CS241 System Programming UDP Communication (IV)

Klara Nahrstedt

Lecture 37

4/21/2006

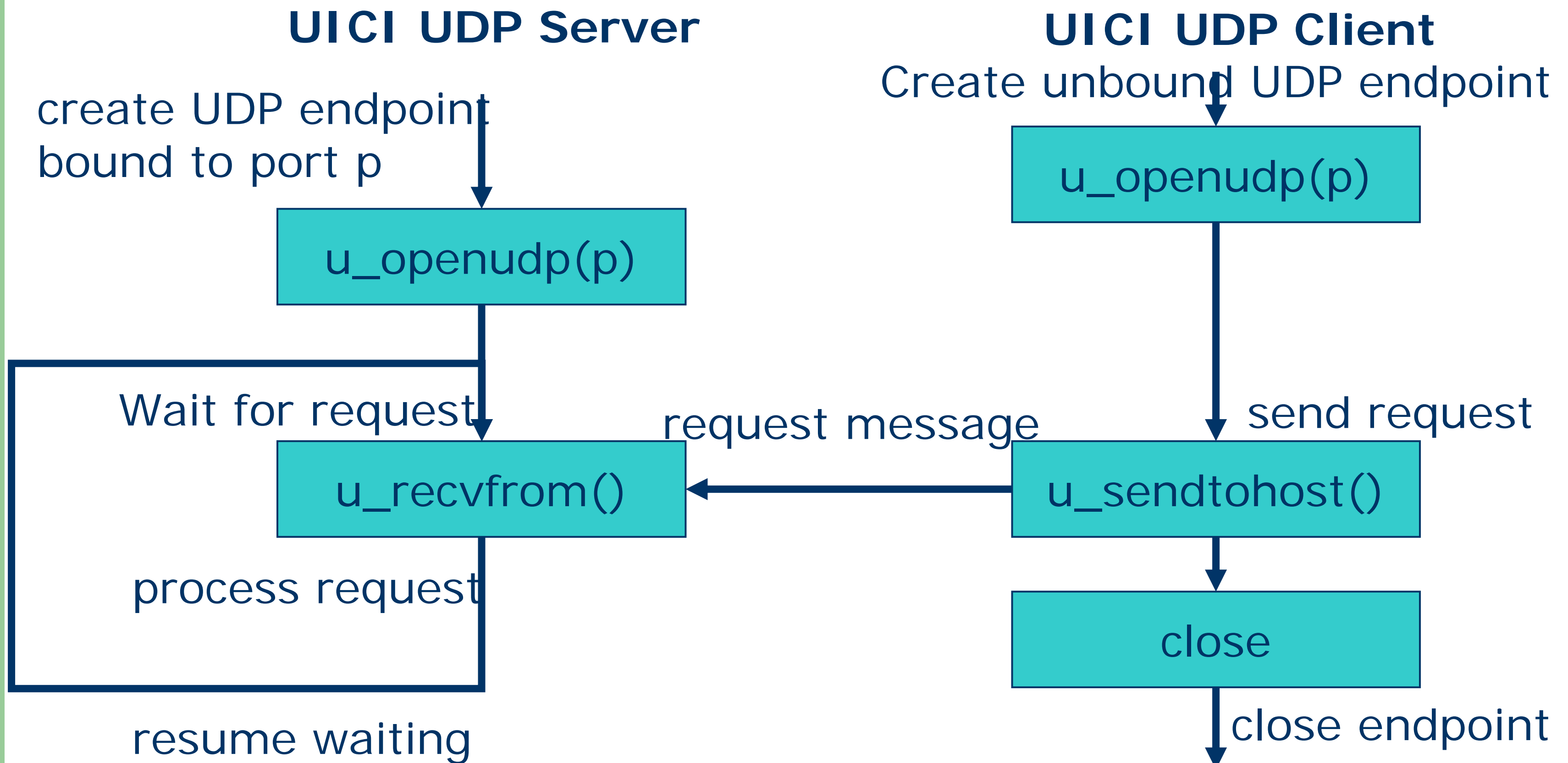
Contents

- Request Reply
- Timed messages
- Acknowledged messages
- Differences between TCP and UDP

Administrative

- Read R&R 20.3-20.6
- Read R&R 20.8
- MP5 is on – deadline May 1
- Quiz 10 – April 21
- Homework 2 – posted April 26 – deadline May 3, midnight
- Quiz 11 – April 28
- Quiz 12 – May 3 – OPTIONAL!!!!! ONLY **TEN BEST RESULTS** FROM QUIZZES WILL COUNT AS SPECIFIED AT THE BEGINNING OF THE SEMESTER

Simple –Request Protocols



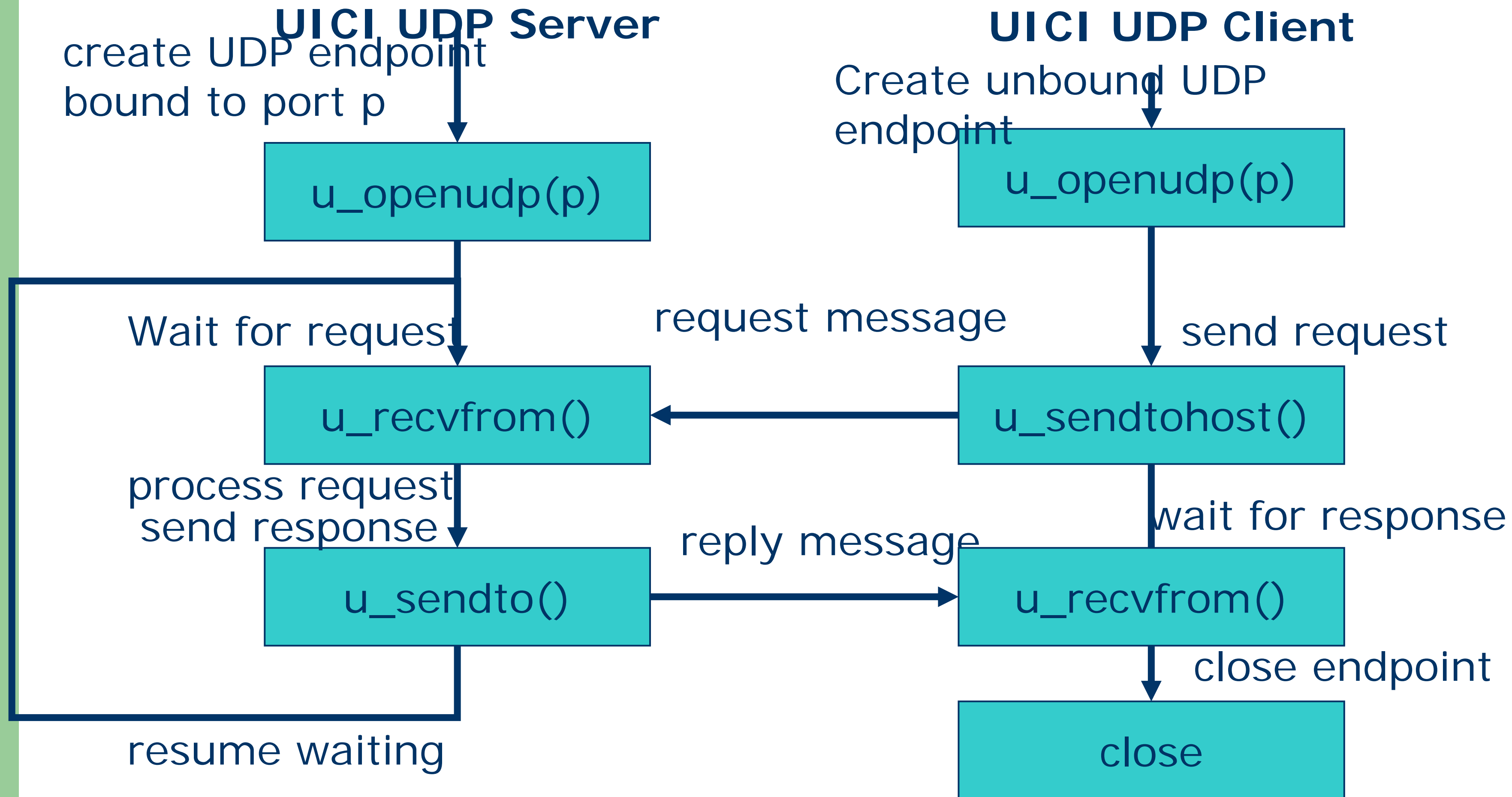
Server_udp.c

```
int main(int argc, char *argv[]) {
  ...
  port = (u_port_t) atoi(argv[1]);  /* create communication endpoint */
  if ((requestfd = u_openudp(port)) == -1) {
    perror("Failed to create UDP endpoint");
    return 1;
  }
  for ( ; ; ) {                              /* process client requests */
    bytesread = u_recvfrom(requestfd, buf, BUFSIZE, &senderinfo);
    if (bytesread < 0) {
      perror("Failed to receive request");
      continue;
    }
    u_gethostinfo(&senderinfo, hostinfo, BUFSIZE);
    if ((r_write(STDOUT_FILENO, hostinfo, strlen(hostinfo)) == -1) ||
       (r_write(STDOUT_FILENO, buf, bytesread) == -1)) {
      perror("Failed to echo reply to standard output");}}}
```

client_udp.c

```
int main(int argc, char *argv[]) {
...
serverport = (u_port_t) atoi(argv[2]);
   if ((requestfd = u_openudp(0)) == -1) {/*create unbnd UDP endpoint*/
      perror("Failed to create UDP endpoint");
      return 1;
   }
   sprintf(request, "[%ld]\n", (long)getpid());      /* create a request */
   rlen = strlen(request);
 /* simple-request protocol sends a request to (server, serverport) */
   byteswritten = u_sendtohost(requestfd, request,
                                             rlen, argv[1], serverport);
   if (byteswritten == -1)
      perror("Failed to send");
   if (r_close(requestfd) == -1 || byteswritten == -1)
      return 1;
   return 0;}
```

Request–Reply Protocols



Client/Server Behavior

- Replies to Client with reply including senderinfo
- Client checks reply to make sure it comes from server and it should check to see that the message contains senderinfo from client

Server request-reply protocol

```
int main(int argc, char *argv[]) {
  ...
    for ( ; ; ) {              /* process client requests and send replies */
    bytesread = u_recvfrom(requestfd, buf, BUFSIZE, &senderinfo);
    if (bytesread == -1) {
      perror("Failed to receive client request");
      continue;
    }
    u_gethostinfo(&senderinfo, hostinfo, BUFSIZE);
    if ((r_write(STDOUT_FILENO, hostinfo, strlen(hostinfo)) == -1) ||
       (r_write(STDOUT_FILENO, buf, bytesread) == -1)) {
      perror("Failed to echo client request to standard output");
    }
    if (u_sendto(requestfd, buf, bytesread, &senderinfo) == -1) {
      perror("Failed to send the reply to the client");
    }
  }
}
```

Client sends process ID and reads reply

```
int main(int argc, char *argv[]) {
...
   bytesread = request_reply(requestfd,  request, strlen(request)+1,
                        argv[1], serverport, reply, BUFSIZE);
   if (bytesread == -1)
      perror("Failed to do request_reply");
   else {
      byteswritten = r_write(STDOUT_FILENO, reply, bytesread);
      if (byteswritten == -1)
         perror("Failed to echo server reply");
   }
   if ((r_close(requestfd)==-1)||(bytesread==-1)||(byteswritten==-1))
      return 1;
   return 0;
}
```

Request-reply –error free delivery

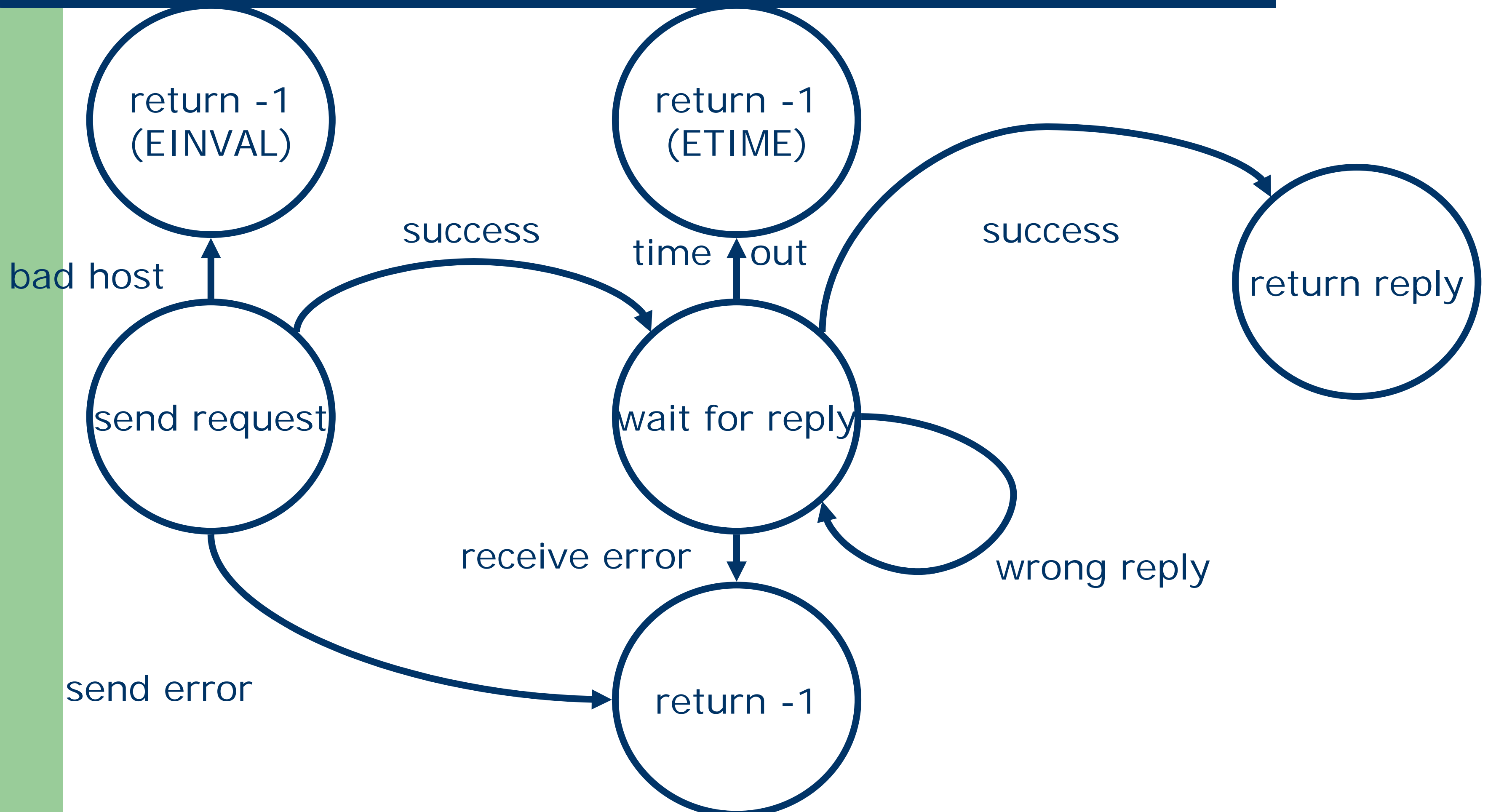
```
#include <sys/types.h>
#include "uiciudp.h"

int request_reply(int requestfd, void* request, int reqlen,
            char* server, int serverport, void *reply, int replen) {
  ssize_t nbytes;
  u_buf_t senderinfo;
                                        /* send the request */
  nbytes = u_sendtohost(requestfd, request, reqlen, server, serverport);
  if (nbytes == -1)
    return (int)nbytes;
                  /* wait for a response, restart if from wrong server */
  while ((nbytes = u_recvfrom(requestfd,reply,replen,&senderinfo))>=0)
    if (u_comparehost(&senderinfo, server, serverport)) /*sender match*/
      break;
  return (int)nbytes;
}
```

Timeouts and Retries

- If either the request or reply message is lost, or if the server crashes, client can hang forever.
- Use select and software timer or use timeout provision on socket

State Diagram of Client for Request-Reply



Simple Timeout

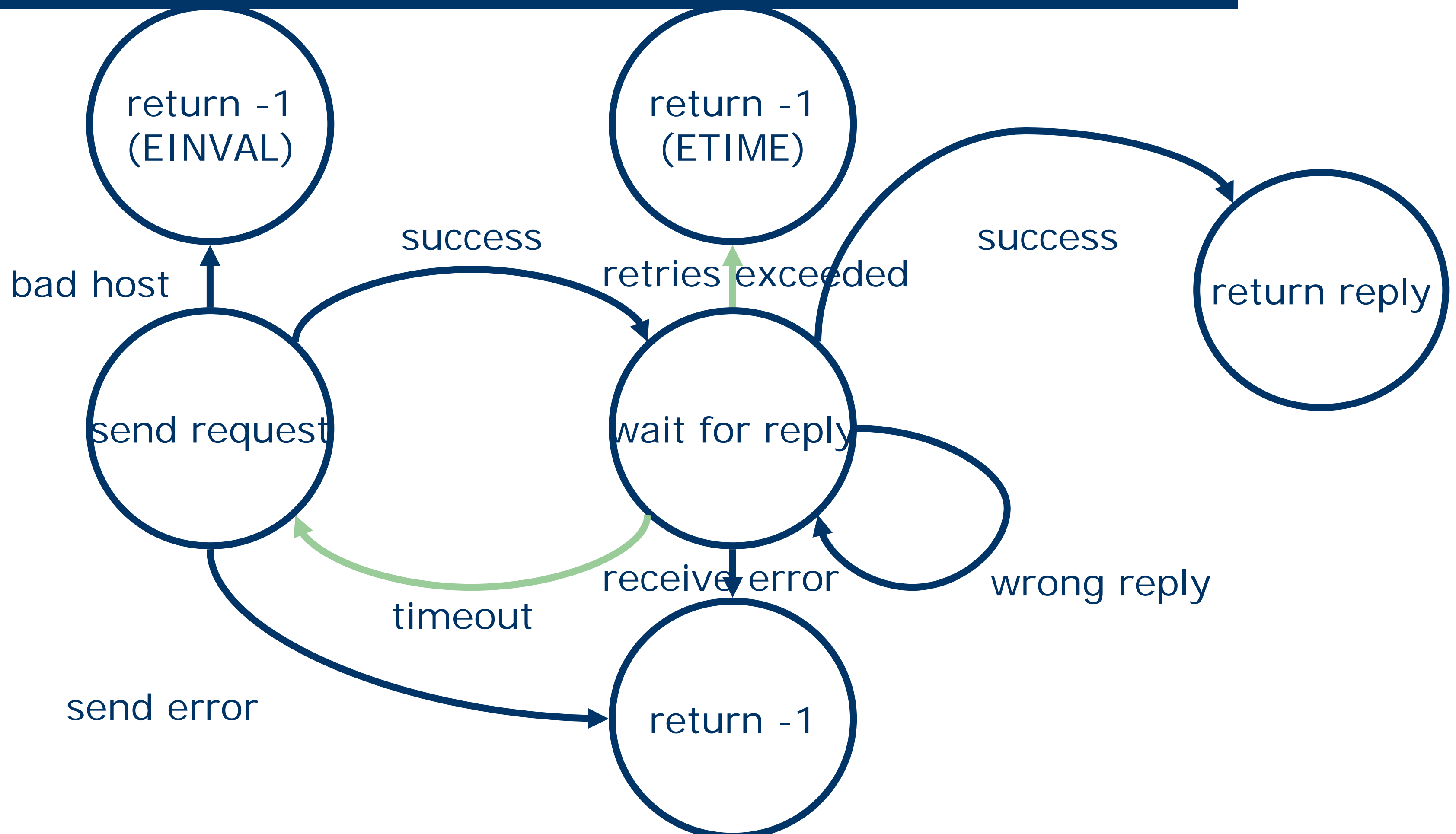
```
int request_reply_timeout(int requestfd, void* request, int reqlen,
                char* server, int serverport, void *reply, int replen,
                double timeout) {
   ssize_t nbytes;
   u_buf_t senderinfo;

                                                    /* send the request */
   nbytes = u_sendtohost(requestfd, request, reqlen, server, serverport);
   if (nbytes == -1)
      return -1;
    /* wait timeout seconds for a response, restart if from wrong server */
   while ((nbytes = u_recvfromtimed(requestfd, reply, replen,
                                    &senderinfo, timeout)) >= 0 &&
             (u_comparehost(&senderinfo, server, serverport) == 0)) ;
   return (int)nbytes;
}
```

Retries

- Client cannot distinguish between server crash and lost message
- Client resets timeout each time it encounters an incorrect responder – allows denial-of-service attack where offenders continually send spurious packets to ports on attacked machine.

State Diagram of Client for Request-Reply with Time-Outs



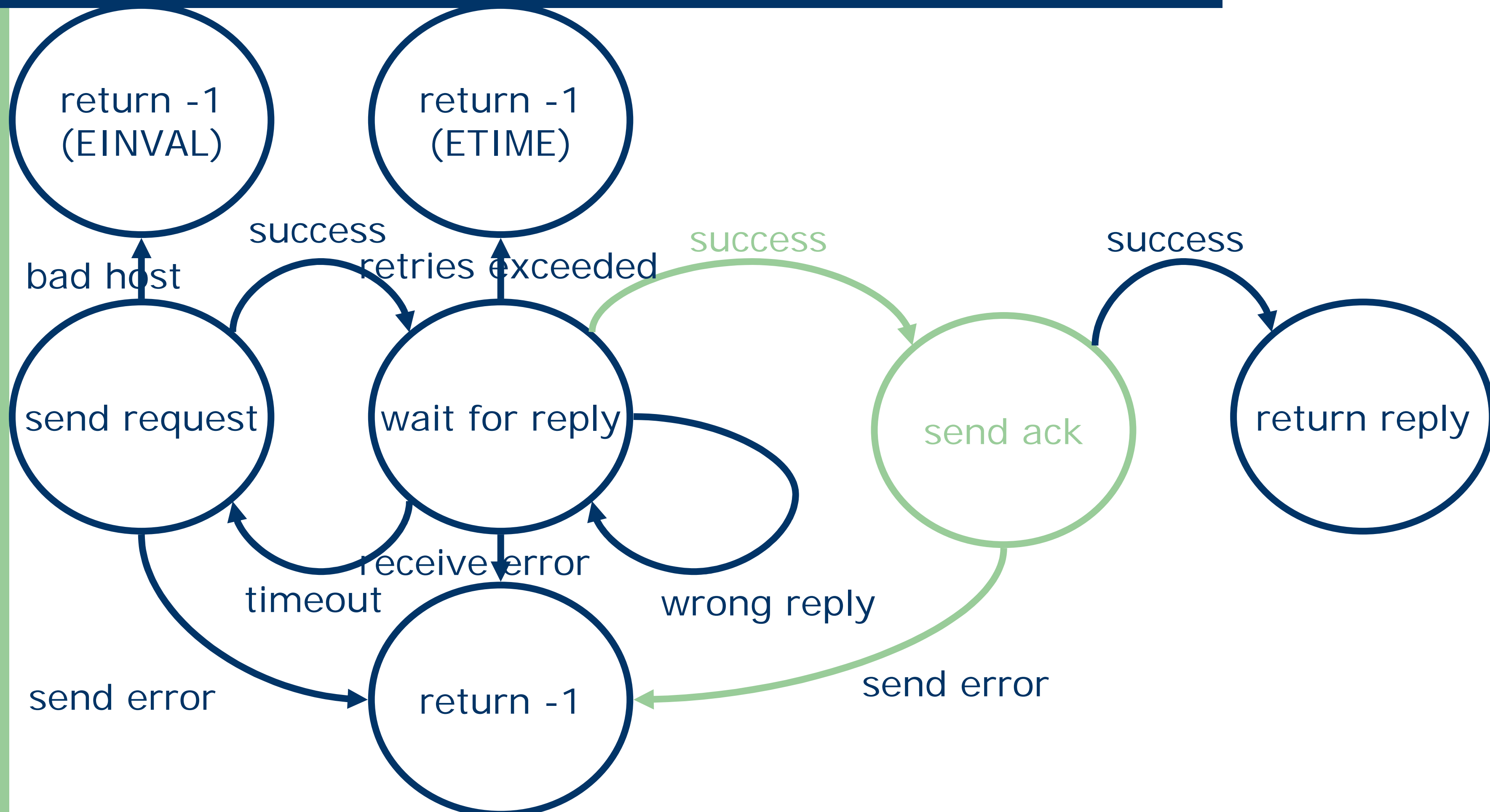
Request-Reply with Timeouts and Retries

```
int request_reply_timeout_retry(int requestfd, void* request, int reqlen,
                char* server, int serverport, void *reply, int replen,
                double timeout, int maxretries) { ...
  retries = 0;
  while (retries < maxretries){/*send process ID to (server, serverport)*/
    nbytes = u_sendtohost(requestfd, request, reqlen, server,serverport);
    if (nbytes == -1)
      return -1;                                  /* error on send */
     /*wait timeout seconds for a response, restart if from wrong server*/
    while (((nbytes = u_recvfromtimed(requestfd, reply, replen,
                                 &senderinfo, timeout)) >= 0) &&
          (u_comparehost(&senderinfo, server, serverport) == 0)) ;
    if (nbytes >= 0)
      break;
    retries++;}
  if (retries >= maxretries) {errno = ETIME;return -1;}
  return (int)nbytes;  }
```

Request-Reply-Acknowledge Protocols

- The request_reply function implements maybe semantics.
- The request_reply_timeout approximates at-least-once semantics
- At-least-once semantics requires idempotent operations – it doesn't matter if the operation is repeated

State Diagram of Client for Request-Reply-Acknowledge



Server Behavior

- Server discards reply after receiving acknowledgment.
- Best used with sequence numbers

Comparison UDP and TCP

- TCP is connection-oriented and UDP is not.
- UDP is based on messages and TCP is based on byte streams.
- TCP delivers streams of bytes in the same order in which they were sent.
- TCP is reliable and UDP is unreliable.
- UDP sendto and TCP write functions return after successfully copying their message into a buffer of the network subsystem. TCP may hold outgoing data in its buffers.

Summary

- UDP
- Request Reply
- Timed messages
- Acknowledged messages
- Differences between TCP and UDP