Network Programming

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani

Pilani Campus

Outline

Outline

- Broadcasting
  - example
  - Race conditions when using signals
- Multicasting
  - Addresses
  - Multicasting on LAN
  - Multicasting on WAN
  - Source Specific Multicast
  - Multicast Socket Options
  - Simple Network Time Protocol
- RPC

innovate achieve lead

BITS Pilani

Pilani Campus

Broadcasting

T1: Ch 20

Broadcasting

- Many networks support the notion of sending a message from one host to all other hosts on the network.
- A special address called the "*broadcast address*" is often used.
- Some popular network services are based on broadcasting
  - YP/NIS
  - Routed
  - DHCP

IPv4 & IPv6 Support

- TCP works only with unicast addresses, UDP supports also broadcasting and multicasting

Type	IPv4	IPv6	TCP	UDP
Unicast	✓	✓	✓	✓
Broadcast	✓			✓
Multicast	opt.	✓		✓
Anycast	*	✓		✓

- Multicasting support is optional in IPv4, but mandatory in IPv6
- Broadcasting support is not provided in IPv6; if an IPv4 application uses broadcasting, recode with IPv6 to use multicasting instead of broadcasting

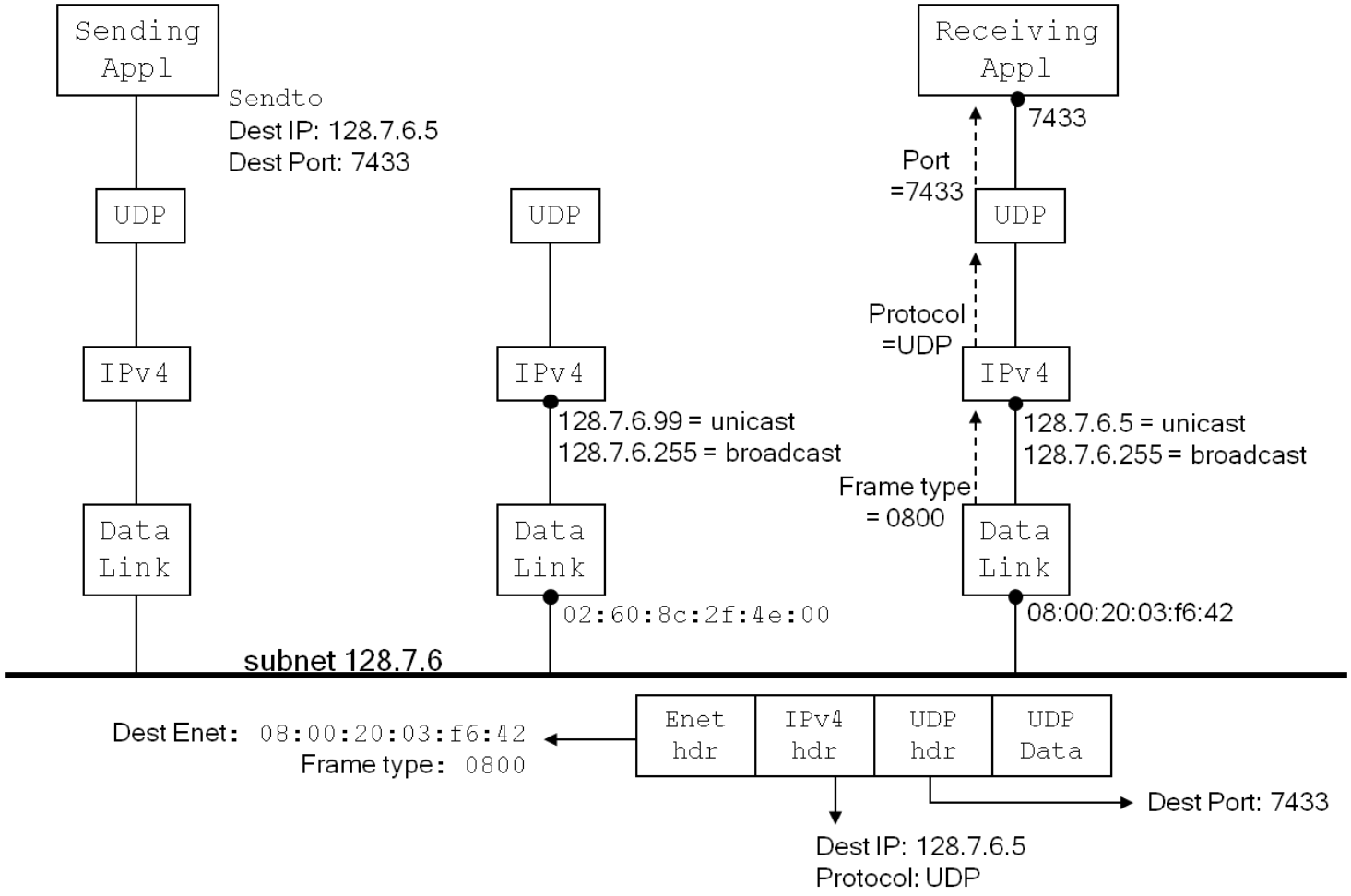
Broadcast Address

- IPv4 address: {netid; subnetid; hostid}
- Subnet-directed Broadcast Address:
  - {netid; subnetid; -1} //-1 means all bits are 1's
  - netid = 128.7, subnetid: 6
  - Broadcast Address: 128.7.6.255
  - Normally, routers do not forward these broadcasts
- Limited Broadcast Address:
  - {-1; -1; -1} or 255.255.255.255
  - Must never be forwarded by a router
  - Subnet-directed broadcast and limited broadcast are the most common
  - Old systems do not understand subnet-directed broadcast
  - For protocols like BOOTP, 255.255.255.255 is the only option

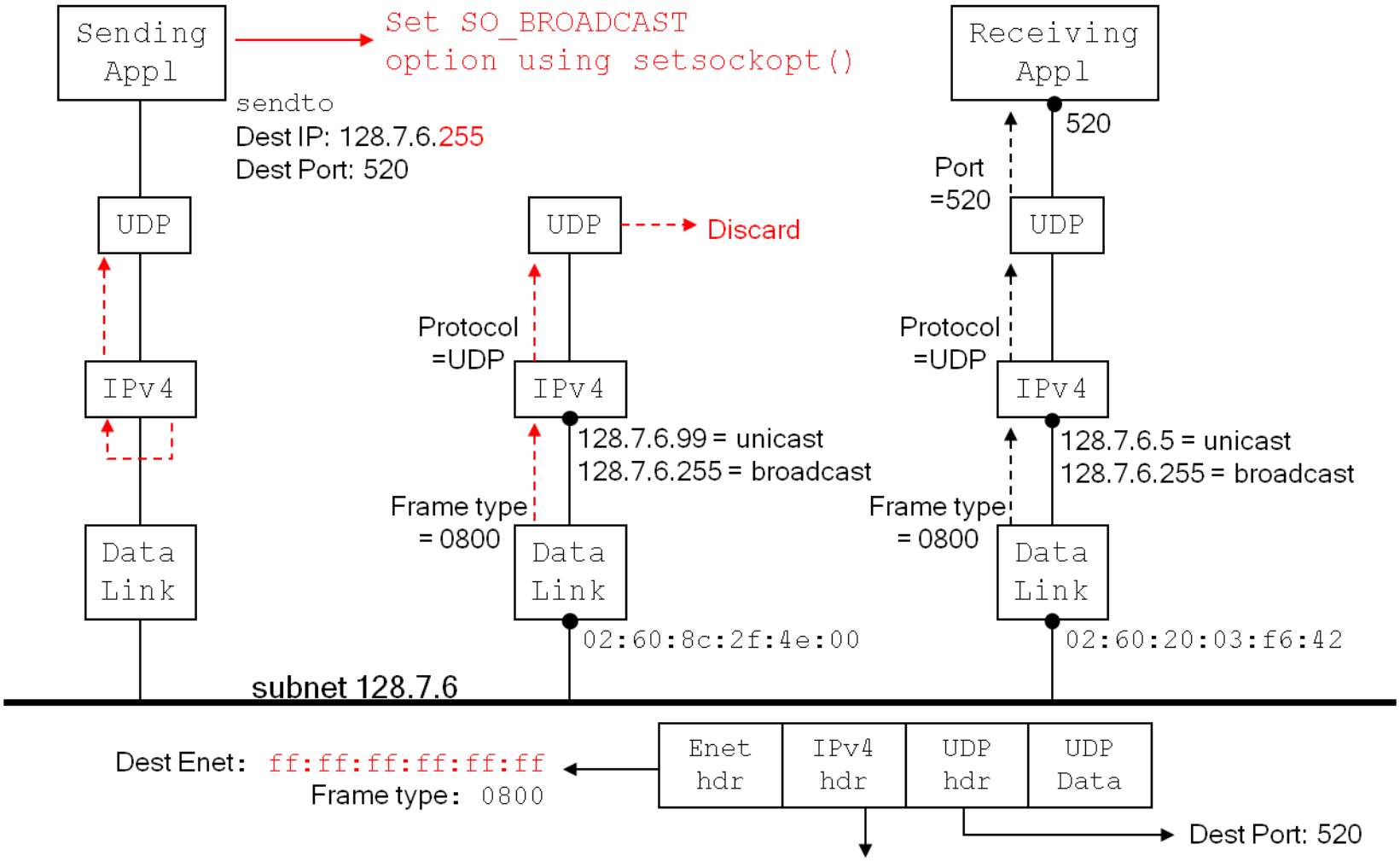
Unicast vs Broadcast

- In Unicast, only peers participate
- In Broadcast, every host on the subnet has to receive the packet and process it up to the transport layer i.e through DL,IP, and UDP
- Every non-IP host also must receive at the datalink layer
- If broadcast datagrams arrive at higher rate, processing can affect severely the performance
-

Unicast in LAN



Broadcast in a LAN



Broadcast

- The datagram that is output by the host is being delivered to itself.
  - Assume that the sending application has bound the port. Receives a copy of each broadcast datagram.
- Every host must completely process the broadcast UDP datagram all the way up the protocol stack, before discarding the datagram.
- Every non-IP host on the subnet must also receive the entire frame at the datalink layer before discarding the frame.

Steps to Write Broadcast Program

- Create a UDP socket
- Socket option has to be set with SO_BROADCAST

```
1   int on=1;
2   setsockopt(sockfd, SOL_SOCKET,SO_BROADCAST,&on,sizeof(on));
```

- Prepare sockaddr structure with { broadcast ip and port no}
- Send data using *sendto().*

Broadcast Client

innovate achieve lead

```
1  int main(int argc, char *argv[]) {
2    in_port_t port = htons((in_port_t) atoi(argv[2]));
3    struct sockaddr_storage destStorage;
4    memset(&destStorage, 0, sizeof(destStorage));
5    size_t addrSize = 0;
6    struct sockaddr_in *destAddr4 = (struct sockaddr_in *) &destStorage;
7    destAddr4->sin_family = AF_INET;
8    destAddr4->sin_port = port;
9    destAddr4->sin_addr.s_addr = INADDR_BROADCAST;
10   addrSize = sizeof(struct sockaddr_in);
11   struct sockaddr *destAddress = (struct sockaddr *) &destStorage;
12   size_t msgLen = strlen(argv[3]);
13   // Create socket for sending/receiving datagrams
14   int sock = socket(destAddress->sa_family, SOCK_DGRAM, IPPROTO_UDP);
15   // Set socket to allow broadcast
16   int broadcastPerm = 1;
17   if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &broadcastPerm,
18       sizeof(broadcastPerm)) < 0)  DieWithSystemMessage("setsockopt() failed");
```

Broadcast Client

innovate achieve lead

```
19    for (;;) {// Run forever
20      // Broadcast msgString in datagram to clients every 3 seconds
21      ssize_t numBytes = sendto(sock, argv[3], msgLen, 0, destAddress, addrSize);
22      if (numBytes < 0)   DieWithSystemMessage("sendto() failed");
23      else if (numBytes != msgLen)
24        DieWithUserMessage("sendto()", "sent unexpected number of bytes");
25      sleep(3);   // Avoid flooding the network
26    }}
```

- Broadcast storm
  - Since every host processes the datagram, and if the host doen't have a server running at that port number, "ICMP port unreachable message will be generated and sent.

innovate achieve lead

BITS Pilani

Pilani Campus

Socket Timeouts

T1: Ch 14.2

Timeout on Sockets

- Three ways to place a timeout on an I/O operation involving a socket
  - Call alarm, which generates the SIGALRM signal when the specified time has expired.
  - Block waiting for I/O in select, which has a time limit built in, instead of blocking in a call to read or write.
  - Use the newer SO_RCVTIMEO and SO_SNDTIMEO socket options.

alarm()

innovate achieve lead

```
1  void dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
2  {   int n;
3      char    sendline[MAXLINE], recvline[MAXLINE + 1];
4      signal(SIGALRM, sig_alrm);
5      while (fgets(sendline, MAXLINE, fp) != NULL) {
6          sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
7          alarm(5);
8          if ( (n = recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL)) < 0) {
9              if (errno == EINTR)
10                 fprintf(stderr, "socket timeout\n");
11             else
12                 err_sys("recvfrom error");
13         } else {
14             alarm(0);
15             recvline[n] = 0;    /* null terminate */
16             fputs(recvline, stdout);
17         }
18     }
19 }
20 static void sig_alrm(int signo)
21 {
22     return;         /* just interrupt the recvfrom() */
23 }
```

Timeout using *select()*

innovate achieve lead

```
 2  int
 3  readable_timeo(int fd, int sec)
 4  {
 5      fd_set          rset;
 6      struct timeval  tv;
 7      FD_ZERO(&rset);
 8      FD_SET(fd, &rset);
 9      tv.tv_sec = sec;
10      tv.tv_usec = 0;
11
12      return(select(fd+1, &rset, NULL, NULL, &tv));
13          /* > 0 if descriptor is readable
14             = 0 if timeout                    */
15  }
```

innovate achieve lead

BITS Pilani

Pilani Campus

Race Conditions in Handling Signals

T1: Ch 20.5

Race Condition


```
1  void dg_cli(…) {
2      setsockopt(sockfd, SOL_SOCKET,SO_BROADCAST,&on,sizeof(on));
3      signal(SIGALRM, func);
4      while(fgets(…)!=NULL) {
5          sendto(…);
6          alarm(1);
7          for(; ; ) {
8              if (n=recvfrom(…) <0) {
9                  if (errno==EINTR)   break;
10                 else  err_sys(…);
11             } else {
12                 recvline[n]=0;
13                 sleep(1);
14                 printf(…);
15             }}}
16 void func( int signo)   {   return; }
```

- Signal is delivered asynchronously. Timing of signal delivery can change the final outcome of the program.
- If signal is delivered at line no 14, what will happen?

Solution1: Block Signals

innovate achieve lead

```
1   sigaddset(&sig1, SIGALRM);
2       signal(SIGALRM, func);
3       while(fgets(…) !=NULL))
4           sendto(…);
5           alarm(5);
6           for(; ; ){
7               sigprocmask(SIG_UNBLOCK, &sig1,NULL);
8               n=recvfrom(…);
9               sigprocmask(SIG_BLOCK,&sig1, NULL);
10              if(n<0) {
11                if (errno==EINTR) break; else err_sys(…);
12              } else { recvline[n]=0; printf(…); }}}
13      void func(…)
14      {return;}
```

- Signal Generation and Delivery is controlled.
- Window is reduced but the problem still persists.
  - Signal may get delivered even before recvfrom() is called.

Solution2: pselect()

```
2       FD_ZERO(&rset);
3       sigemptyset(&sigset_empty);
4       sigemptyset(&sigset_alrm);
5       sigaddset(&sigset_alrm, SIGALRM);
6       signal(SIGALRM, recvfrom_alarm);
7       while (fgets(sendline, MAXLINE, fp) != NULL) {
8           sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);
9           sigprocmask(SIG_BLOCK, &sigset_alrm, NULL);
10          alarm(5);
11          for ( ; ; ) {
12              FD_SET(sockfd, &rset);
13              n = pselect(sockfd + 1, &rset, NULL, NULL, NULL, &sigset_empty);
14              if (n < 0) {
15                  if (errno == EINTR) break;
16                  else err_sys("pselect error");
17              } else if (n != 1)
18                  err_sys("pselect error: returned %d", n);
19              len = servlen;
20              n = recvfrom(sockfd, recvline, MAXLINE, 0, preply_addr, &len);
21              recvline[n] = 0;     /* null terminate */
22          }
23      }
```

- Using *sigprocmask()*, block SIGALRM signal.
- To *pselect()*, give empty signal mask. When pselect() returns, it will restore the mask atomically.

Solution 3: sigjmp(), longjmp()

innovate achieve lead

```
1   signal(SIGALRM, func);
2       while (fgets(…)!=NULL) {
3           sendto(…);
4           alarm(5);
5           for(; ;) {
6               if (sigsetjmp(jmpbuf, 1) != 0)
7                   break;
8               n=recvfrom(…);
9               recvline[n]=0;
10              printf(…);
11          }
12  void func(…) {
13  siglongjmp(jmpbuf, 1);
14  }
```

- When signal handler executes, call siglongjmp().

Solution4: self-pipe trick

```
1  void dg_cli(…) {
2      setsockopt(…);
3      pipe (pipefd);
4      FD_ZERO(&rset);
5      signal(SIGALRM, func);
6      while(fgets(…)!=NULL){
7          sendto(…);
8          alarm(5);
9          for(; ;) {
10             FD_SET(sockfd, &rset);
11             FD_SET(pipefd[0],&rset);
12             if(n = select (…) <0) {
13                 if (errno==EINTR) continue; else err_sys(…); }
14             if (FD_ISSET(sockfd, &rset) ) {
15                 recvfrom(…); printf(…); }
16             if (FD_ISSET(pipefd[0], &rset)) {
17                 read(pipefd[0], &n, 1); break; }
18 void func(int signo) {
19     write (pipefd[1], "", 1); return;}
```

- Self pipe trick

innovate achieve lead

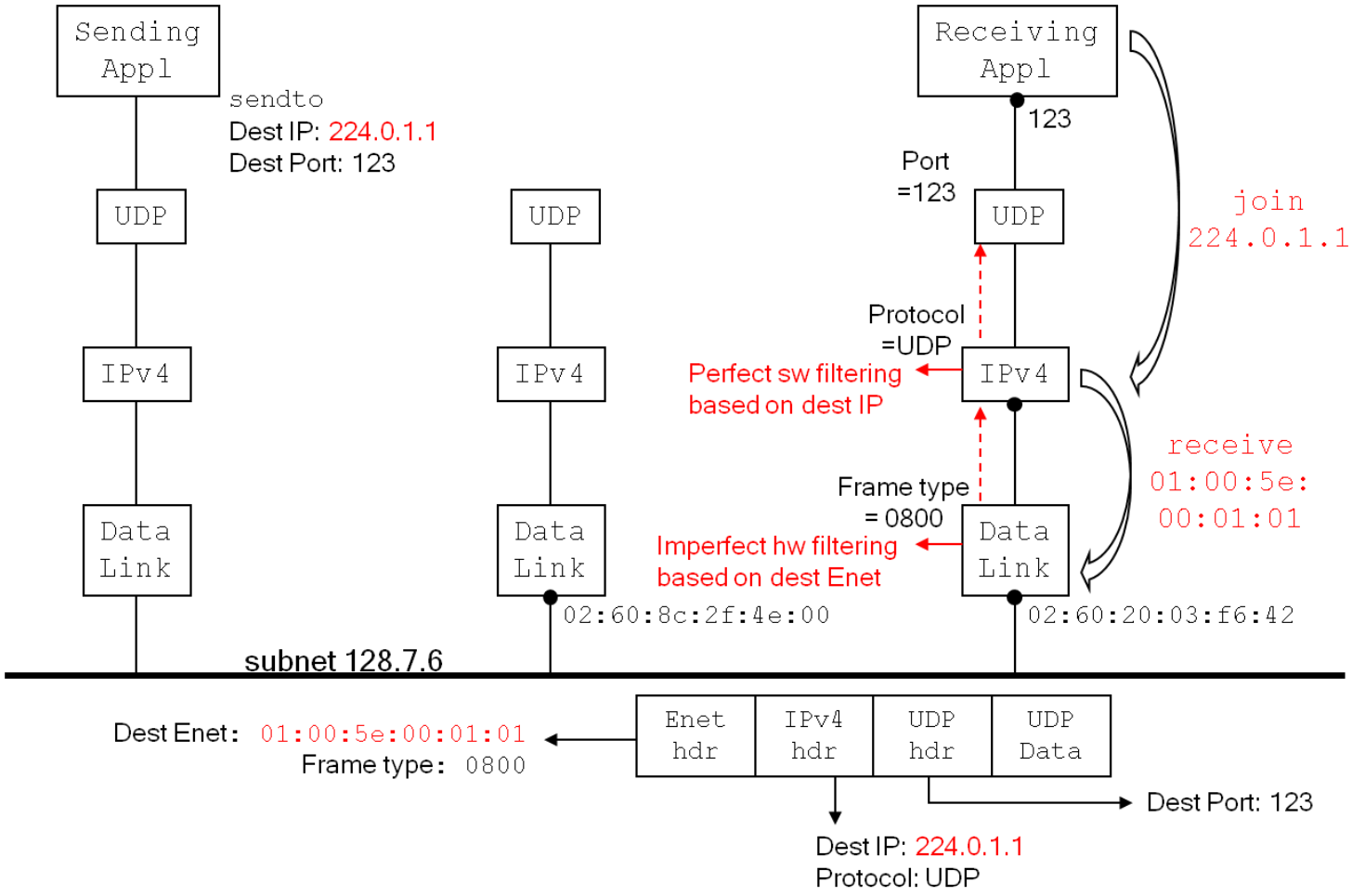
BITS Pilani

Pilani Campus

Mulicasting

T1: Ch 21

Multicasting on a LAN



Multicast Addresses

- IPv4 Class D addresses are multicast addresses
- Range 224.0.0.0 to 239.255.255.255
  - 32 bit Class D address is called the group address
- A mapping from IPv4 multicast addresses to Ethernet addresses is also defined
  - High order 24 bits always 01:00:5e
  - 25th bit is 0
  - Low order 23 bits from lowest 23 bits of multicast group address
- Not one-to-one, many (32) multicast addresses to a single Ethernet address
- Broadcasting is normally limited to LANs, whereas Multicasting can be done in LANs or WANs

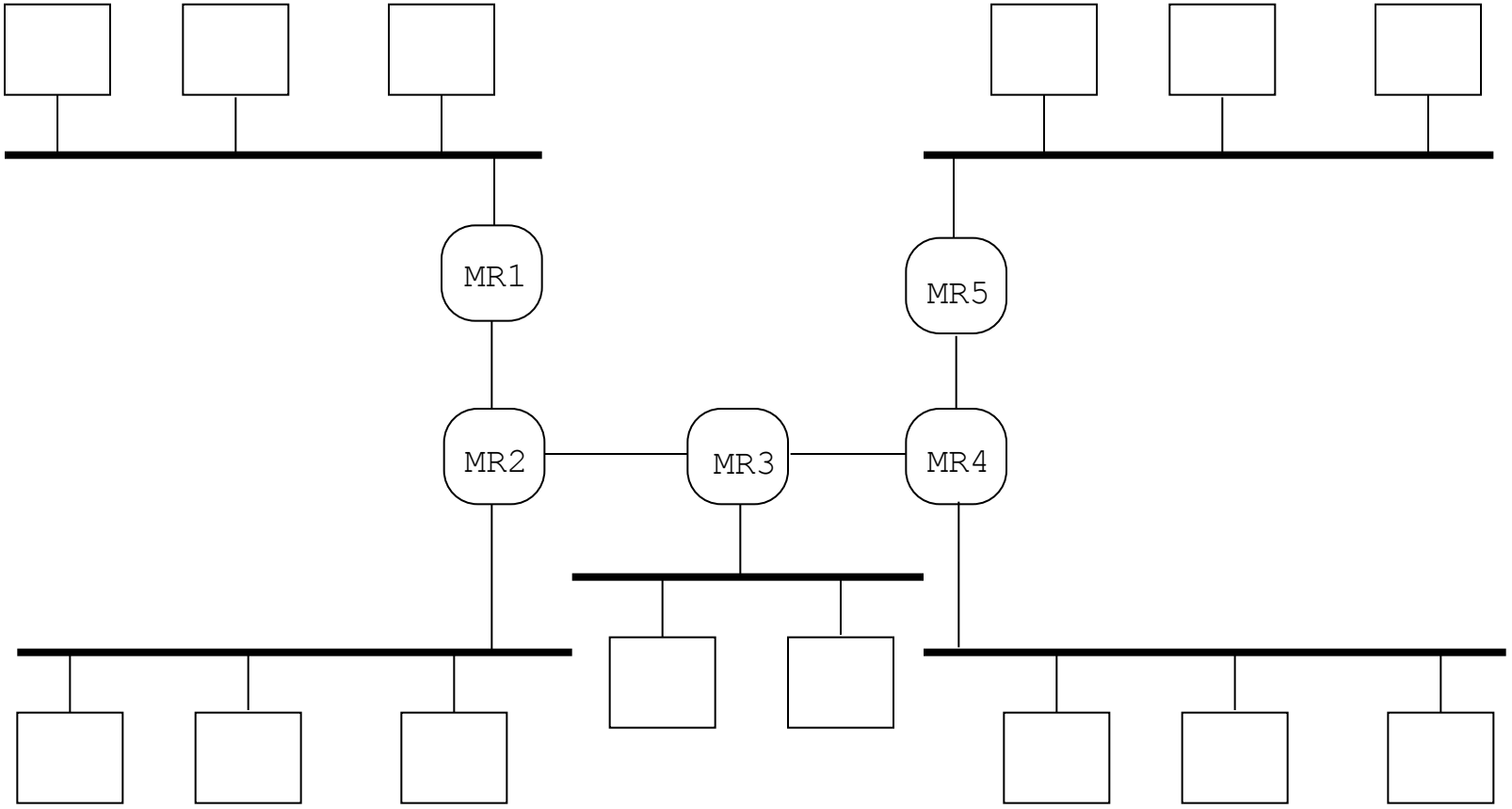
Multicast Session

- In the case of streaming multimedia, the combination of an IP multicast address and a transport-layer port is referred to as a session.
  - Audio/video teleconference may comprise two sessions;
    - one for audio and one for video.
    - These sessions almost always use different ports and sometimes also use different groups for flexibility in choice when receiving.

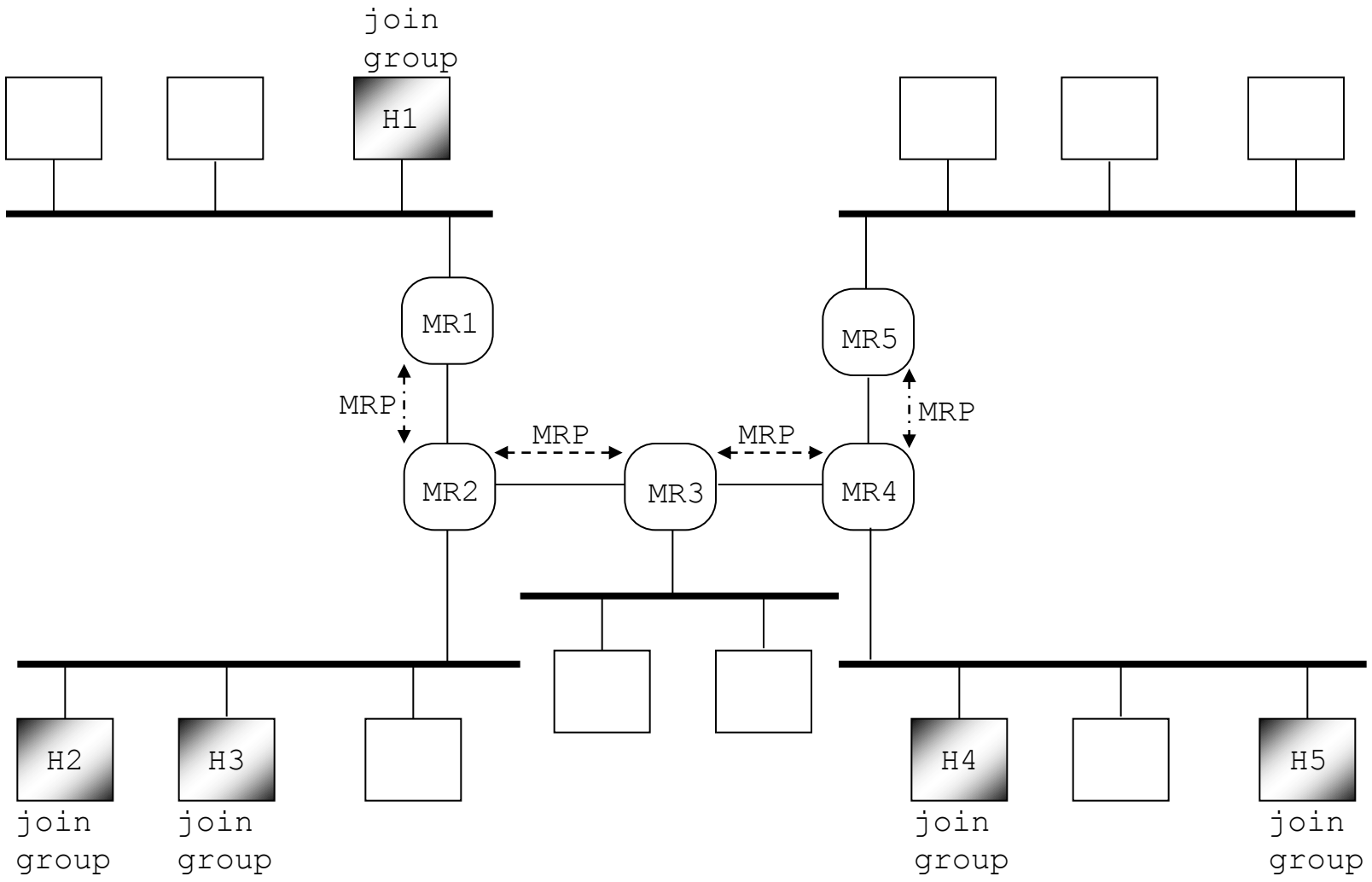
Multicasting on a WAN

- Multicasting on a single LAN is simple.
  - Benefit over broadcast is that it reduces the load on all the hosts interested in multicasting.
- Multicasting on a WAN requires
  - Hot to router protocol : IGMP
  - Router to router protocol: Multicast routing protocol.
- Alternatives to multicast are:
  - broadcast flooding
  - and sending individual copies to each receiver.

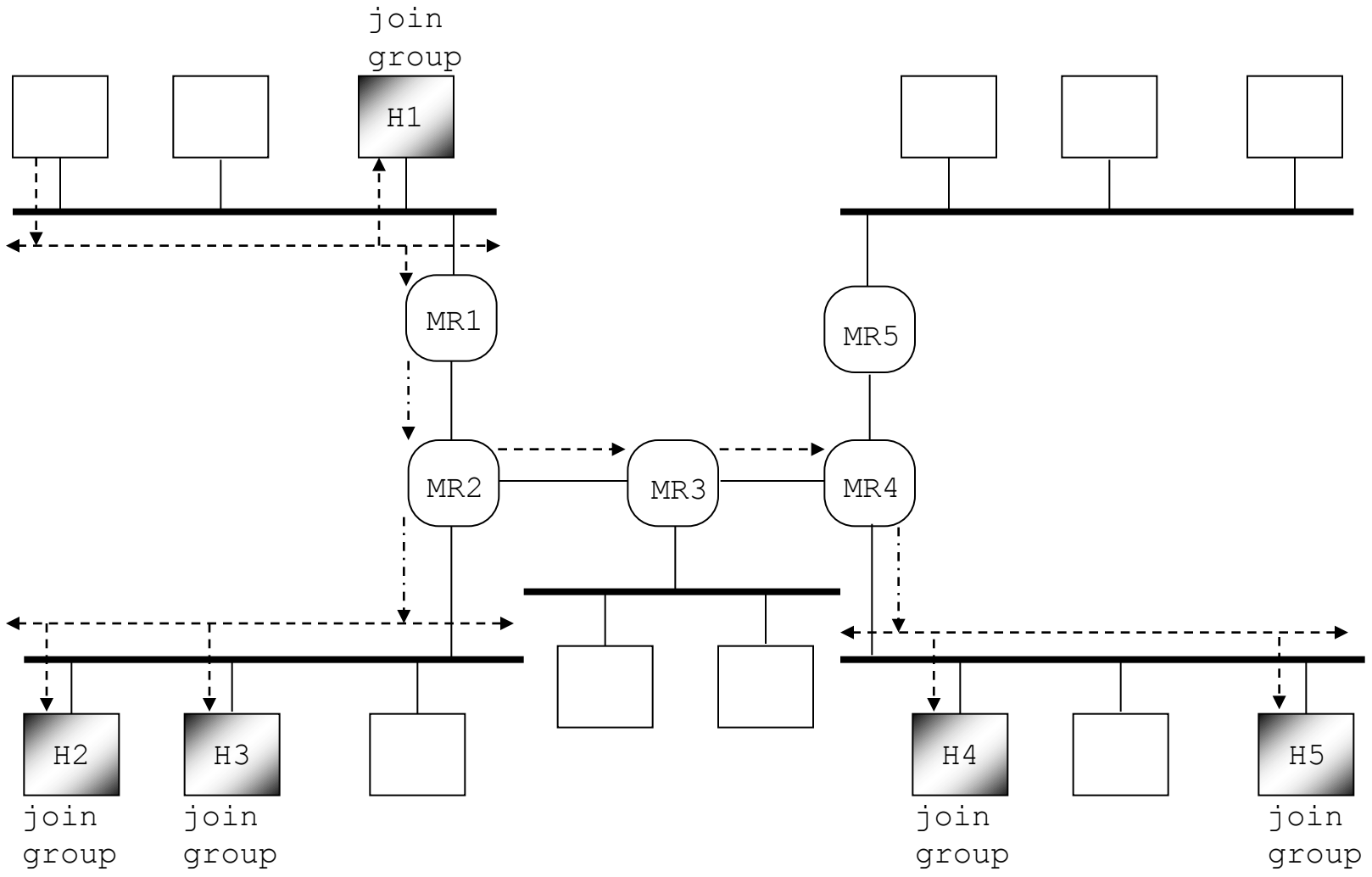
Multicasting on a WAN



Hosts joining a Multicast Group



Sending packets on a WAN



Multicasting on a WAN

- All interested multicast routers receive the packets, MR5 does not receive any since there are no interested hosts in its LAN
- Packets are put to the specific LAN only if there are hosts in that LAN to receive those packets, MR3 only forwards
- Multicast router MR2 both puts packets on its LAN for hosts H2 & H3, and also makes a copy of the packets and forwards them to MR3.

- **This behavior is something unique to multicast forwarding.**

Source Specific Multicast

innovate achieve lead

- Multicasting on a WAN has been difficult to deploy for several reasons.
- Rendezvous problem:
  - needs to get the data from all the senders, which may be located anywhere in the network, to all the receivers, which may similarly be located anywhere.
- Multicast address allocation:
  - There are not enough IPv4 multicast addresses to statically assign them to everyone who wants one, as is done with unicast addresses.

Source Specific Multicast (SSM)

- combines the group address with a system's source address
  - The receivers supply the sender's source address to the routers as part of joining the group.
    - This removes the rendezvous problem from the network, as the network now knows exactly where the sender is.
  - It retains the scaling properties of not requiring the sender to know who all the receivers are. This simplifies multicast routing protocols immensely.
  - It redefines the identifier from simply being a multicast group address to being a combination of a unicast source and multicast destination (which SSM now calls a channel).
  - An SSM session is the combination of source, destination, and port

Steps for Multicast Sender

innovate achieve lead

- To send to a multicast group,
  - Create a udp socket
  - Prepare *sockaddr* structure with multicast group and port number.
  - Use *sendto()* to send the packet into the network.
- Optional
  - IP_MULTICAST_IF, IPV6_MULTICAST_IF
    - Specify the interface for outgoing multicast datagrams sent on this socket.
  - IP_MULTICAST_TTL, IPV6_MULTICAST_HOPS
    - Set the IPv4 TTL or the IPv6 hop limit for outgoing multicast datagrams. If this is not specified, both will default to 1, which restricts the datagram to the local subnet.
  - IP_MULTICAST_LOOP, IPV6_MULTICAST_LOOP
    - Enable or disable local loopback of multicast datagrams. By default, loopback is enabled.

```c
#define EXAMPLE_PORT 6000
#define EXAMPLE_GROUP "239.0.0.1"
main(int argc)
{
    struct sockaddr_in addr;
    int addrlen, sock, cnt;
    struct ip_mreq mreq;
    char message[50];
    sock = socket(AF_INET, SOCK_DGRAM, 0);/* set up socket */
    bzero((char *)&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port = htons(EXAMPLE_PORT);
    addrlen = sizeof(addr);
      /* send */
   addr.sin_addr.s_addr = inet_addr(EXAMPLE_GROUP);
   while (1) {
      time_t t = time(0);
      sprintf(message, "time is %-24.24s", ctime(&t));
      printf("sending: %s\n", message);
      cnt = sendto(sock, message, sizeof(message), 0,
                (struct sockaddr *) &addr, addrlen);
      sleep(5);
       }
    }
```

Steps for Multicast Receiver

innovate achieve lead

- Create UDP socket
- Prepare *sockaddr* structure with muticast port no.
- Bind it to socket.
- Join the Multicast group
  - IP_ADD_MEMBERSHIP, IPV6_JOIN_GROUP
    - Join an any-source multicast group on a specified local interface.

```
2  struct ip_mreq {
3    struct in_addr   imr_multiaddr;    /*  IPv4 class D multicast addr */
4    struct in_addr   imr_interface;    /*  IPv4 addr of local interface */
5  };
6
7  struct ipv6_mreq {
8    struct in6_addr  ipv6mr_multiaddr; /*  IPv6 multicast addr */
9    unsigned int     ipv6mr_interface; /*  interface index, or 0 */
10 };
```

```c
#define EXAMPLE_GROUP "239.0.0.1"
main(int argc)
{
   struct sockaddr_in addr;
   struct ip_mreq mreq;
   sock = socket(AF_INET, SOCK_DGRAM, 0);/* set up socket */
   if (sock < 0) { perror("socket");exit(1); }
   bzero((char *)&addr, sizeof(addr));
   addr.sin_family = AF_INET;
   addr.sin_addr.s_addr = htonl(INADDR_ANY);
   addr.sin_port = htons(EXAMPLE_PORT);
   addrlen = sizeof(addr);
 if (bind(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0) {
        perror("bind"); exit(1);  }
     mreq.imr_multiaddr.s_addr = inet_addr(EXAMPLE_GROUP);
     mreq.imr_interface.s_addr = htonl(INADDR_ANY);
     if (setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
            &mreq, sizeof(mreq)) < 0) {
      perror("setsockopt mreq");  exit(1); }
    while (1) {
     cnt = recvfrom(sock, message, sizeof(message), 0,
          (struct sockaddr *) &addr, &addrlen);
     printf("%s: message = \"%s\"\n", inet_ntoa(addr.sin_addr), message);
        }
    }
```

Joining a Source Specific Group

- IP_ADD_SOURCE_MEMBERSHIP

```
1  struct ip_mreq_source {
2    struct in_addr   imr_multiaddr;    /* IPv4 class D multicast addr */
3    struct in_addr   imr_sourceaddr;    /* IPv4 source addr */
4    struct in_addr   imr_interface;     /* IPv4 addr of local interface */
5  };
```

Socket Options

- Use setsockopt() to modify socket options
  - IP_ADD_MEMBERSHIP
    - Join a multicast group on a specified local interface
  - IP_DROP_MEMBERSHIP
    - Leave a multicast group
  - IP_MULTICAST_IF
    - Specify the interface for outgoing multicast datagrams sent on this socket
  - IP_MULTICAST_TTL
    - Set the IPv4 TTL parameter (if not specified, default=1)
  - IP_MULTICAST_LOOP
    - Enable or disable local loopback (default is enabled)

innovate achieve lead

BITS Pilani

Pilani Campus

RPC

T2: Ch 16

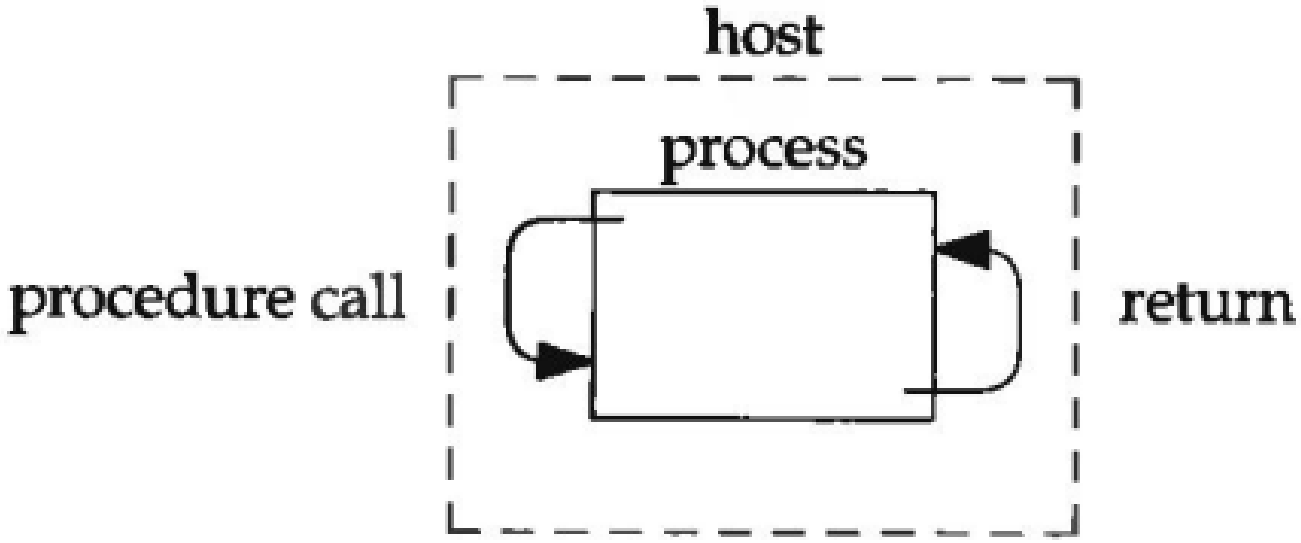
Programming Paradigm

- When we build an application, there is a choice
  - Build one large monolithic program.
  - Distribute the application among multiple processes that communicate with each other.
- Processes can be on the same host or different hosts.
  - IPC or RPC
- For communication among multiple hosts:
  - Explicit network programming
    - Direct calls to socket API
    - Connect(), bind() …
  - Implicit network programming
    - We code our application using procedure calls without worrying about the details of network I/O, protocols etc.
    - RPC runtime system provides communication support.
    - RPC provides transparency.

Distributed Program Design

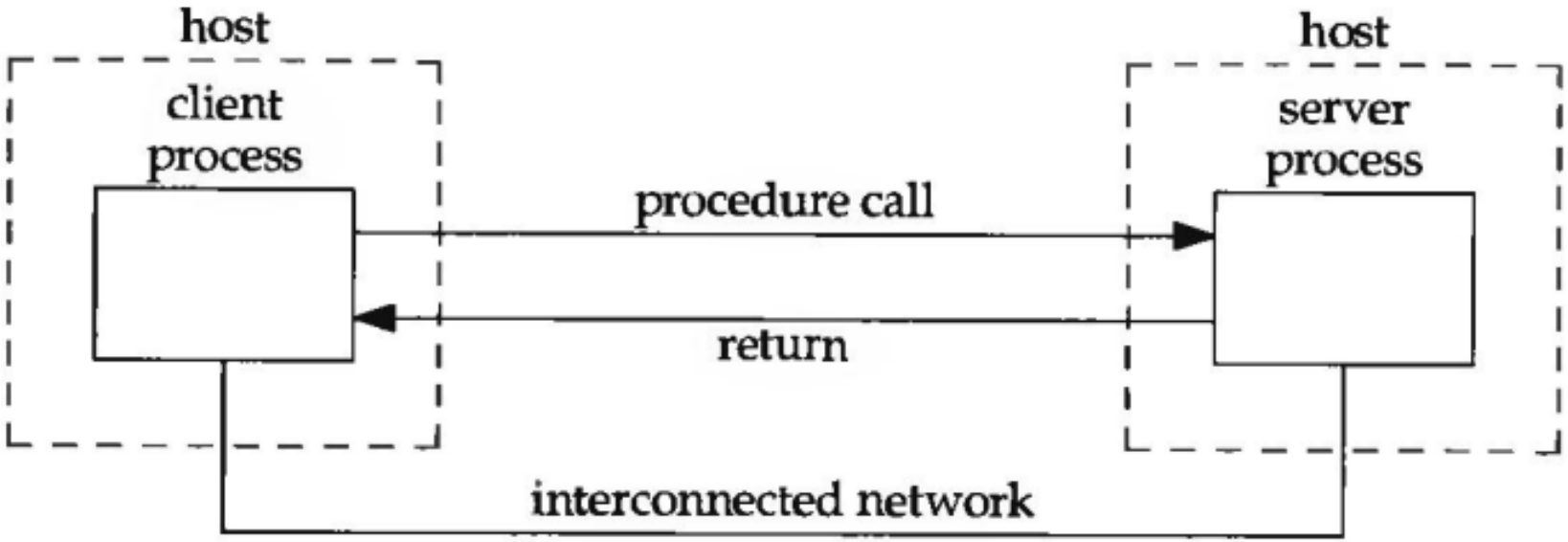
- Communication-Oriented Design
  - Design protocol first
  - Build programs that adhere to the protocol
- Application-Oriented Design
  - Build application(s)
  - Divide programs up and add communication protocols

Local Procedure Call



Remote Procedure Call

- Procedure being called and the calling procedure are in different processes.
- Caller as the client and the called as the server.

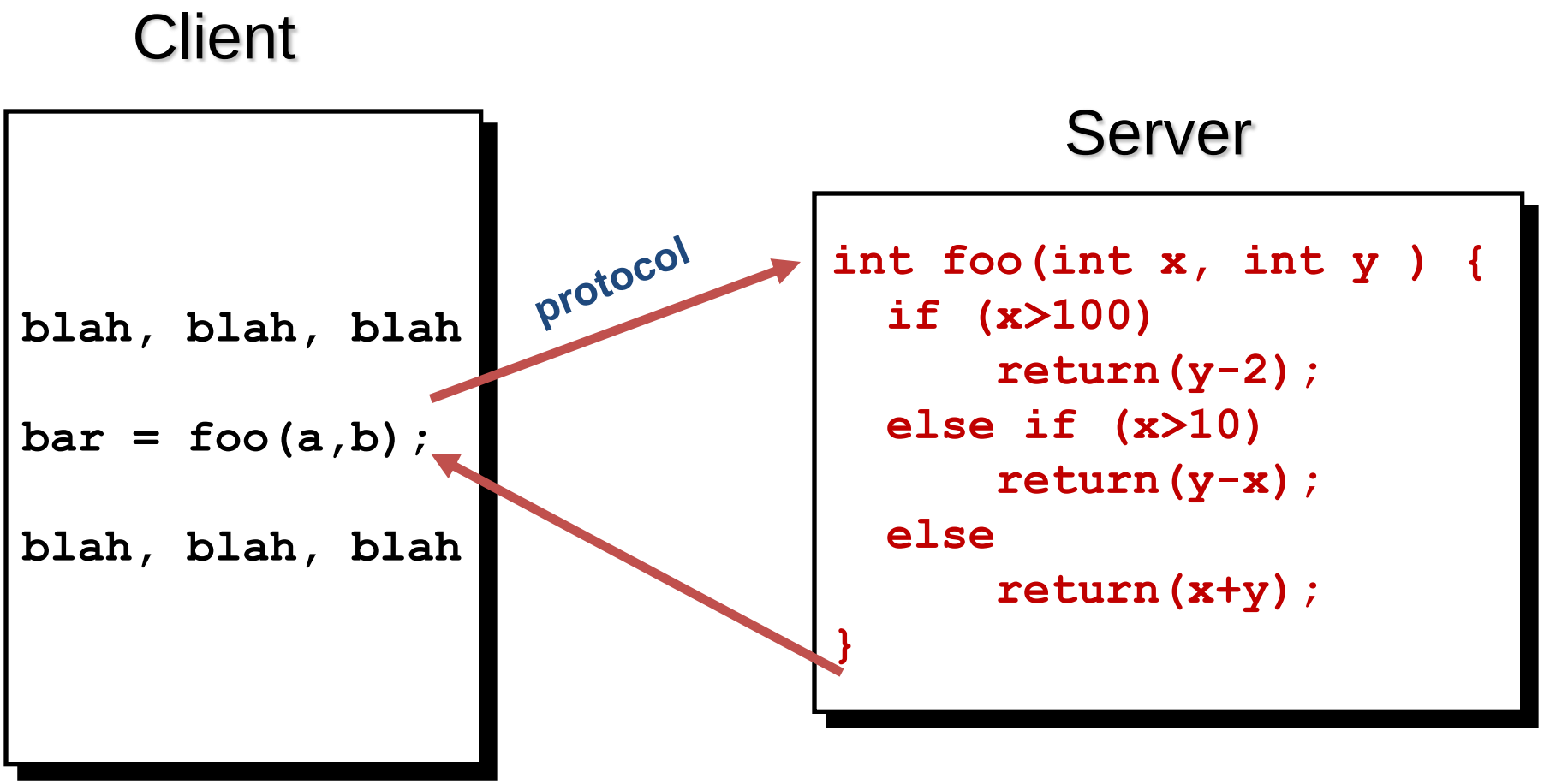


RPC: Remote Procedure Call

- Call a procedure (subroutine) that is running on another machine.

- Issues:
  - identifying and accessing the remote procedure
  - parameters
  - return value

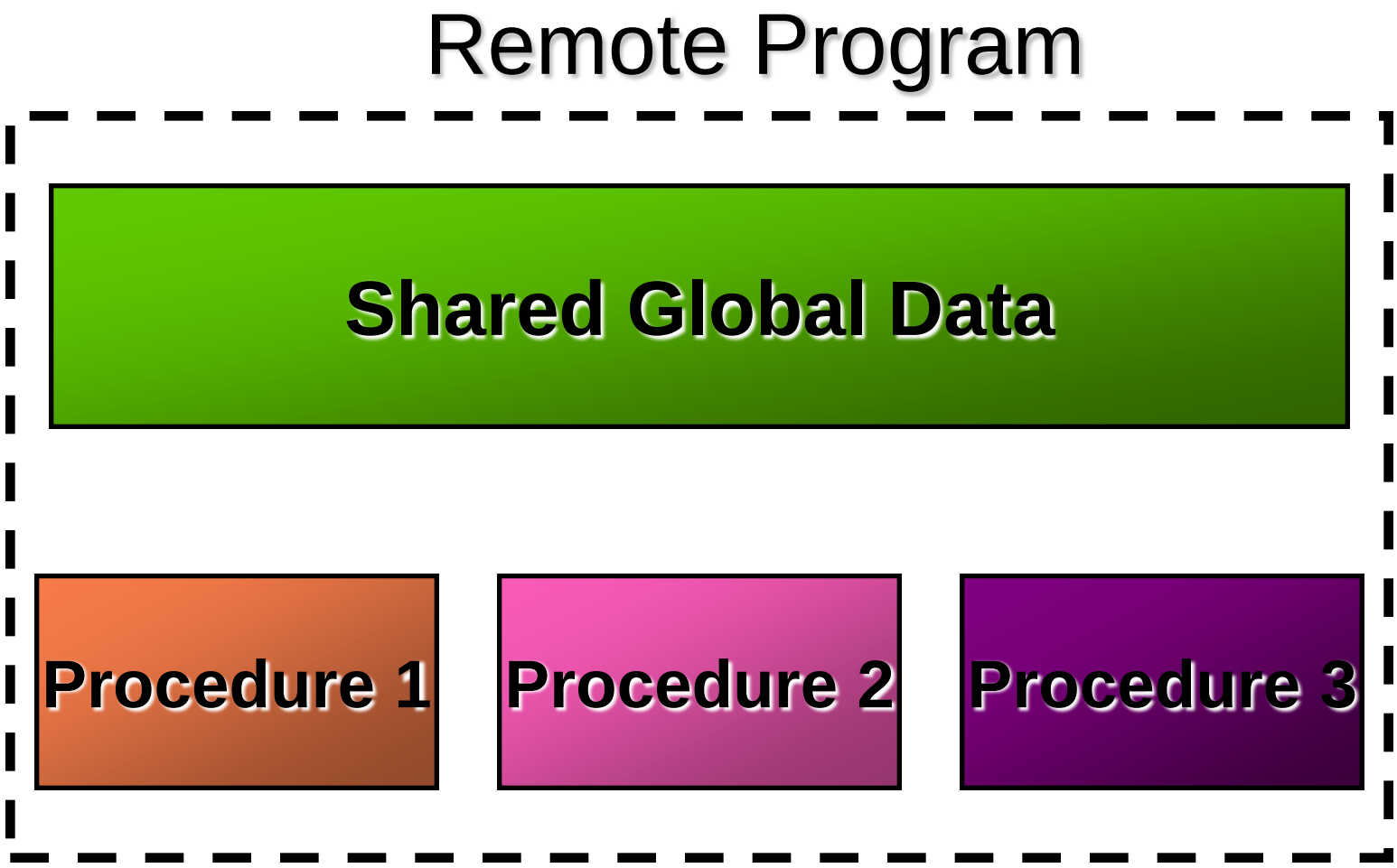
Remote Subroutine



Sun RPC

- There are a number of popular RPC specifications.
- Sun RPC (ONC RPC) is widely used.
- NFS (Network File System) is RPC based.
- Rich set of support tools.

Sun RPC Organization



Procedure Arguments

innovate achieve lead

- To reduce the complexity of the interface specification,
  - Sun RPC includes support for a single argument to a remote procedure
    - Typically the single argument is a structure that contains a number of values.
    - Newer versions can handle multiple args

Procedure Identification

innovate achieve lead

- Each procedure is identified by:
  - Hostname (IP Address)
  - Program identifier (32 bit integer)
  - Procedure identifier (32 bit integer)
    - usually start at 1 and are numbered sequentially
  - Program version identifies
    - for testing and migration
    - typically start at 1 and are numbered sequentially

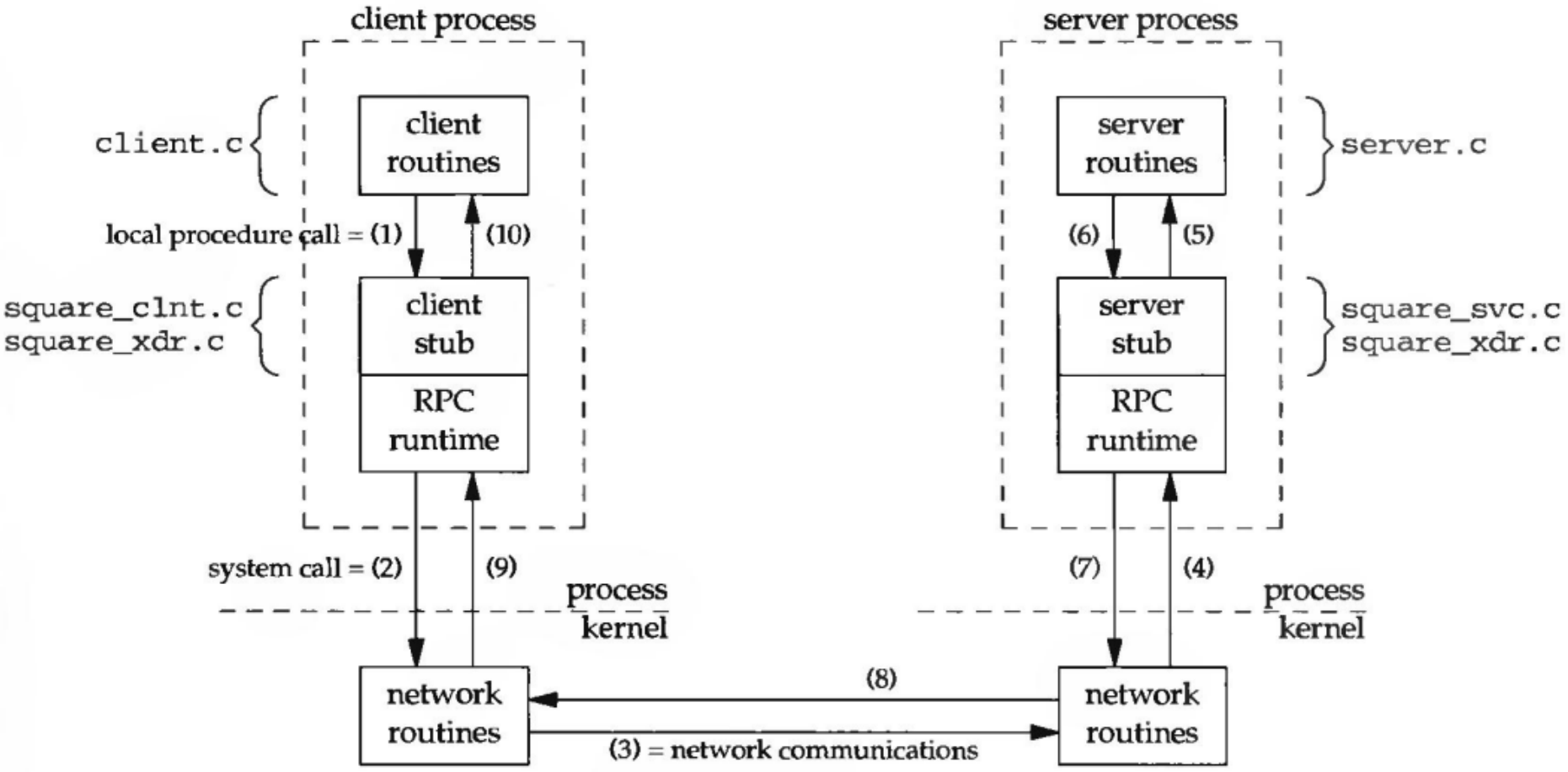
Program Identifiers

- Each remote program has a unique ID.

- Sun divided up the IDs:

0x00000000 – 0x1fffffff	Sun
0x20000000 – 0x3fffffff	SysAdmin
0x40000000 – 0x5fffffff	Transient
0x60000000 – 0xffffffff	Reserved

RPC Subsystem



Iterative Server

innovate achieve lead

- Sun RPC specifies that at *most* one remote procedure within a program can be invoked at any given time.
  - If a 2nd procedure is called, the call blocks until the 1st procedure has completed.

- Having an iterative server is useful for applications that may share data among procedures.
  - Eg: database to avoid insert/delete/modify collisions

Call Semantics

- What does it mean to call a local procedure?
  - the procedure is run exactly one time.

- What does it mean to call a remote procedure?
  - It might not mean "run exactly once"!

Remote Call Semantics

- To act like a local procedure
  - exactly one invocation per call
    - a reliable transport (TCP) is necessary.
- Sun RPC does not support reliable call semantics !
- "At Least Once" Semantics
  - if we get a response (a return value)
- "Zero or More" Semantics
  - if we don't hear back from the remote subroutine

Network Communication

- The actual network communication is nothing new
  - it's just TCP/IP.
- Many RPC implementations are built upon the sockets library.
  - the RPC library does all the work!
- We are just using a different API,
  - the underlying stuff is the same!

Dynamic Port Mapping

- Servers typically do not use well known protocol ports!
- Clients know the Program ID
  - and host IP address
- RPC includes support for looking up the port number of a *remote program.*
  - A port lookup service runs on each host
- RPC servers register
  - "I'm program 17 and I'm looking for requests on port 1736”

The portmapper

- Each system which will support RPC servers runs a *port mapper* server
  - provides a central registry for RPC services
- Servers tell the port mapper what services they offer
- Clients ask a remote port mapper for the port number corresponding to Remote Program ID
- The portmapper is itself an RPC server!
  - is available on a well-known port (111)

RPC Programming

- RPC library is a collection of tools for automating the creation of clients and servers
  - clients are processes that call remote procedures
  - servers are processes that include procedure(s) that can be called by clients
- RPC library
  - XDR routines
  - RPC run time library
    - call rpc service
    - register with portmapper
    - dispatch incoming request to correct procedure
  - Program Generator

RPC Run-time Library

innovate achieve lead

- High- and Low-level functions that can be used by clients and servers
- High-level functions provide simple access to RPC services
  - High-Level RPC library calls support UDP only
    - must use lower-level RPC library functions to use TCP
  - High-Level library calls do not support any kind of authentication

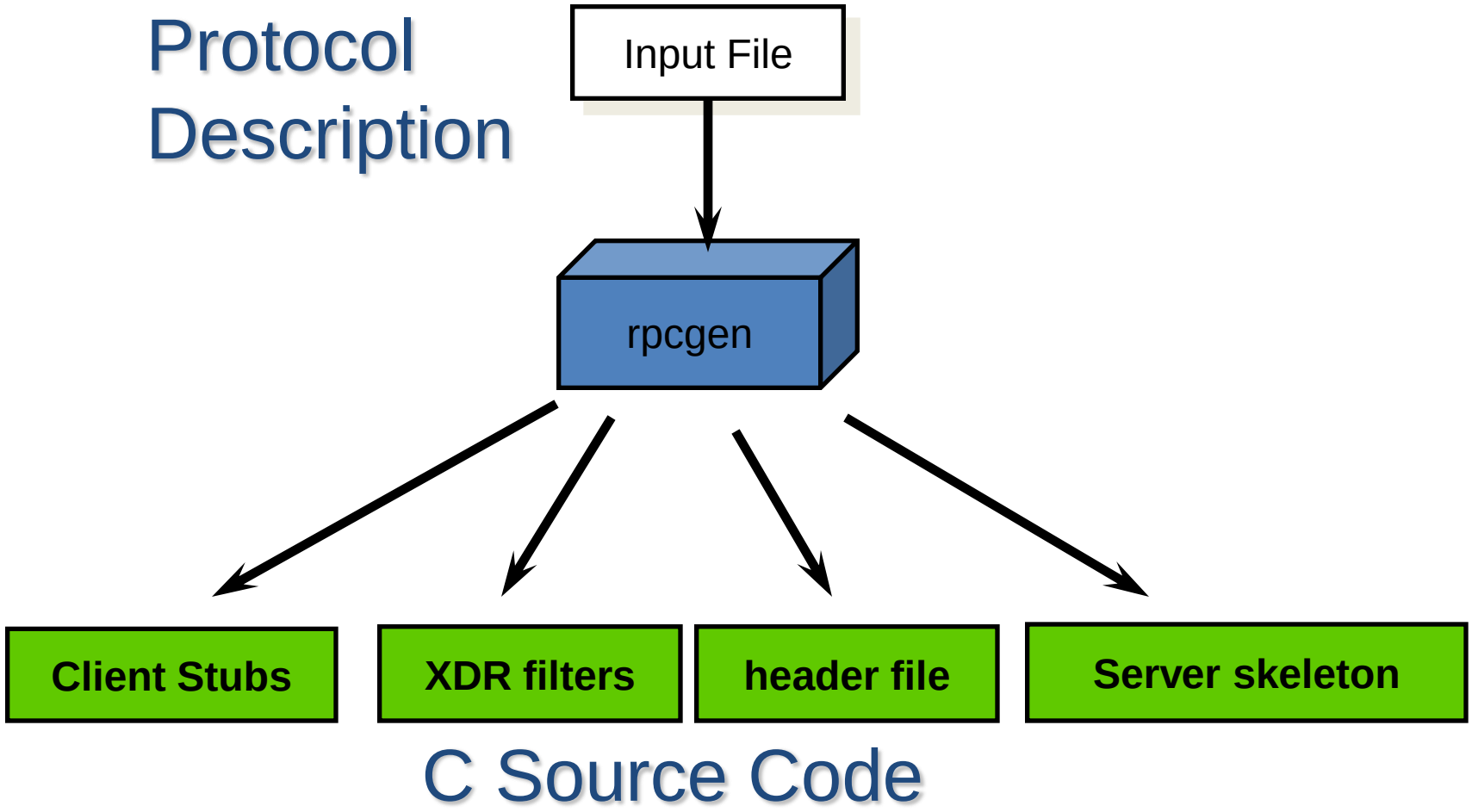
Low-level RPC Library

- Full control over all Inter-Process Communication options
  - TCP & UDP
  - Timeout values
  - Asynchronous procedure calls
- Multi-tasking Servers
- Broadcasting

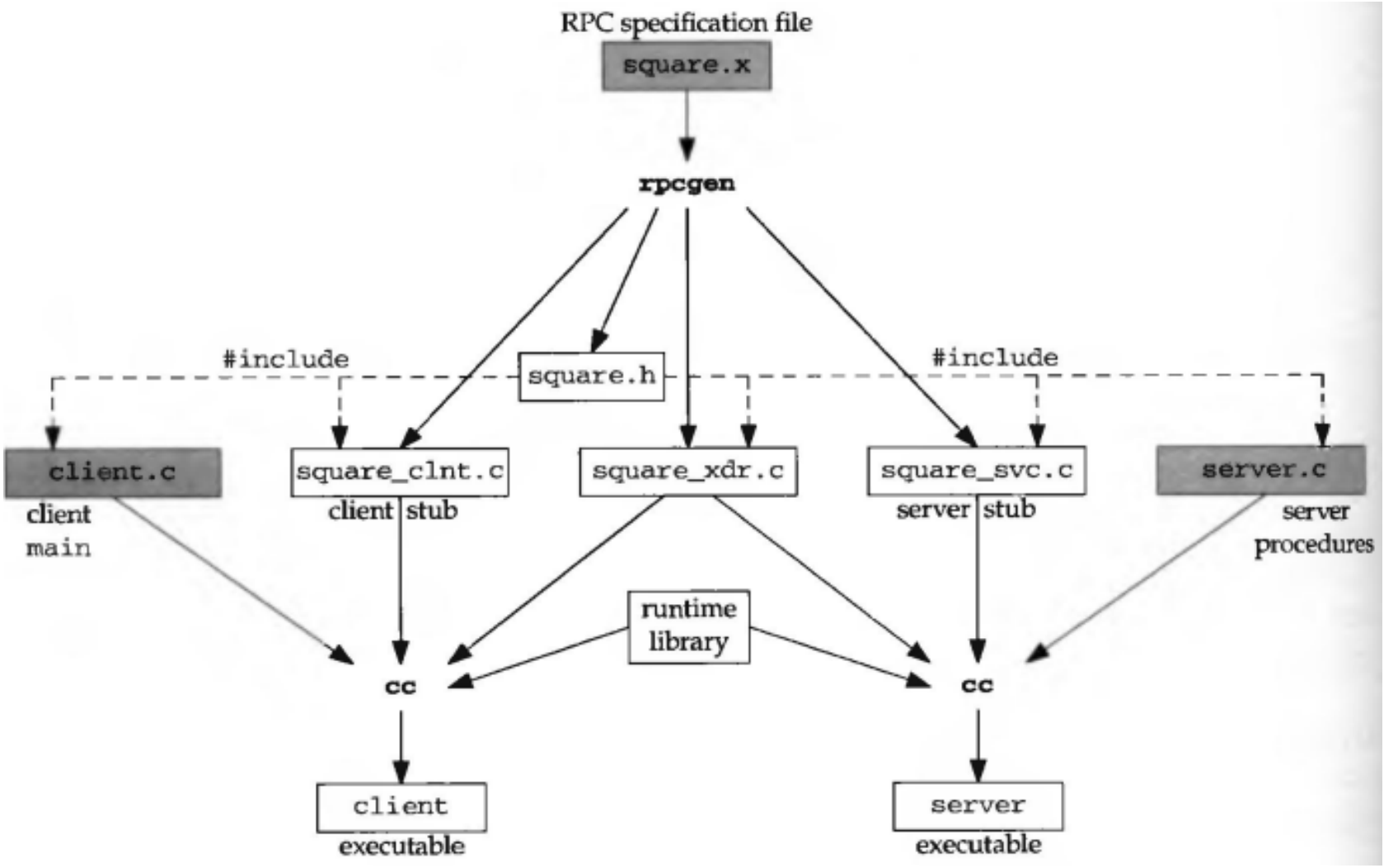
RPCGEN

- There is a tool for automating the creation of RPC clients and servers.
- The program *rpcgen* does most of the work for you
- The input to rpcgen is a *protocol definition*
  - in the form of a list of remote procedures and parameter types

RPCGEN


Protocol Description
Input File
rpcgen
Client Stubs
XDR filters
header file
Server skeleton
C Source Code

RPCGEN


RPC specification file
square.x
rpcgen
#include
square.h
#include
client.c
client
main
square_clnt.c
client stub
square_xdr.c
square_svc.c
server stub
server.c
server
procedures
runtime
library
cc
cc
client
executable
server
executable

rpcgen Output Files

innovate achieve lead

> rpcgen –C foo.x

foo_clnt.c (client stubs)

foo_svc.c (server main)

foo_xdr.c (xdr filters)

foo.h (shared header file)

Client Creation

innovate achieve lead

> gcc -o fooclient foomain.c foo_clnt.c foo_xdr.c -lnsl

- foomain.c is the client main() (and possibly other functions) that call rpc services via the client stub functions in foo_clnt.c

- The client stubs use the xdr functions

Server Creation

gcc -o fooserver fooservices.c foo_svc.c foo_xdr.c –lrpcsvc -lnsl

- fooservices.c contains the definitions of the actual remote procedures.

Example Protocol Definition


```
struct twonums {
   int a;
   int b;
};
program UIDPROG {
   version UIDVERS {
      int   RGETUID(string<20>) = 1;
      string RGETLOGIN( int ) = 2;
      int   RADD(twonums)   = 3;
   } = 1;
} = 0x20000001;
```

RPC Programming with rpcgen

Issues:

- Protocol Definition File
- Client Programming
  - Creating an "RPC Handle" to a server
  - Calling client stubs
- Server Programming
  - Writing Remote Procedures

Protocol Definition File

- Description of the *interface* of the remote procedures.
  - Almost function prototypes
- Definition of any data structures used in the calls (argument types & return types)
- Can also include shared C code (shared by client and server).

Acknowledgements

Q&A

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You