CS4514 HELP Session 3

Concurrent Server Using Go-Back-N

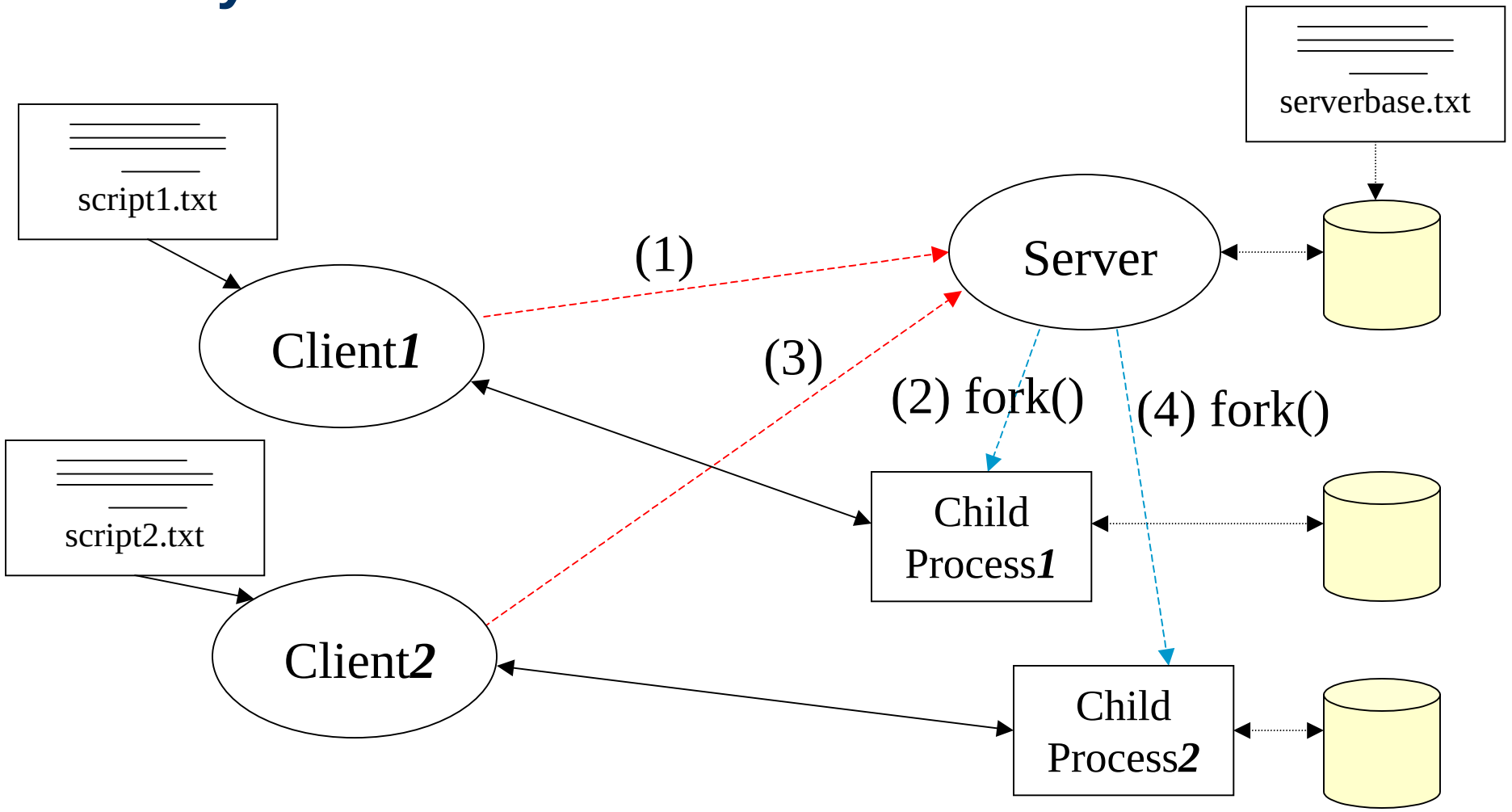
Song Wang

12/08/2003

Description

- You are supposed to implement a simple concurrent server and client having four emulated network protocol stack.
  - Application layer: Read and execute commands
  - Network layer: Message←→Packet (send&recv)
  - Datalink layer: Packet ←→ Frame and Go-Back-N sliding window protocol
  - Physical layer: TCP connection.
- Your programs should compile and work on any one of ccc.WPI.EDU.

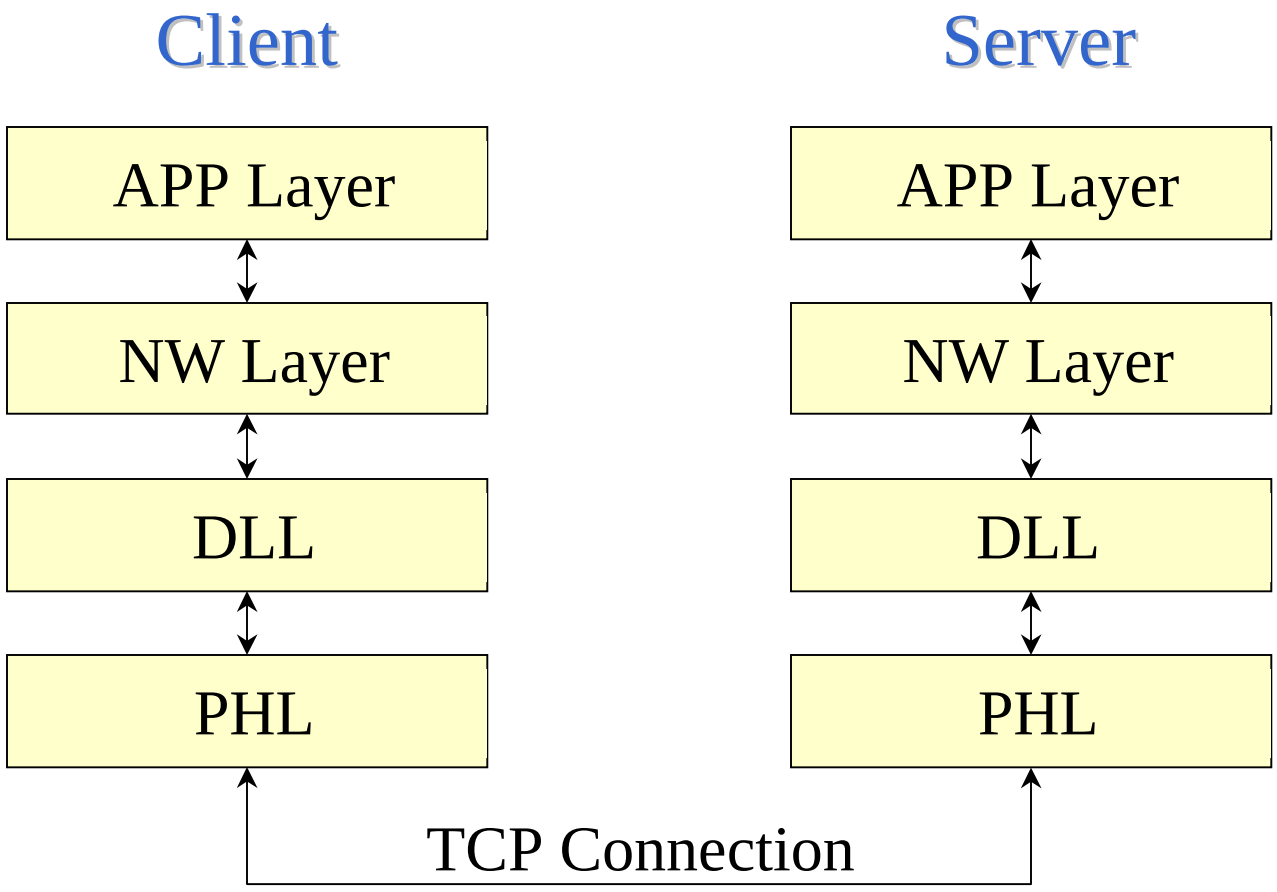
System Overview



Note: each child process keeps a separate copy of the DB.

we do not keep data consistency for the serverbase

System Framework



Concurrent Server (fork())

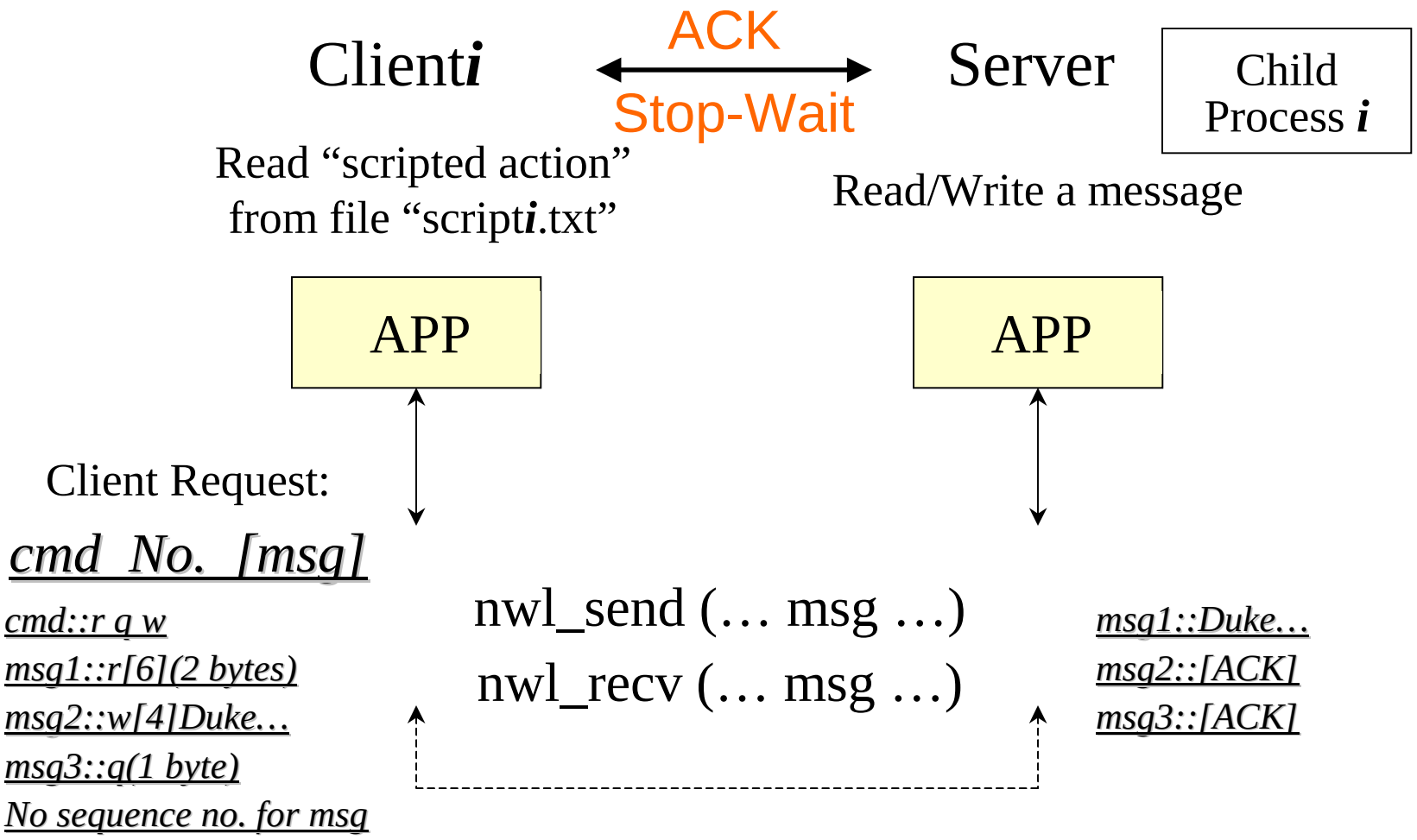
- fork() will make a child process with memo ry copy.
  - The initial serverbase will be copied to each c hild process.
  - fork() will return child pid in parent process an d 0 in child process.
  - Remember to close socket after using.

Concurrent TCP Server Example

```
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 */
}
```

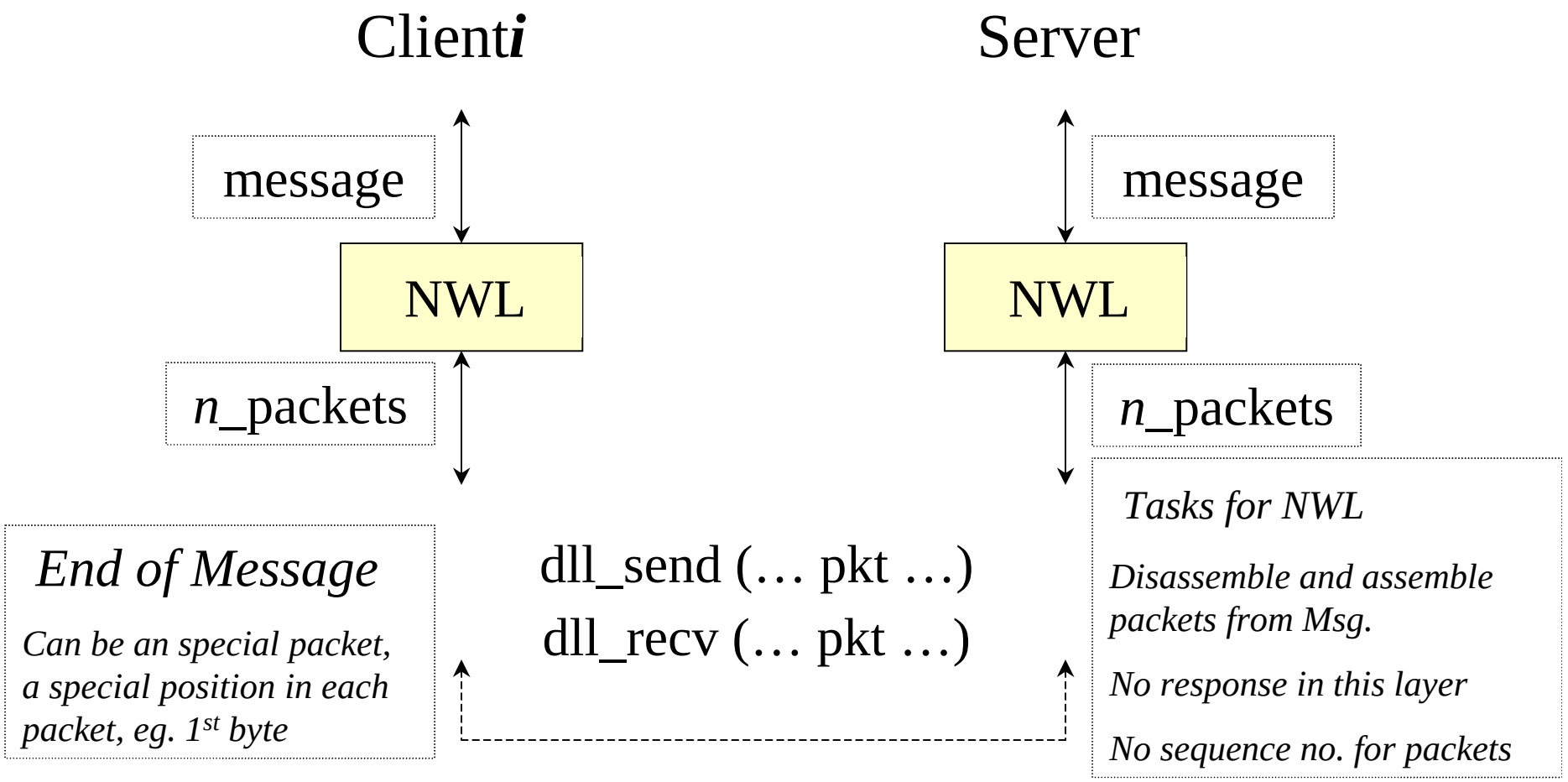
How the System Works: Layer by Layer



Note: The max_size of a message is 290 bytes

The number referring to tuple position is 1 to 14

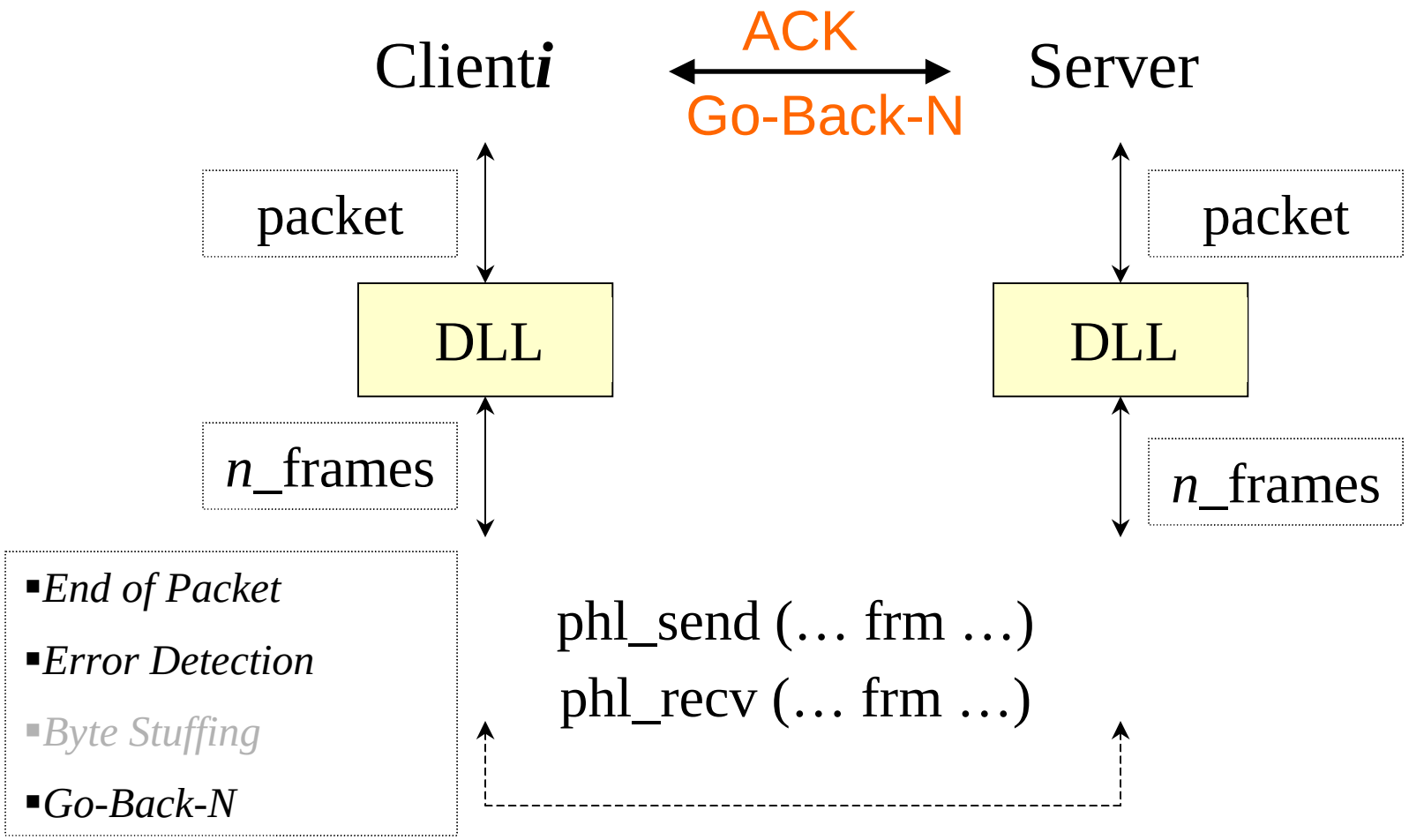
How the System Works: Layer by Layer



Note: The max_size of a packet is 72 bytes

The network layer will send packets until blocked by the Data Link Layer

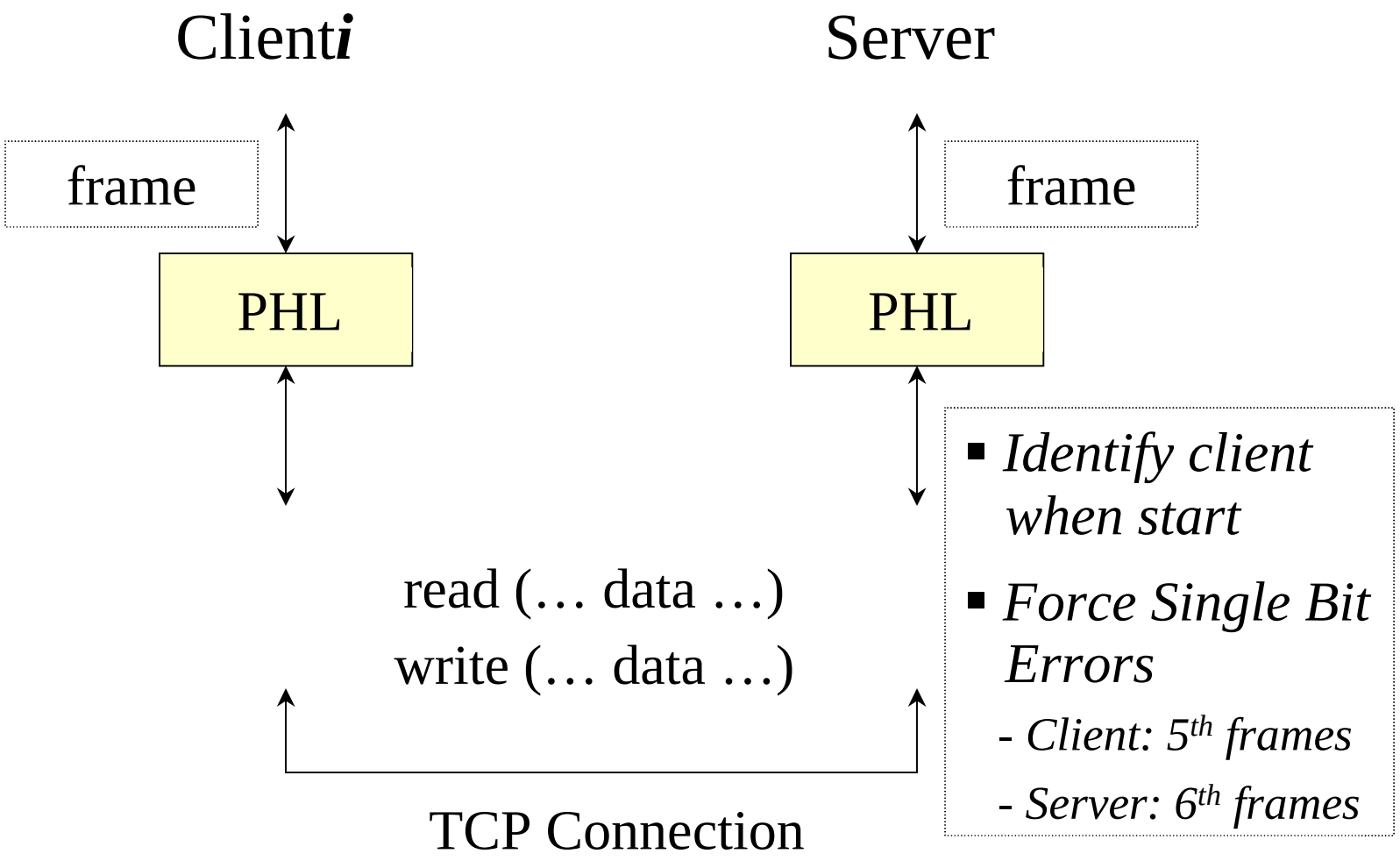
How the System Works: Layer by Layer



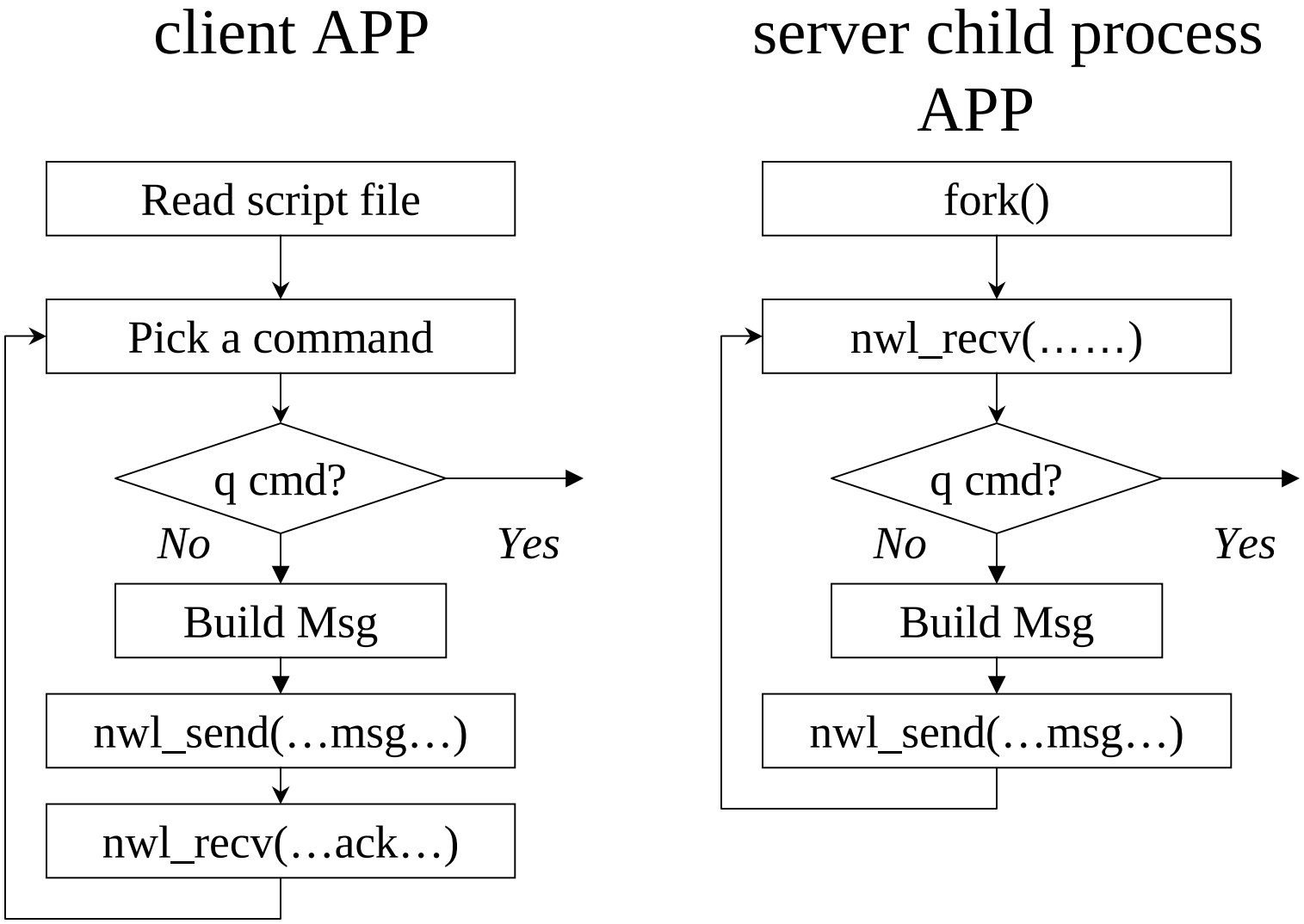
Note: The max_size of a frame payload is 48 bytes

Sliding window size >=3

How the System Works: Layer by Layer

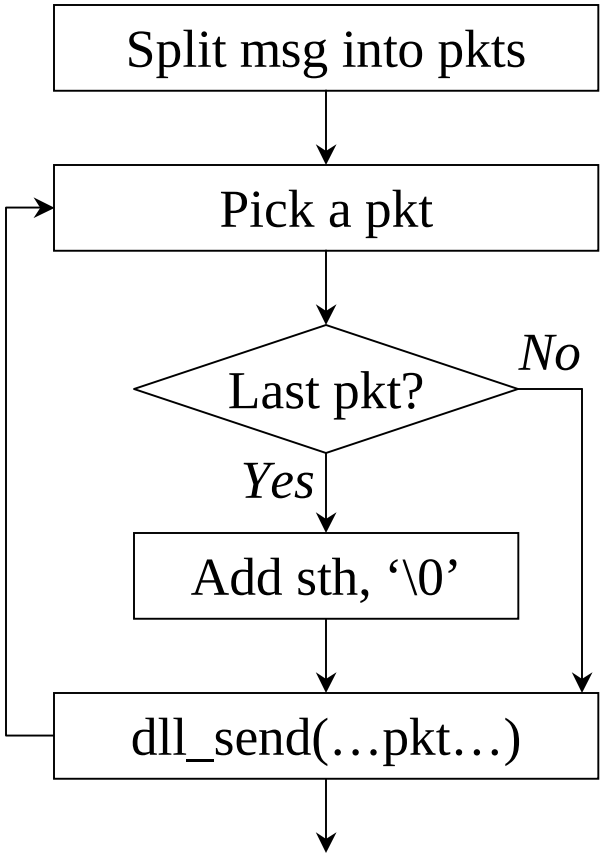


How the Functions Work: Layer by Layer

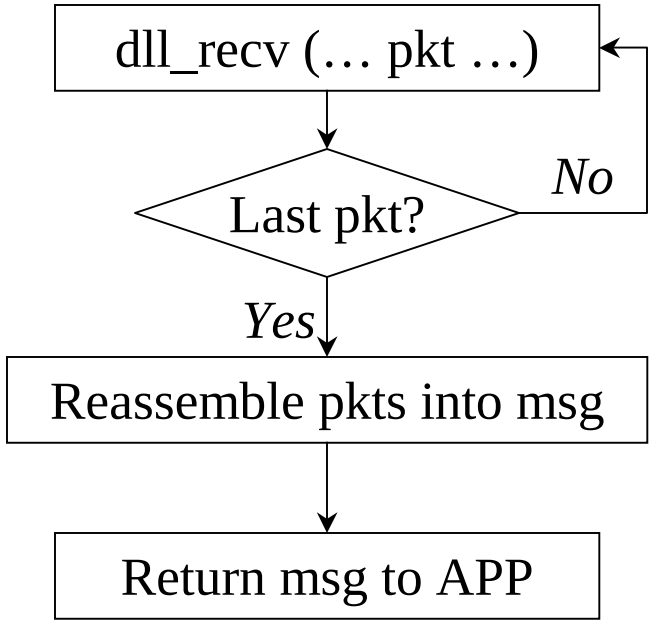


How the Functions Work: Layer by Layer

nwl_send (… msg …)

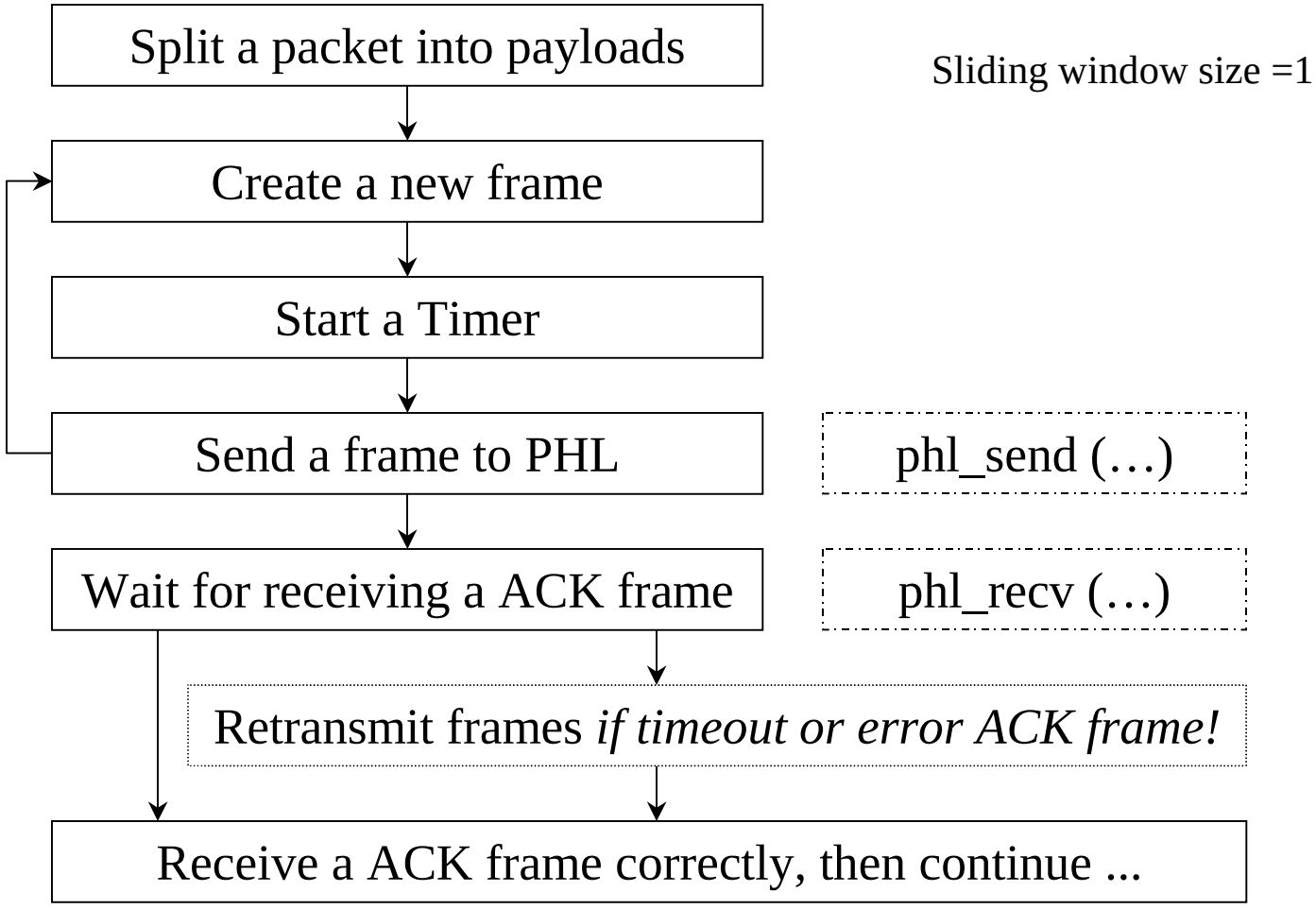


nwl_recv (… msg …)

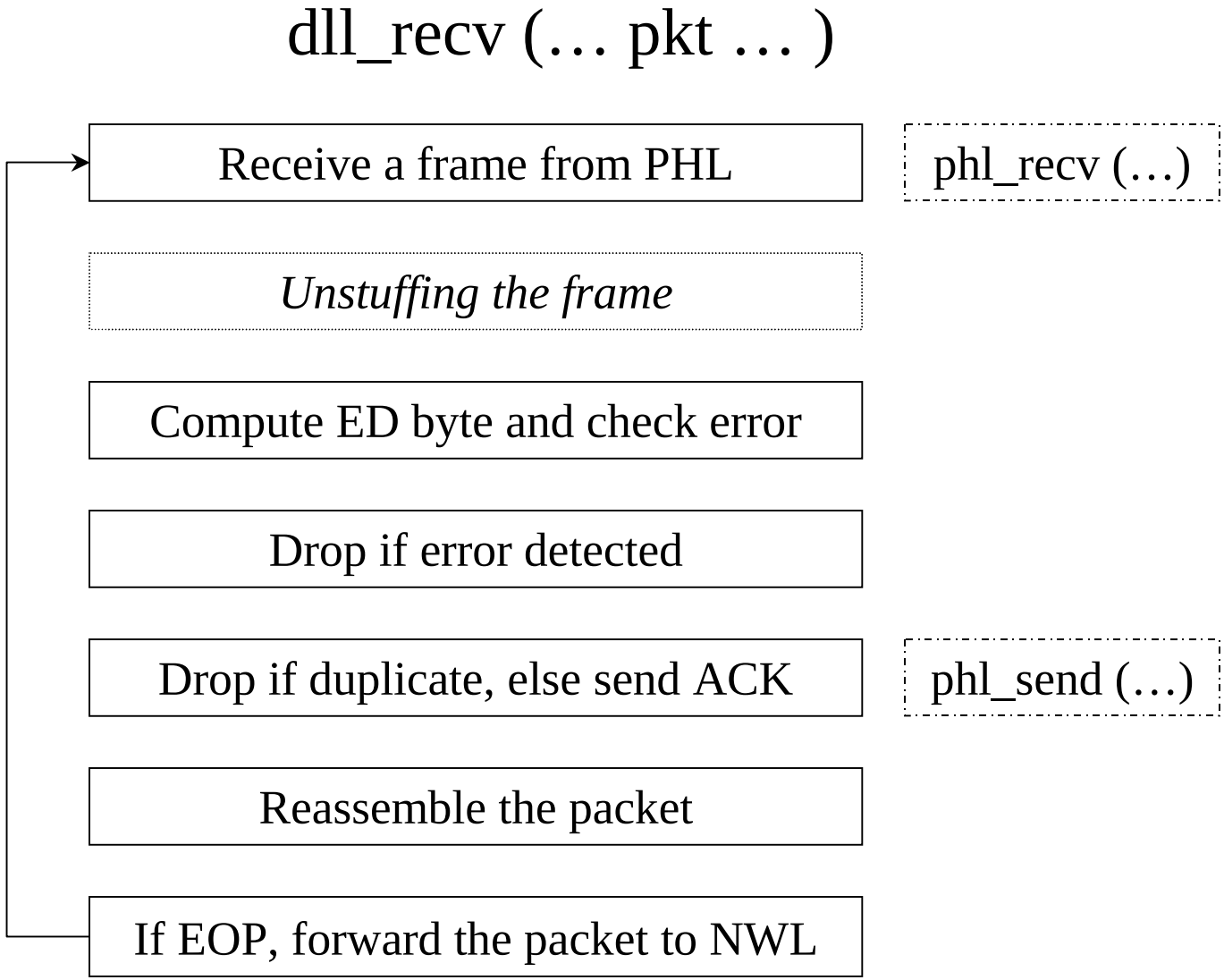


How the Functions Work: Layer by Layer

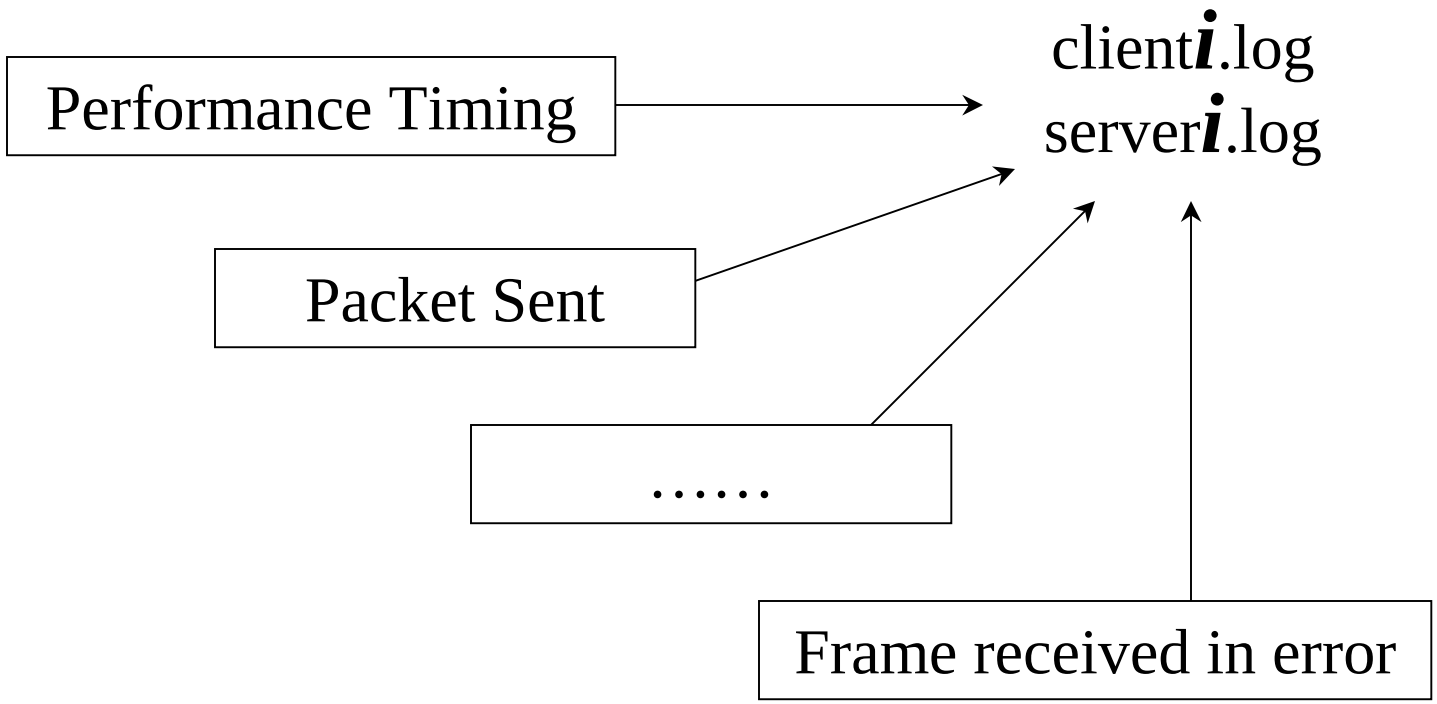
dll_send (… pkt …)



How the Functions Work: Layer by Layer



Log Significant Events



Project Tips

- Sliding Window Protocol: Go-Back-N (N>3)
  - Try to implement Go-Back-1 first
  - Then implement Go-Back-N (multiple timers)
- Maybe easier to merge PHL and DLL
- How to terminate client process:
  - When the client gets the response to the quit message
  - A “clean” way to terminate the server child process?

Relative Timer Example

```
/* example for start_timer, stop_timer, send_packet */
/* you SHOULD modify this to work for project 3, this is just a TIMER EXAMPLE */
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/timers.h>
#include <sys/select.h>
#include <sys/types.h>
#include <errno.h>
#define TIMER_RELATIVE 0
#define MAX_SEQ 3
extern int errno;
typedef unsigned int seq_nr;
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
timer_t timer_id[MAX_SEQ];
```

```
void timeout() {
   printf("time out!\n");
}
void start_timer(seq_nr frame_nr) {
   struct itimerspec time_value;
   signal(SIGALRM, timeout);
   time_value.it_value.tv_sec = 1;  /* timeout value */
   time_value.it_value.tv_nsec = 0;
   time_value.it_interval.tv_sec = 0;  /* timer goes off just once */
   time_value.it_interval.tv_nsec = 0;
   timer_create(CLOCK_REALTIME, NULL, &timer_id[frame_nr]);  /* create timer */
   timer_settime(timer_id[frame_nr], TIMER_RELATIVE, &time_value, NULL);  /* set timer */
}
void stop_timer(seq_nr ack_expected) {
   timer_delete(timer_id[ack_expected]);
}
void send_packet(packet *p) {
   fd_set readfds;
   int sockfd;
```

```
while(packet hasn't been finished sending) {
    /* send frame if we can */
    while(there's place left in sliding window) {
        /* construct a frame from the packet */
        /* send this frame; start timer; update sliding window size */
    }
    /* check data from physical layer */
    FD_ZERO(&readfds);
    FD_SET(sockfd, &readfds);
    if (select(sockfd+1, &readfds, (fd_set *)NULL, (fd_set *)NULL, (struct timeval*)NULL) < 0) {
        if (errno == EINTR) {  /* receive timeout signal */
            /* timeout handler should have resent all the frames that haven't been acknowledged */
            continue;
        }  else {
            perror("select error");  /* select error */
            exit(1);
        }
    }
```

```
            if (FD_ISSET(sockfd, &readfds)) {  /* a frame come from socket */
            /* read a frame from the socket */
            if (cksum() == FALSE) {  /* error check */
               continue;  /* do nothing, wait for timer time out */
            }
            else {
               /* check to see if this frame is a data or ACK frame, and do corresponding processing */
               continue;
            }
         }
      }
   }
```