CS4516 HELP Session 1

Global Employee Location Server

Jeff Zhou
jeffz@cs.wpi.edu
3/19/2011

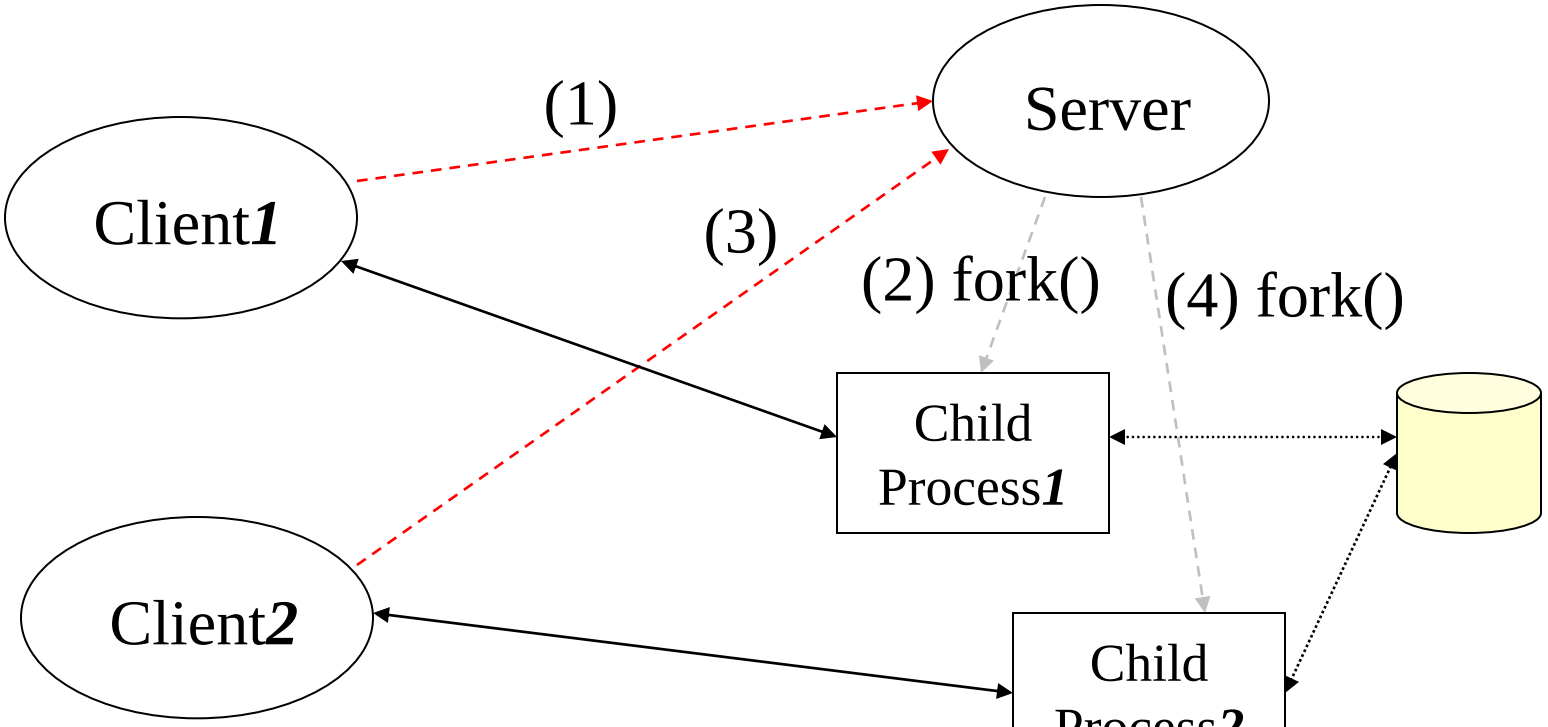
Modified based on CS4514 B05 slides

Description

- **Objective:**

  To implement a simple concurrent server that has four emulated network protocol stacks.
  - **Application layer: Messages**
  - **Network layer: Messages←→Packets**
  - **Datalink layer: Packets ←→ Frames and Selective Repeat Sliding Window protocol**
  - **Physical layer: TCP connection**

System Overview

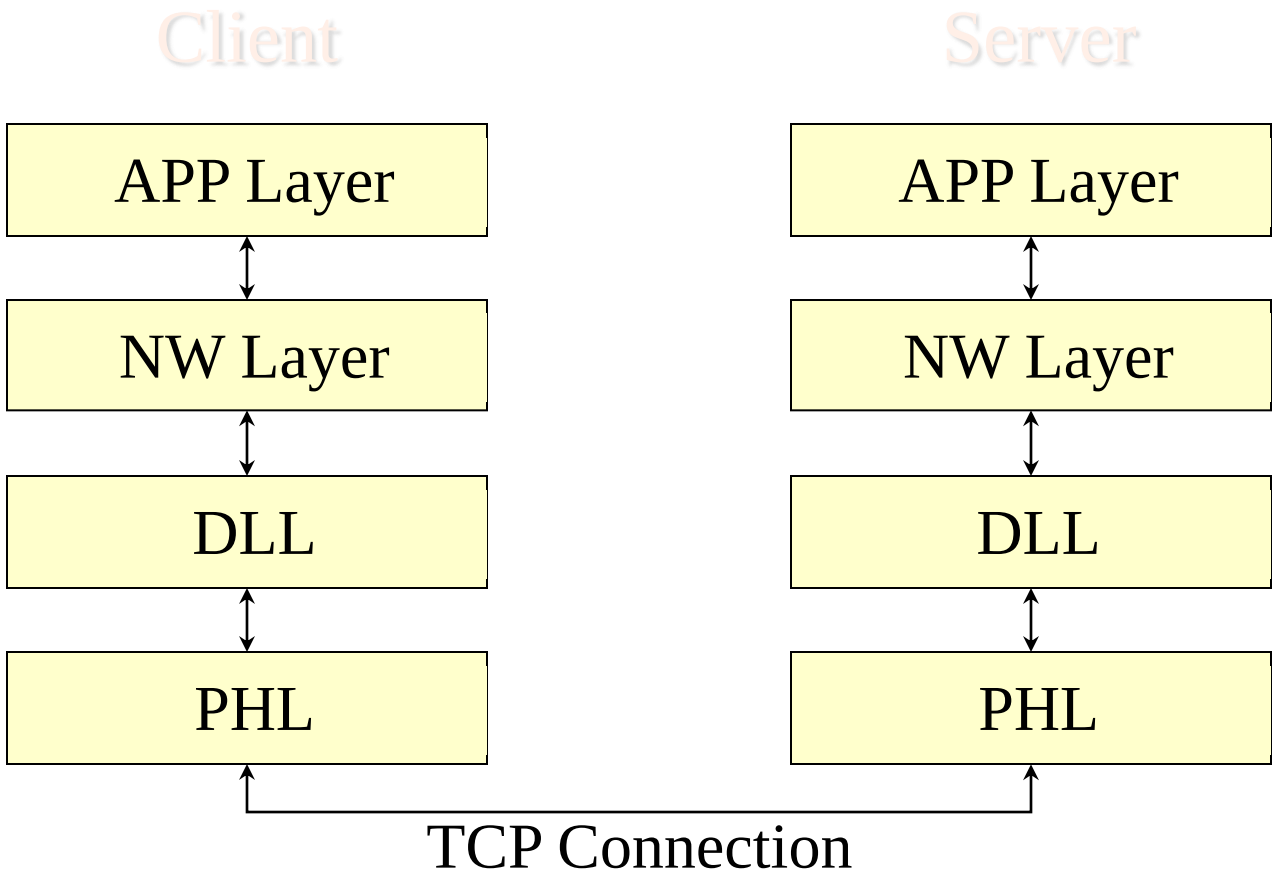


You can either use multiprocesses (fork())

or multithreading (pthread)

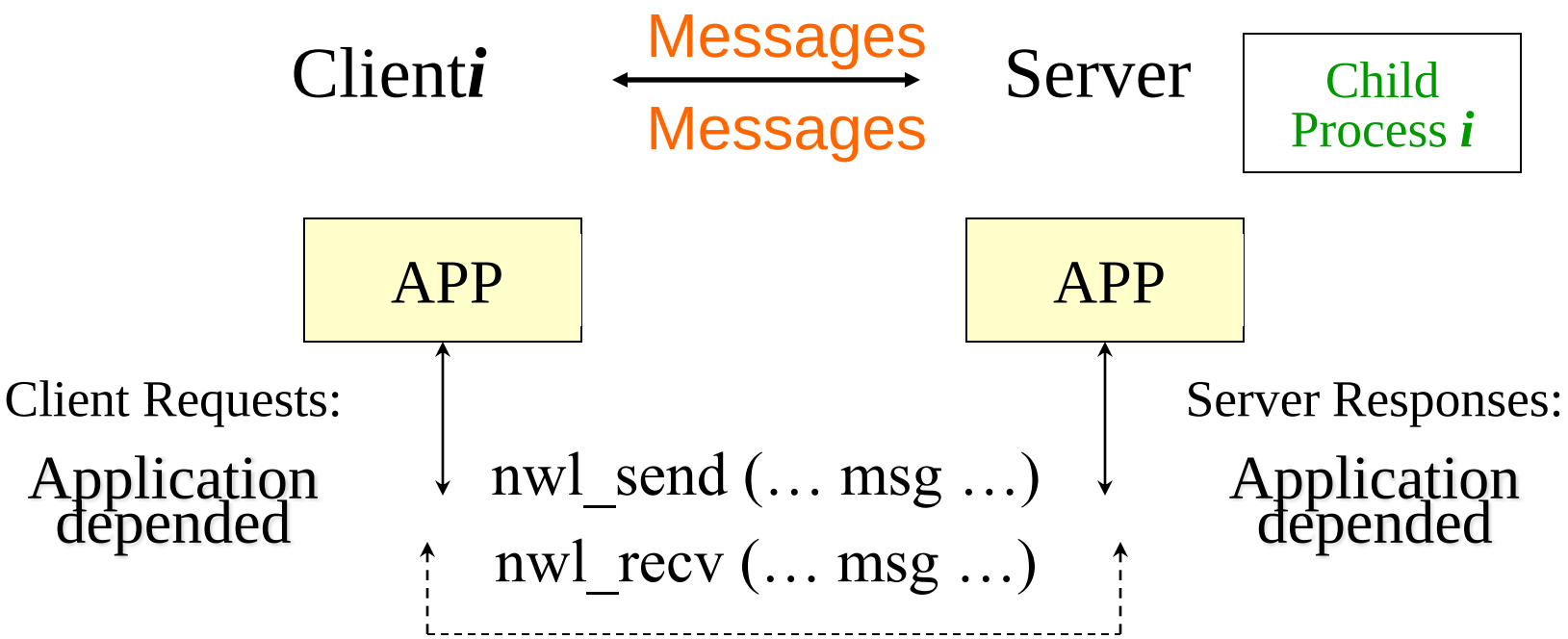
You need to implement concurrent access to the database.

System Framework



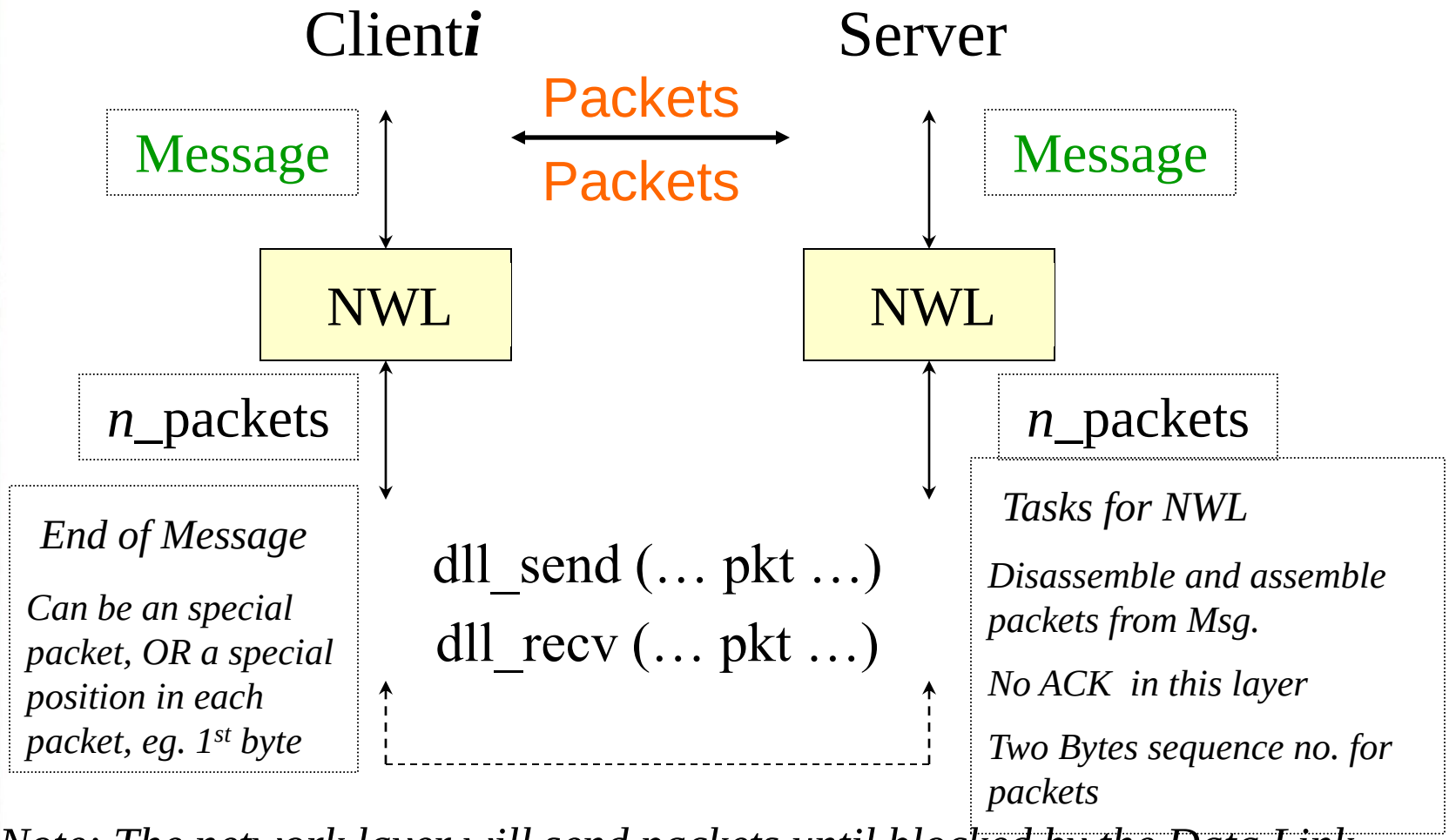
Four Layer stacks

How the System Works: Layer by Layer
Application Layer



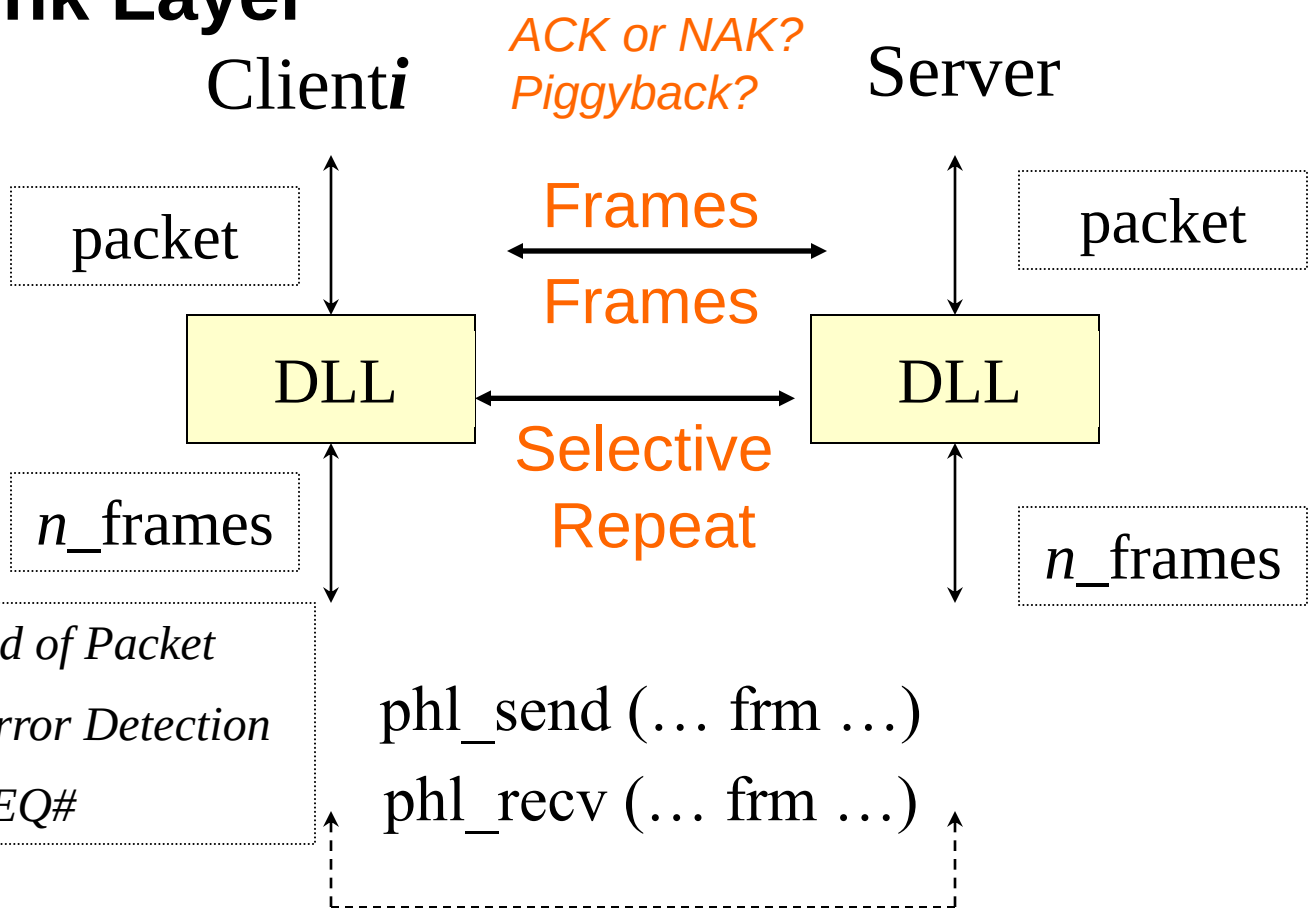
- *At least 5 operations, at least one long operation in each direction that will easily test your sliding window*
- *Message that specifies the client type (employee or HR)*
- *Input and query a photo, select all users in a country*

How the System Works: Layer by Layer
Network Layer



Note: The network layer will send packets until blocked by the Data Link Layer. But HOW?

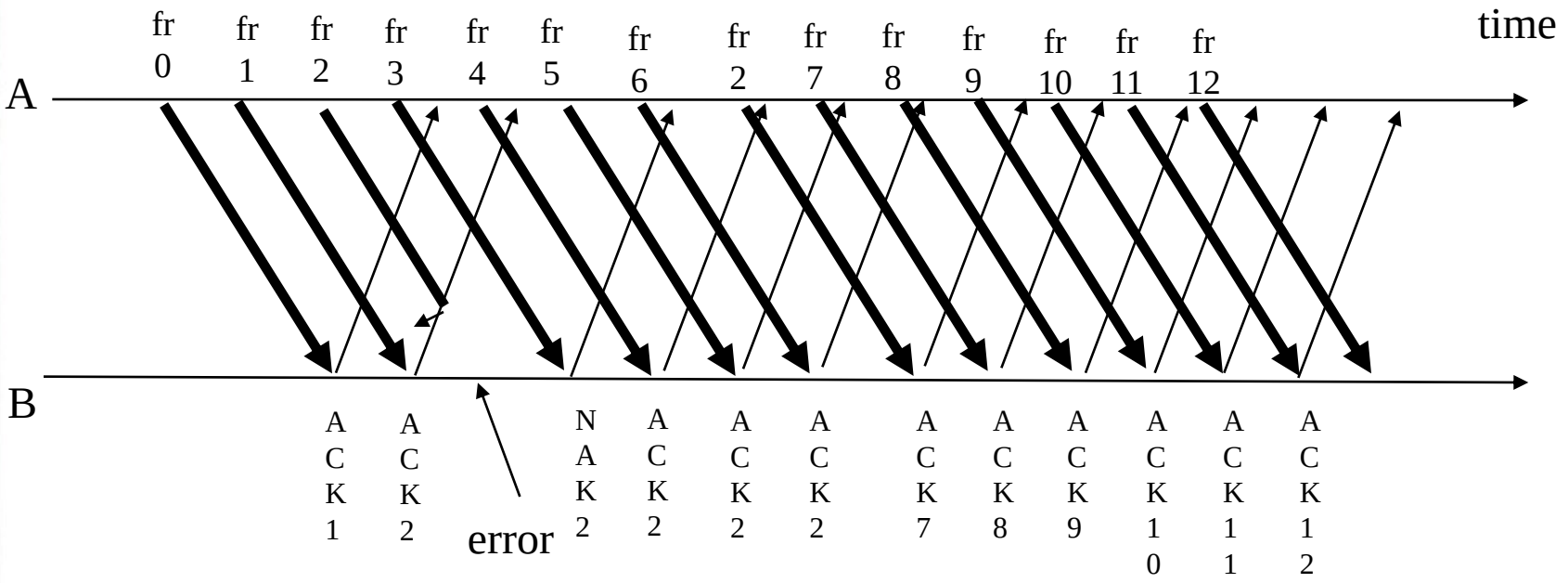
How the System Works: Layer by Layer
DataLink Layer



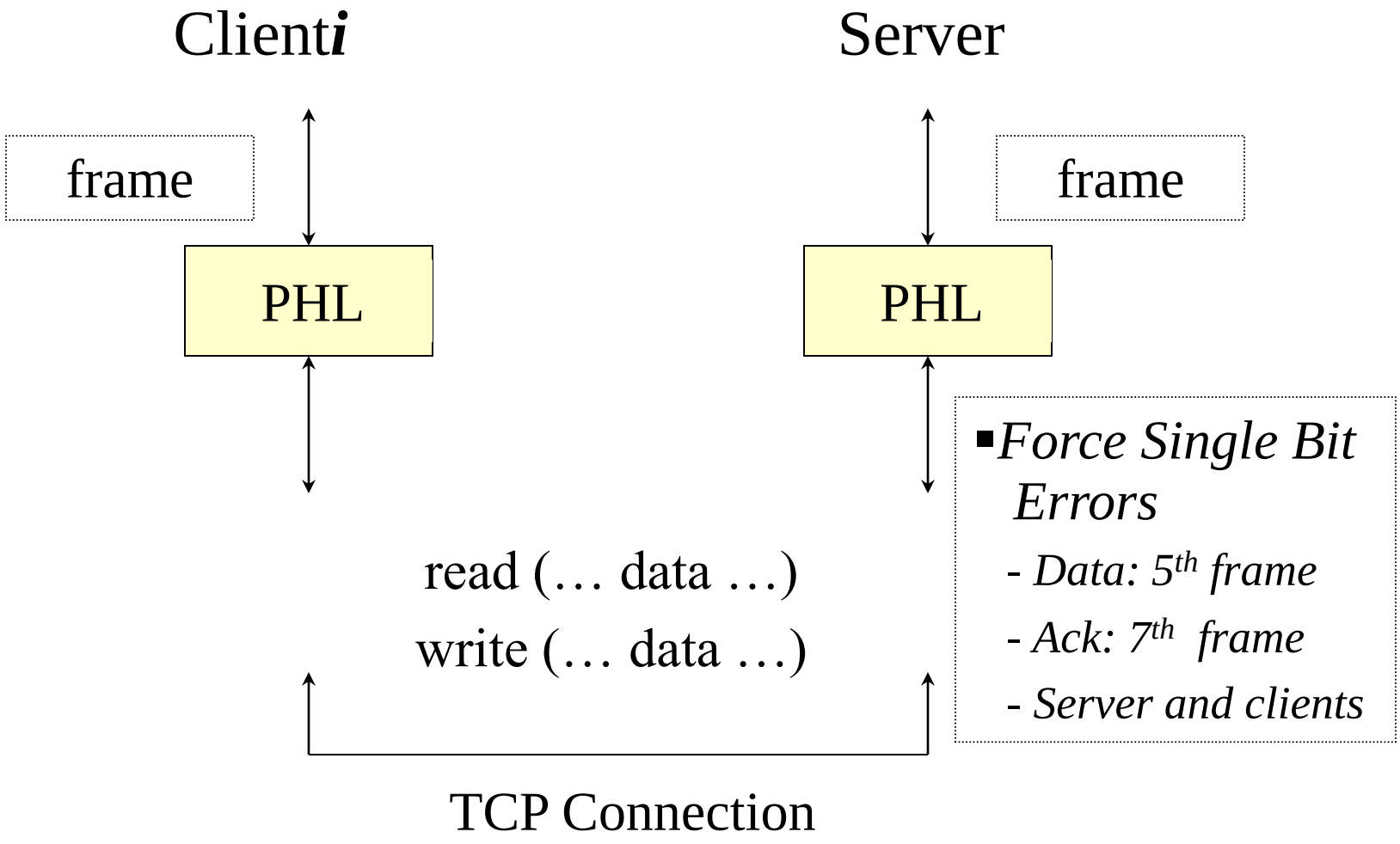
Maximum packet size 128 bytes

Sliding window size >=4

Selective Repeat

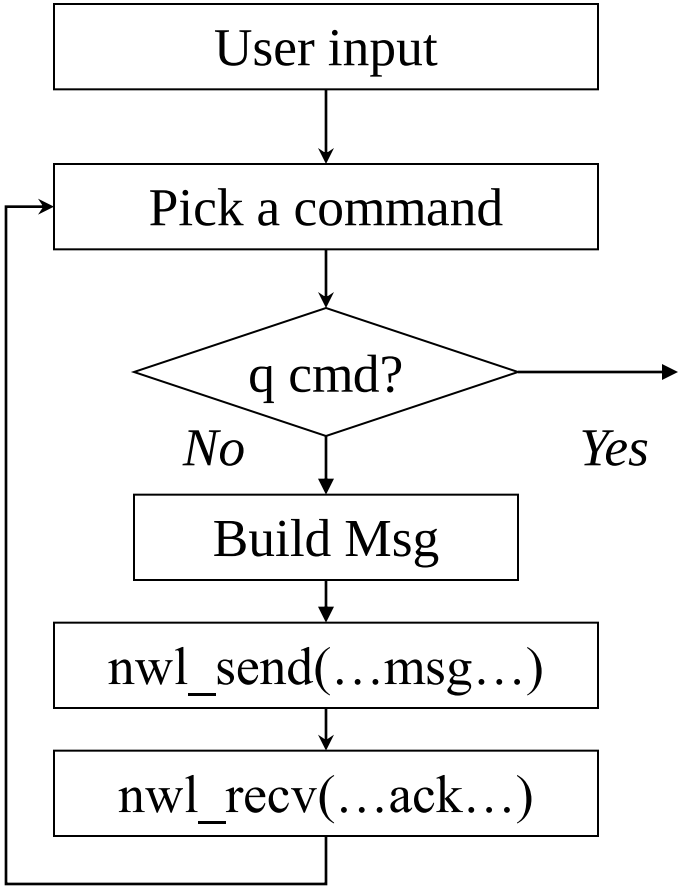


How the System Works: Layer by Layer
Physical Layer

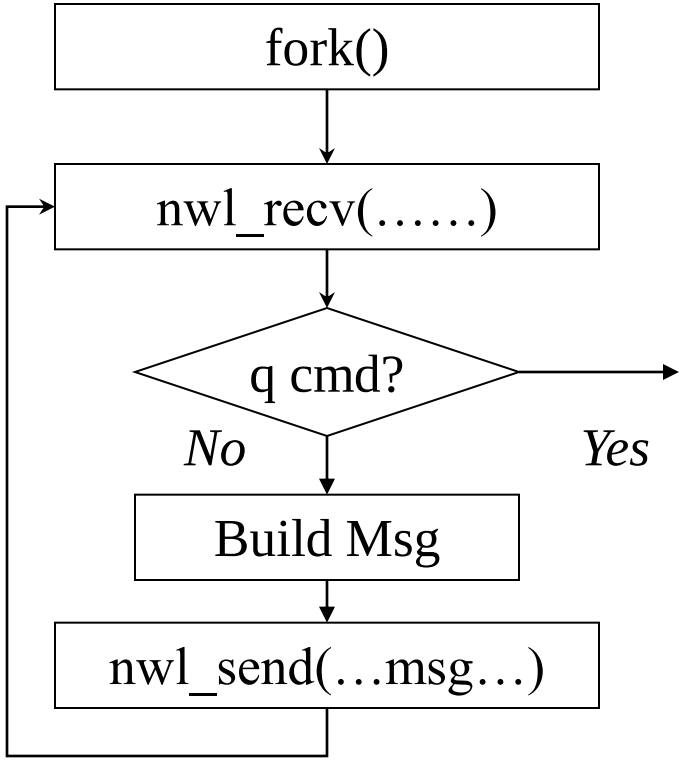


How the Functions Work: Layer by Layer

client APP

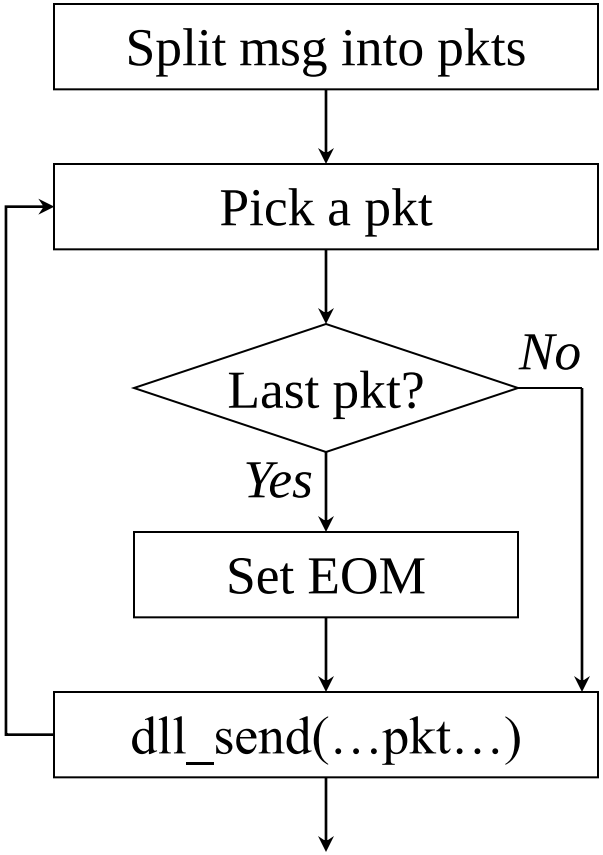


server child process APP

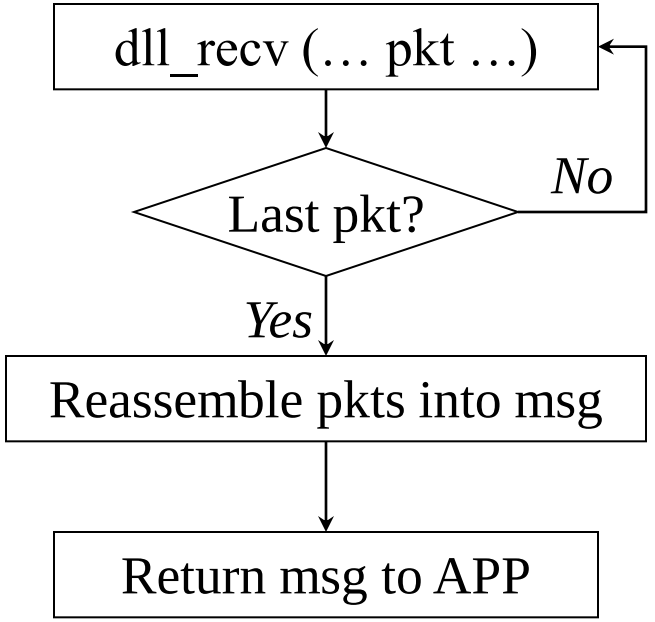


How the Functions Work: Layer by Layer

nwl_send (… msg …)



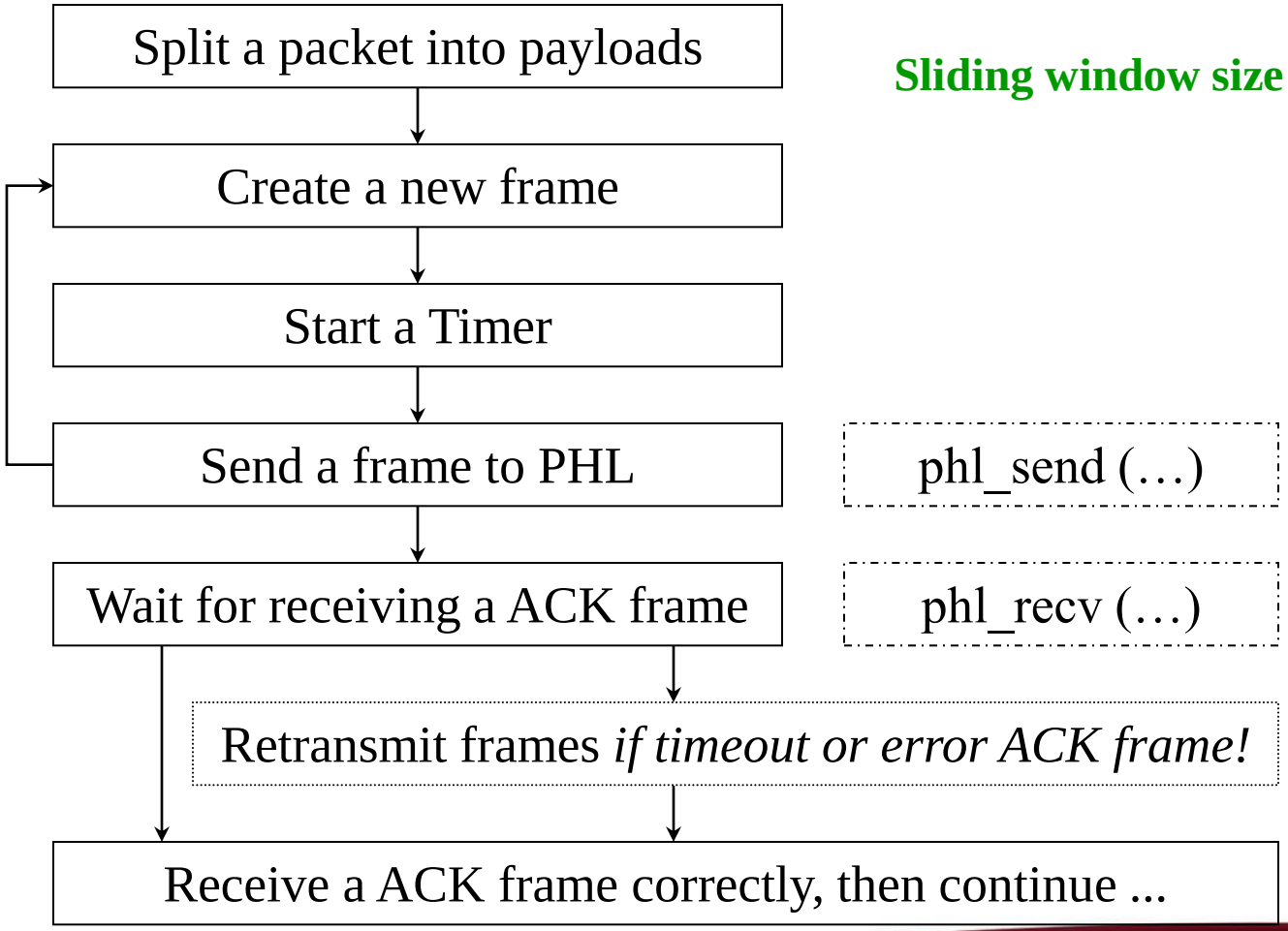
nwl_recv (… msg …)



Note: you need have a mechanism to decide the last packet in a message (EOM). The diagram here offers only a reference.

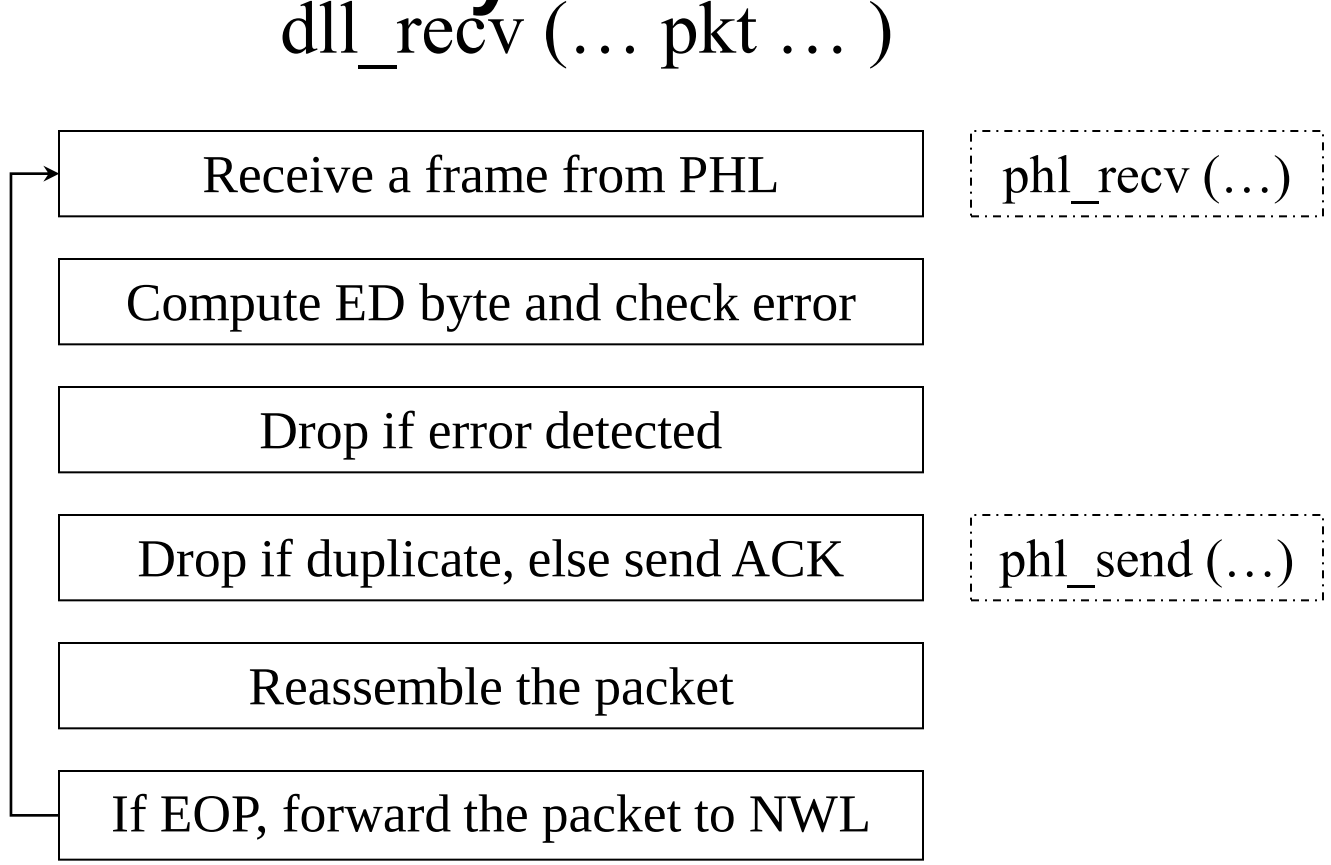
How the Functions Work: Layer by Layer

dll_send (… pkt …)



Sliding window size =1

How the Functions Work: Layer by Layer



Question: When is the correct time to send *NAK or ACK*?

Not after ED drop, but on receiving next frame or dup frame.

Debugging output

- Output that helps debugging the program
- Can be easily turned on/off by a macro
- The following statistics must be calculated and reported:
  - The total number of data frames transmitted successfully
  - The total number of data frames received successfully
  - The total number of data frames received with errors
  - The total number of ACK's transmitted successfully
  - The total number of ACK's received successfully
  - The total number of ACK's received with errors
  - The total number of duplicate frames received.

Project Tips-1

- **Sliding Window Protocol: Selective repeat (N>=4)**
  - **Try to implement windows size 1 first**
  - **Then implement N (multiple timers)**
- **Follow the example in the book (protocol 6)**
- **How to terminate client process:**
  - **When the client gets the response to the quit message**
  - **A "clean" way to terminate the server child process/thread? Use wait()/pthread_join()!**

Project Tips-2

- **Simulate multiple timer in software**
  - **Approach I**
    - **Using link list or array**
    - **pp.223 on textbook()**
    - **Need signal()**
  - **Approach II**
    - **Using link list or array**
    - **Update the *struct timeval* for next select() call**

Project Tip3

- ***How could the NWL Keep sending packets until blocked by the Data Link Layer ?***

  Our suggestion is that you could use pipe to implement it: NWL keeps writing packets to the pipe until the pipe is full.
- **A simple code of pipe could be found at**

  http://web.umr.edu/~ercal/284/PipeExamples/Examples.html
- **Pipe is more like a socket between local processes.**

Concurrent TCP Server Example (fork)

```
pid_t pid;
int listenfd, connfd;


/* 1. create a socket socket() */
if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 )
err_quit("build server socket error\n", -1);
/* 2. fill in sockaddr_in{ } with server's well-known port */
…
/* 3. bind socket to a sockaddr_in structure bind() */
bind (listenfd, ...);
/* 4. specify the backlog of incoming connection requests  listen() */
listen (listenfd, LISTENQ);
while(1){
    connfd = accept(listenfd, ... );  /* probably blocks */
    if(( pid = fork()) == 0){
        close(listenfd);  /* child closes listening socket */
        doit(connfd);     /* process the request */
        close(connfd);    /* done with this client */
        exit(0);
    }
    close(connfd);    /* parent closes connected socket */
}
```

Select()

- Why select()? (recv() and send() at the same time)

int select(int nfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, struct timeval *timeout);

```
int main(void)
{
    fd_set rfds;
    struct timeval tv;
    int retval;

    /* Watch stdin (fd 0) to see when it has input. */
    FD_ZERO(&rfds);
    FD_SET(0, &rfds);

    /* Wait up to five seconds. */
    tv.tv_sec = 5;
    tv.tv_usec = 0;

    retval = select(1, &rfds, NULL, NULL, &tv);
        /* Don't rely on the value of tv now! */

    if (retval == -1)
            perror("select()");
        else if (retval)
            printf("Data is available now.\n");
            /* FD_ISSET(0, &rfds) will be true. */
        else // retval == 0 here
            printf("No data within five seconds.\n");

        exit(EXIT_SUCCESS);

}
```

Questions?