

Communication Networks Laboratory  
The University of Kansas EECS 780  
Introduction to Socket Programming

Egemen Çetinkaya and James P.G. Sterbenz

Department of Electrical Engineering & Computer Science  
Information Technology & Telecommunications Research Center  
The University of Kansas



*jpgs@eecs.ku.edu*

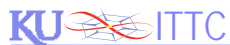
*ekc@itc.ku.edu*

*<http://www.itc.ku.edu/~jpgs/courses/nets>*

18 February 2009

rev. 9.0

© 2004–2009 James P.G. Sterbenz



© James P.G. Sterbenz

Socket Programming  
Outline

- L2.1 Motivation and overview
- L2.2 Socket programming stages
- L2.3 Socket programming examples

18 February 2009

KU EECS 780 – Comm Nets – Socket Prog. Lab

NET-L1-2

## Socket Programming

### Motivation and Overview

- L2.1 Motivation and overview
- L2.2 Socket programming stages
- L2.3 Socket programming examples

## Motivation and Overview

### Socket Programming and Applications

- Also called client/server application development
- Introduced in 4.1 BSD in 1982
- Network application implementations
  - standard network application
    - based on RFCs
    - programs conform rules
    - port numbers should be implemented per RFC
  - proprietary network application
    - they don't conform RFCs
    - code will not interoperate
    - don't use standard RFC well-known port numbers (0-1024)

KU ITTC © James P.G. Sterbenz

## Motivation and Overview

### E2E Application Data Flow and Sockets

- Application process send messages to TL via sockets
- Application process is controlled by the developer
- TL (TCP,UDP) is controlled by the OS

The diagram illustrates the flow of data through a network. On the left, an 'End host' has a protocol stack with layers: Application (containing a 'Socket'), Transport, Network, Link, and Physical. A downward arrow indicates data flow from the Application layer to the Physical layer. This data enters a 'Channel' (represented by a circle) and moves to a 'Router'. The Router has a protocol stack with layers: Network, Link, and Physical. An upward arrow shows data moving from the Physical layer to the Network layer. From the Router, data enters another 'Channel' and moves to a second 'End host'. This second End host has a protocol stack with layers: Application (containing a 'Socket'), Transport, Network, Link, and Physical. An upward arrow indicates data flow from the Physical layer to the Application layer.

18 February 2009 KU EECS 780 – Comm Nets – Socket Prog. Lab NET-L1-5

KU ITTC © James P.G. Sterbenz

## Motivation and Overview

### Socket Properties

- Socket
  - an interface which application processes can send & receive messages to/from another application process
  - host local
  - created by application
  - controlled by the OS
- Determined by triple `<domain, type, protocol>`
  - domain – specifies address family (e.g. AF\_INET)
  - type – comm. style (e.g. SOCK\_STREAM, SOCK\_DGRAM)
  - protocol – 0 provides default protocol for AF & socket type

18 February 2009 KU EECS 780 – Comm Nets – Socket Prog. Lab NET-L1-6

**KU ITTC** © James P.G. Sterbenz

---

## Motivation and Overview

### Sockets and Processes

- Socket is a method for Inter Process Communication
- Processes are created via client/server programs
- IPC can be done on a single host as well

read/write from/to sockets

---

18 February 2009 KU EECS 780 – Comm Nets – Socket Prog. Lab NET-L1-7

**KU ITTC** © James P.G. Sterbenz

---

## Socket Programming

### Socket Programming Stages

- L2.1 Motivation and overview
- L2.2 Socket programming stages
- L2.3 Socket programming examples

---

18 February 2009 KU EECS 780 – Comm Nets – Socket Prog. Lab NET-L1-8

## Socket Programming Stages

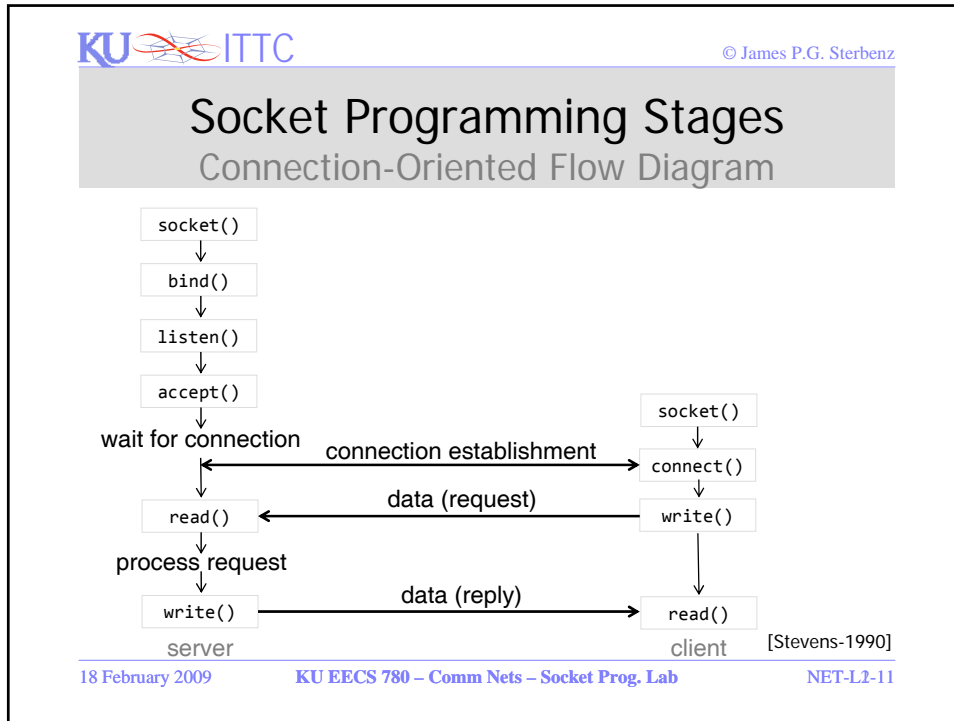
### Planning Phase


1. Developer decides programming language and OS
  - C, Java etc. and UNIX, Linux, MS etc.
2. Developer should decide:
  - to run the application on TCP
    - TCP is connection oriented, reliable byte stream channel
  - to run the application on UDP
    - UDP is connectionless service, best effort, no guarantee
  - skip transport layer to run the application
    - for hop nodes: e.g. ICMP
    - also called raw sockets
3. Developer implements the code

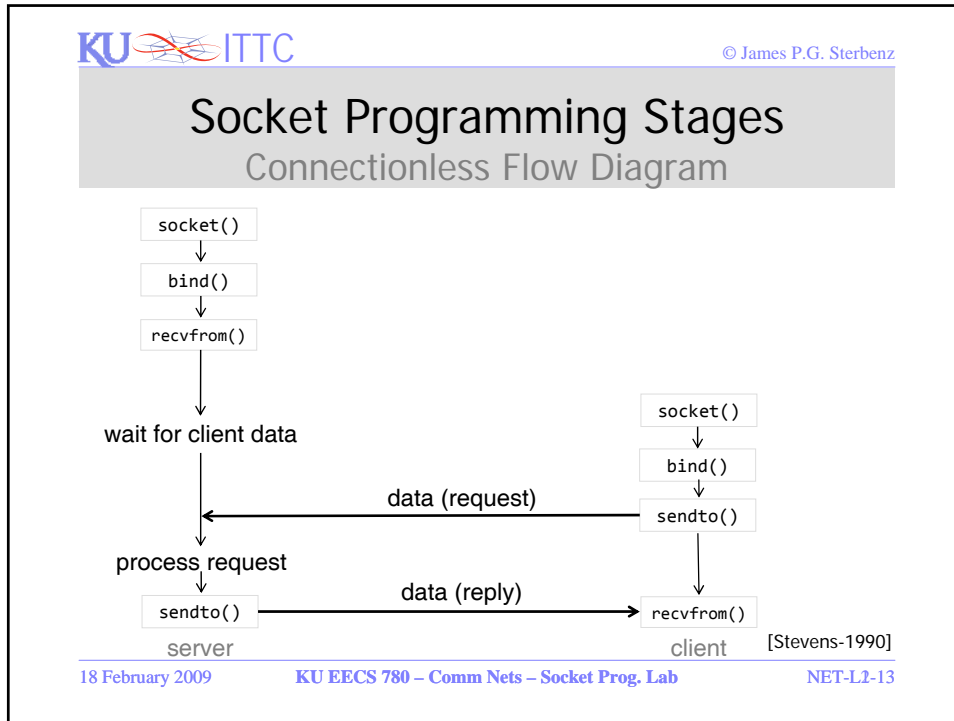
## Socket Programming Stages

### Socket Programming with TCP

- In order to establish connect. between client & server
- Server process
  - has to be ready to respond client's requests
  - server has to have a welcoming socket
- Client process
  - creates socket
  - specifies the destination
  - 3-way handshake occurs
    - client invokes server's `welcomeSocket accept()` method
    - server responds this by creating dedicated `connectionSocket`
    - TCP establishes pipe betw. `connectionSocket-clientSocket`



- 
© James P.G. Sterbenz
- ## Socket Programming Stages
- ### Socket Programming with UDP
- Connectionless transport between client & server
    - there is no initial handshaking
    - unlike TCP client can be started first
  - Client attaches destination address to each packet
  - Client process
    - creates `clientSocket` of type `DatagramSocket`
    - in TCP `clientSocket` is of type `Socket`
  - Server process
    - creates `serverSocket` of type `DatagramSocket`
    - there is no `welcomeSocket` as in TCP
- 18 February 2009
KU EECS 780 – Comm Nets – Socket Prog. Lab
NET-L2-12



KU ITTC © James P.G. Sterbenz

## Socket Programming

### Socket Programming Examples

- L2.1 Motivation and overview
- L2.2 Socket programming stages
- L2.3 Socket programming examples

---

18 February 2009 KU EECS 780 – Comm Nets – Socket Prog. Lab NET-L2-14

## Socket Programming Examples

### TCP Client<sub>1</sub>

```

void main(int argc, char *argv[])
{
    struct sockaddr_in sad;           /* structure to hold server's IP address */
    int clientSocket;               /* socket descriptor */
    struct hostent *ptrh;           /* pointer to a host table entry */

    char Sentence[128];
    char modifiedSentence[128];

    host = argv[1]; port = atoi(argv[2]);

    clientSocket = socket(PF_INET, SOCK_STREAM, 0); /* create client socket */

    memset((char *)&sad, 0, sizeof(sad));          /* clear sockaddr structure */
    sad.sin_family = AF_INET;                      /* set family to Internet */
    sad.sin_port = htons((u_short)port);
    ptrh = gethostbyname(host);                   /* convert host name to IP adr */
    memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);

    /* connect socket to specified server */
    connect(clientSocket, (struct sockaddr *)&sad, sizeof(sad));
    
```

## Socket Programming Examples

### TCP Client<sub>2</sub>

```

/* get input stream from the user */
gets(Sentence);

/* send sentence to server */
n=write(clientSocket, Sentence, strlen(Sentence)+1);

/* read line from the server and write the modified sentence to screen*/
n=read(clientSocket, modifiedSentence, sizeof(modifiedSentence));
printf("FROM SERVER: %s\n",modifiedSentence);

/* close connection */
close(clientSocket);
}
    
```

## Socket Programming Examples

### TCP Server<sub>1</sub>

```

void main(int argc, char *argv[])
{
    struct sockaddr_in sad;           /* structure to hold server's IP address */
    struct sockaddr_in cad;         /* structure to hold client's IP address */
    int welcomeSocket, connectionSocket; /* socket descriptor */
    struct hostent *ptrh;          /* pointer to a host table entry */
    char clientSentence[128];
    char capitalizedSentence[128];

    port = atoi(argv[1]);

    /* create welcoming socket at port and bind a local address */

    welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);

    memset((char *)&sad, 0, sizeof(sad)); /* clear sockaddr structure */
    sad.sin_family = AF_INET;           /* set family to Internet */
    sad.sin_addr.s_addr = INADDR_ANY;  /* set the local IP address */
    sad.sin_port = htons((u_short)port); /* set the port number */

    bind(welcomeSocket, (struct sockaddr *)&sad, sizeof(sad));

```

## Socket Programming Examples

### TCP Server<sub>2</sub>

```

/* Specify the maximum number of clients that can be queued */
listen(welcomeSocket, 10)

/* Main server loop - accept and handle requests */
while(1) {

    /* Wait on welcoming socket for contact by client */
    connectionSocket=accept(welcomeSocket, (struct sockaddr *)&cad, &alen);

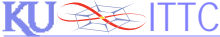
    n=read(connectionSocket, clientSentence, sizeof(clientSentence));

    /* capitalize Sentence and store the result in capitalizedSentence */

    /* write out the result to socket */
    n=write(connectionSocket, capitalizedSentence, strlen(capitalizedSentence)+1);

    close(connectionSocket);
}

```


© James P.G. Sterbenz

## Socket Programming Examples

### UDP Client<sub>1</sub>

```

void main(int argc, char *argv[])
{
  struct sockaddr_in sad;           /* structure to hold an IP address */
  int clientSocket;               /* socket descriptor */
  struct hostent *ptrh;           /* pointer to a host table entry */
  char Sentence[128];
  char modifiedSentence[128];

  host = argv[1]; port = atoi(argv[2]);

  /* Create client socket, NO connection to server */


  clientSocket = socket(PF_INET, SOCK_DGRAM, 0);

  /* determine the server's address */

  memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
  sad.sin_family = AF_INET;          /* set family to Internet */
  sad.sin_port = htons((u_short)port);
  ptrh = gethostbyname(host);       /* Convert host name to IP address */
  memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
    
```

---

18 February 2009
KU EECS 780 – Comm Nets – Socket Prog. Lab
NET-L1-19


© James P.G. Sterbenz

## Socket Programming Examples

### UDP Client<sub>2</sub>

```

/* get input stream from user */
gets(Sentence);

/* send line to server */

addr_len =sizeof(struct sockaddr);
n=sendto(clientSocket, Sentence, strlen(Sentence)+1,
         (struct sockaddr *) &sad, addr_len);

/* read line from server */

n=recvfrom(clientSocket, modifiedSentence, sizeof(modifiedSentence).
           (struct sockaddr *) &sad, &addr_len);

printf("FROM SERVER: %s\n",modifiedSentence);

/* close connection */

close(clientSocket);
}
    
```

---

18 February 2009
KU EECS 780 – Comm Nets – Socket Prog. Lab
NET-L1-20

## Socket Programming Examples

### UDP Server<sub>1</sub>

```

void main(int argc, char *argv[])
{
    struct sockaddr_in sad;           /* structure to hold server's IP address */
    struct sockaddr_in cad;          /* structure to hold client's IP address */
    int serverSocket;                /* socket descriptor */
    struct hostent *ptrh;            /* pointer to a host table entry */

    char clientSentence[128];
    char capitalizedSentence[128];

    port = atoi(argv[1]);

    /* create welcoming socket at port & bind a local address */

    serverSocket = socket(PF_INET, SOCK_DGRAM, 0);
    memset((char *)&sad, 0, sizeof(sad)); /* clear sockaddr structure */
    sad.sin_family = AF_INET;           /* set family to Internet */
    sad.sin_addr.s_addr = INADDR_ANY;  /* set the local IP address */
    sad.sin_port = htons((u_short)port); /* set the port number */

    bind(serverSocket, (struct sockaddr *)&sad, sizeof(sad));

```

## Socket Programming Examples

### UDP Server<sub>2</sub>

```

/* main server loop - accept and handle requests */
while(1) {
    /* receive messages from clients */
    n=recvfrom(serverSocket, clientSentence, sizeof(clientSentence), 0
               (struct sockaddr *) &cad, &addr_len );

    /* capitalize Sentence and store the result in capitalizedSentence */

    for(i=0; i <= strlen(clientSentence); i++) {
        if((clientSentence[i] >= 'a') && (clientSentence[i] <= 'z'))
            capitalizedSentence[i] = clientSentence[i] + ('A' - 'a');
        else
            capitalizedSentence[i] = clientSentence[i];
    }
    /* write out the result to socket */
    n=sendto(connectionSocket, capitalizedSentence, strlen(capitalizedSentence)+1, 0
             (struct sockaddr *) &cad, &addr_len);

    close(connectionSocket);
} /* end of while loop, loop back and wait for another connection client*/

```

## Socket Programming Acknowledgements

Some material in these foils comes from the textbook  
supplementary materials:

- Kurose & Ross,  
*Computer Networking:  
A Top-Down Approach Featuring the Internet*, 4th ed.  
[http://wps.aw.com/aw\\_kurose\\_network\\_4](http://wps.aw.com/aw_kurose_network_4)
- Stevens,  
*UNIX Network Programming*  
*Prentice Hall, 1990*  
<http://www.kohala.com/start/unp.html>

## Socket Programming Acknowledgements

Some material in these foils comes from the textbook  
supplementary materials:

- Donahoo & Calvert,  
*TCP/IP Sockets in C: Practical Guide for Programmers*  
*Morgan Kaufmann, 2001*  
<http://cs.ecs.baylor.edu/~donahoo/practical/CSockets/>
- Donahoo & Calvert,  
*TCP/IP Sockets in Java: Practical Guide for Programmers*, 2nd ed.  
*Morgan Kaufmann, 2008*  
<http://cs.ecs.baylor.edu/~donahoo/practical/JavaSockets/>

## Socket Programming

### Additional Reading

Some material in these foils comes from the textbook  
supplementary materials:

- [http://gaia.cs.umass.edu/ntu\\_socket/](http://gaia.cs.umass.edu/ntu_socket/)
- <http://www.frostbytes.com/~jimf/papers/sockets/sockets.html>
- <http://beej.us/guide/bgnet/>
- <http://java.sun.com/docs/books/tutorial/networking/sockets/>