
I.e., make layer  $n - 1$  of the other computer to generate data to layer  $n$ .
- Data is **modified** to achieve the functionality of the layer, by either breaking it to multiple parts, adding header and trailer to it, or changing it a bit. The other end does the reverse to recover the original message.  
E.g., layer 2 would add a "check-sum" for error detection.

Network(0234B)-1.23

# How messages are delivered and re-emerge in other side



Here  $M$  is broken up into  $M_1$  and  $M_2$  for transmission.