

## Personnel

### Lecture 1

#### Network software: an Overview

- **Computer and Communication Networks**, class B (other than CSIS and SE).
- **Textbook: Communication Networks**, Fundamental Concepts and Key Architectures [CN]. Leon-Garcia Widjaja, 2nd Ed. McGraw-Hill. (Buy the textbook!)
- **Reference: TCP/IP Illustrated, Volume 1, The Protocols** [Tlv1]. W. Richard Stevens. Prentice Hall.

#### Reference

- CN 2.4

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 news-group, so **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**.
- Make an **appointment** with us by sending a personal mail.  
This is also 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 (if they are not overwhelmed by spams).

Network(0234B)-1.2

## 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 to 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 more time to digest. This is **vital**: it is usually the only way for me to know that you need more time.

Network(0234B)-1.4

## Mode of instruction

### Lectures

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

### Small Group Tutorials

- Provides supervised lab experience in 13 hours. **Compulsory**.  
But many overrun tutorials, so be prepared to have 2 hours per week.
- **Study the reading material well** before going to tutorials.  
They are available at least 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, e.g., you might get a slot on an otherwise off day.

Network(0234B)-1.3

## Assessment

### Assignments (30%)

- 4 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 Feb 3

- Subjective assessment by TAs and lecturer.
- No more than one absense allowed. Each further absense results in a loss of 1/12 of the **40%** you get in tutorials and assignments.

**Exam (50%)**, overrides overall assignment score if exam score is higher.

Network(0234B)-1.5

## 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 usually needs more time 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?

Our primary focus: **digital communication in computer networks**.

- **Conceptual frameworks** in which we write software to use the network.
- What we need from **hardware** to communicate.  
Characteristics of fibre ,copper wire and the "ether".
- What are the common **characteristics** of networking hardware, and what are the implications on our networking system.
- What additional **services** we want our networking system to provide, and how to **achieve** them through networking system software.
- The commonly used **protocols** and **tools** that allow all these be done.
- The **Internet** will be our primary example.

Network(0234B)-1.8

## Lecture schedule

- Week 1: Admin, Sockets, Layering
- Week 2: Application layer
- Week 3: Physical layer
- Week 4-5: Data link
- Week 6: Multiple Access
- Week 7-10: Network layer (Week 10: make up class)
- Week 11: quiz
- Week 12-13: Transport layer
- Week 14 (revision week): Network security

Network(0234B)-1.9

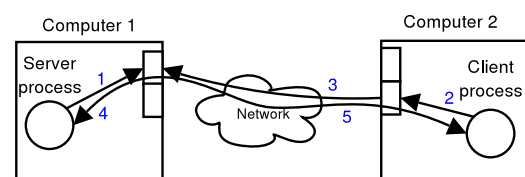
## How to write software to use the network?

Most computer systems that use the system use the following model... although other models are possible, as we will learn later during tutorials.

- A program in one computer (called the **server**) starts running. It will run "forever", i.e., seldom stop, and if stopped it will resume soon.
- The program allocates a **port** in the OS and **listen** to it. Here a port is a metaphor to differentiate network programs in the same computer.
- Another program (called the **client**) runs, possibly in another computer. It also allocates a port, and use it to **connect** to the server.
- The server **accepts** the connection, and the two computers communicate through the connection.
- One or both parties **release** the connection. The server listens to the port again for any incoming connections.

Network(0234B)-1.10

## The steps involved



- 1: Server allocates a well-known port and listen
- 2: Client allocates a temporary port
- 3: Client connects to the server port
- 4: Server accepts and thus creates a connection
- 5: Data is sent and received until one side breaks the connection

Note: "server" and "client" are programs, not computers, because it is usually meaningful for both to sit in the same computer.

Network(0234B)-1.11

### Some conceptual clarifications

- How to **identify a computer**? Each computer must have a **network address** in order to participate in a network communication. In the Internet, it is a 32-bit **IP address** that is **globally** unique.
- How to **identify a port**? Typically, it is a **number** unique within the computer, managed by the OS. In the Internet, it is a 16-bit **TCP port number**.  
"TCP" is the primary way in which connections are made in the Internet.
- How the **client knows** the address and port number of the **server**? Basically, by somebody knowing it. The computer running the server program has a fixed network address, and the server program typically listens to a fixed ("well known") port number.  
The client will tell the server both information when trying to connect.
- What if **another party** connects when a connection is in progress? That **depends**. Some systems would just ignore the new request, some allows programs to make new connection while having a connection.

Network(0234B)-1.12

### The socket programming interface

- We want the **OS to keep track of the TCP ports and connections** so that we don't have to worry about them: just like the way OS keeps the filename internally so that we don't need it for every file read/write.
- The most widely used application interface is called **Berkeley socket** interface, which uses **sockets** to keep track of ports and connections.
- A socket is a **type of file descriptor (fd)**, so we can use standard Unix system calls *read()*, *write()*, *close()*, etc, on them.
- For files, we use *open()* to get an fd and linked to a particular file. For sockets, we use *socket()* to **get an fd**. Servers can use *bind()* to **link it to a TCP port**, while client will let the system to choose ports.
- Some further system calls operate on sockets for **listening** to ports, **accepting** connections, **connecting** to a remote port, to setup some **options** for the socket, etc. Details: tutorial 1.

Network(0234B)-1.13

### What is hidden?

It is not that difficult to write network programs. In this course we will examine **what makes that possible**. E.g.,

- How raw hardware channels is used for sending and receiving digital signals, and how finite-length message are carried.
- How a LAN can reduce system cost.
- How channel errors or crashes of computers are tolerated.
- How to adjust transfer speed to avoid choking up receiver or network.
- How a message is routed through a network of multiple computers to reach destination, and how this fact can be hidden.
- How to recover from problems that arises when different parts of the message route through the network differently.

Network(0234B)-1.14

### The communication medium

An easier first step: to establish communication.

- We need a channel which can be used for communication...
  - It links up **two or a few** computers.
  - It communicates **raw bits** with some physical phenomenon.
  - It gives occasional **errors**.
- A lot of details have to be defined. E.g.,
  - Type of communication: point-to-point or broadcast channel.
  - Physical media used to connect the computers: wire or wireless.
  - Which part of the channel is used?
  - What is the physical shape of the media used?

Network(0234B)-1.15

### The role of headers

We want to "shape up" the communication medium to deal with each of the problems that we have a couple of slides before.

How? When sending data, we also send some **control information** in the **header/trailer** to achieve extra functionality.

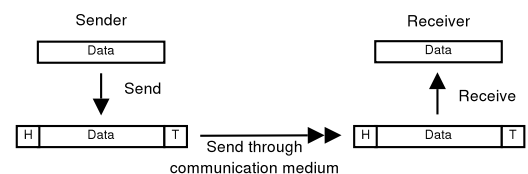
Header = before actual message, trailer = after actual message.

One simple example: **error control**.

- If the medium has an error rate that is too high, we can compute a "**checksum**" of each message and send it with the actual message.
- The receiver also computes the checksum and **check whether it is the same** as what is sent by the sender. If it isn't, the receiver requests for a **resend**, thus removing the error.

Network(0234B)-1.16

### Protocols



Note: most of the time, control info is **not** sent in a separate channel. Instead it is sent using the same channel.

For such scheme to be successful, the two communication ends must **agree on how to interpret** the messages. Such agreement is called a **protocol**.

One of the **main objectives of our course**: how to wisely use headers and protocols to solve our problems.

Network(0234B)-1.17

## Layering

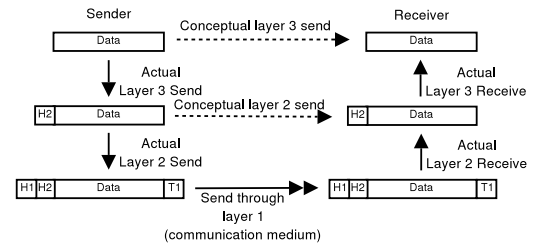
But... there are too many problems to solve in one-shot with a single header. Instead, we want to do it step-by-step, in **"layers"**.

- Start with the physical medium, accepting its limitations. Call this the **low-layer service** that we can access.
- Then we add a few functionality on top of this low-layer service, by **making use of a certain protocol** whenever we use this service.
- This results in a new, **high-layer service**, with extended (or completely different) functionality.
- This extended service is then **treated as a low-layer service** and the above is repeated until all the desired functionalities are added.

So at the same time there are **many** protocols working together. We call a collection of such concurrently working protocols to be a **protocol stack**.

Network(0234B)-1.18

## A simple three-layer protocol stack



Not every layer has a trailer. But each layer has a separate header.

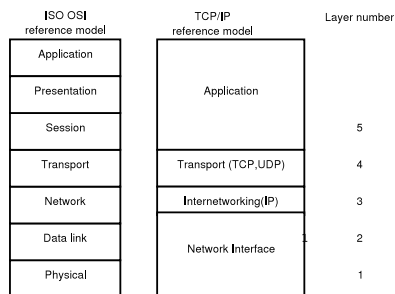
When designing an upper-layer (e.g., layer 3) protocol, we **ignore** the fact that the lower-layer (e.g., layer 2) might not be "directly" connected.

Instead, we simply utilize the service advertised by the lower layer.

Network(0234B)-1.19

## ISO OSI reference model vs. TCP/IP

The ISO OSI model is designed, with 7 layers, to facilitate development of standards. TCP/IP, the standard for the Internet, is designed separately from ISO OSI, but we can still view it as having a layer structure.



We draw layers for only one end of the communication, for convenience.

Network(0234B)-1.20

## The data link layer

What does each layer provide? Let's have a brief overview before we examine each of them in detail.

- The **datalink layer** is responsible for **framing**: turning the continuous bit stream into a data packet stream. Each part of the bit stream is usually called **frames** (to give the feeling that the bit stream is "partitioned").
- It is also responsible for...
  - **Error detection and recovery**: Discover errors in the bit stream and recover from it.
  - **Multiple access**: In broadcast media, allows connected parties to agree on who to send at each time. Usually separated into its own "MAC sub-layer" just below datalink (Multiple ACcess).

Network(0234B)-1.21

## The network layer

- The **network layer** is responsible for **routing**: turn a set of computers connected using links into a single network.
- This is done by having some intermediate computers ("routers") to help forwarding messages until it reaches the intended computer.
- Network layer messages are usually called **packets**, giving an idea that it is a small pack of information to be delivered with some service.
- The network layer is also responsible for...
  - **Congestion detection**: Detect that the sender is sending too much data for the network to transfer, and take action to inform it to stop.
  - **Segmentation and assembly**: When the large packet has to pass through a link with a small maximum frame size, the packet is chopped into smaller ones, to be reassembled in the receiving end.

Network(0234B)-1.22

## The transport layer

- The **transport layer** is responsible for communication of **processes**.
- This is done by attaching a port number to each message. The transport layer messages are usually called **segments**.
- The transport layer is also responsible for...
  - **Connection management**: allow on-demand establishment and removal of connections, making sure they won't interfere one another.
  - Turning packet streams into **error-free byte stream**, to conform to the read/write abstraction of Unix file descriptors.
    - This is why we call it segment: each transport message is a small part of a single long stream of byte.
  - **Flow control**: avoid overflowing buffers of the receiver.
  - **Congestion handling**: react to the discovery of congestion.

Network(0234B)-1.23

### The session and presentation layer

- The **session layer** is responsible for a few services that a few application would need: **check-pointing**, so that a crashed connection can recover at a certain point; and **dialog management**, to make sure at each time only one end of a connection will talk.  
They are rarely used and we won't go into it further.
- The **presentation layer** (or **representation layer**) is responsible for the interpretation of data within a message, e.g., to convert between little-endian and big-endian.  
Most underlying network communication is done with big-endian, most significant bit sent first. We call this the "network byte order".
- When the presentation layer is used, it **completely changes the abstraction** of communication. We no longer care about "sending network messages", but instead care about "calling **remote procedures**" or "invoking **remote objects**".  
We will look at one example during a future tutorial.

Network(0234B)-1.24

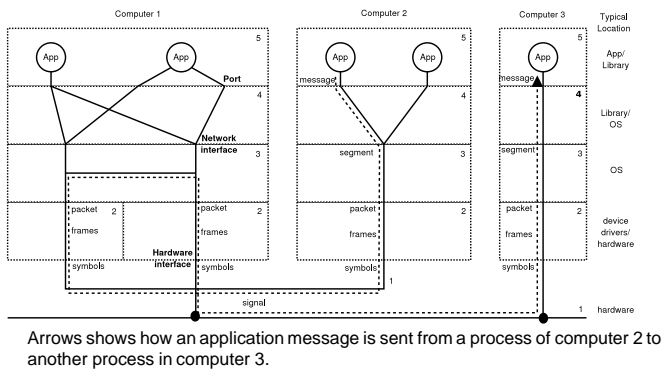
### The application layer

- The **application layer** is the what individual application work on.
- It sometimes rely the service of the presentation layer, but most of the time it is skipped and use only the transport layer primitives.
- The application layer is **highly application specific** and does not fit into our main objective on "how to make easy socket communication possible". We only look at it briefly in the next lecture.

Network(0234B)-1.25

### Schematics

With these functionalities defined, the network works this way:



Network(0234B)-1.26

### PDU's and SDU's

- When talking about a layer-*n*, we will need to refer to the message that the **upper** layer sends it.
- It is troublesome to always switch among words like "messages", "segments", "packets" and "frames". We use the following convention:
  - The message is received by layer-*n* through the *n*-SAP.
  - The message received by layer-*n* is called a *n*-**Service Data Unit**, or *n*-**SDU**. ("Service" here means "External".)
  - The message generated by layer-*n* is called a *n*-**Protocol Data Unit**, or *n*-**PDU**. ("Protocol" here means "Internal").
- So the physical signal is a 1-PDU, a frame is a 2-PDU or 1-SDU, a packet is a 3-PDU or 2-SDU, and a segment is a 4-PDU or 3-SDU. When focused on network layer (layer 3), SDU = segment, PDU = packet. A TCP port is a layer-4 SAP.

Network(0234B)-1.27

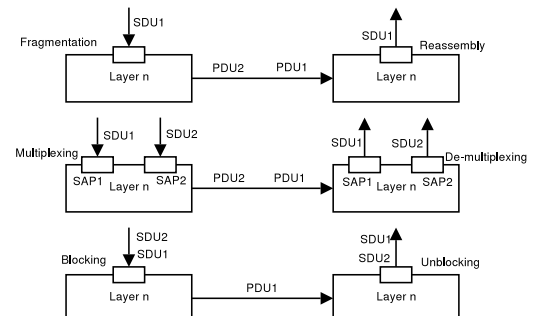
### General operation across layers

A few simple services are needed in many layers. We introduce the terminology here, and we get to the actual headers needed later.

- When a SDU is too large to put into an PDU, the SDU is broken into many PDUs. This is called **segmentation**, and the receiving end will combine them again, which is called **reassembly**.
- If there are many SAPs for a service, the SDU is tagged with an identification of the upper-layer when making up a PDU. This is called **multiplexing**, and the other end performs **demultiplexing** to give the SDU to the appropriate SAP on the receiving side.
- When a SDU is too small and would be inefficient to put into an individual PDU, it may be combined with other SDUs to form a single PDU. This is called **blocking**, and the other end performs **unblocking** to recover the smaller SDUs.

Network(0234B)-1.28

### Schematics



Network(0234B)-1.29