PART XIX

PRIVATE NETWORK INTERCONNECTION (NAT AND VPN)

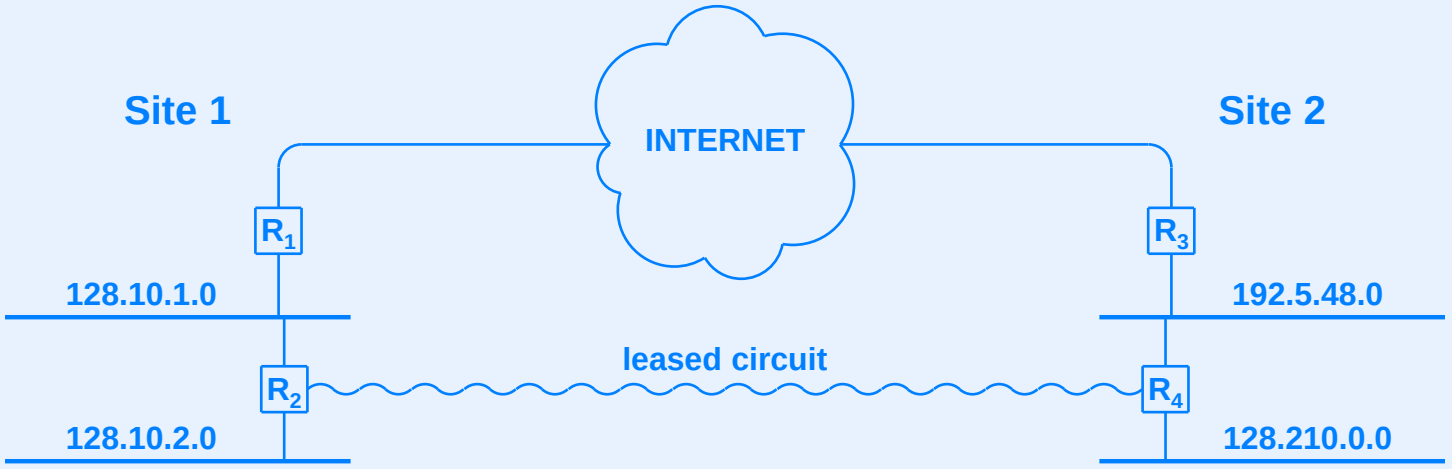
Definitions

- An internet is *private* to one group (sometimes called *isolated*) if none of the facilities or traffic is accessible to other groups
  - Typical implementation involves using leased lines to interconnect routers at various sites of the group
- The global Internet is *public* because facilities are shared among all subscribers

Hybrid Architecture

- Permits some traffic to go over private connections
- Allows contact with global Internet

Example Of Hybrid Architecture



The Cost Of Private And Public Networks

- Private network extremely expensive
- Public Internet access inexpensive
- Goal: combine safety of private network with low cost of global Internet

Question

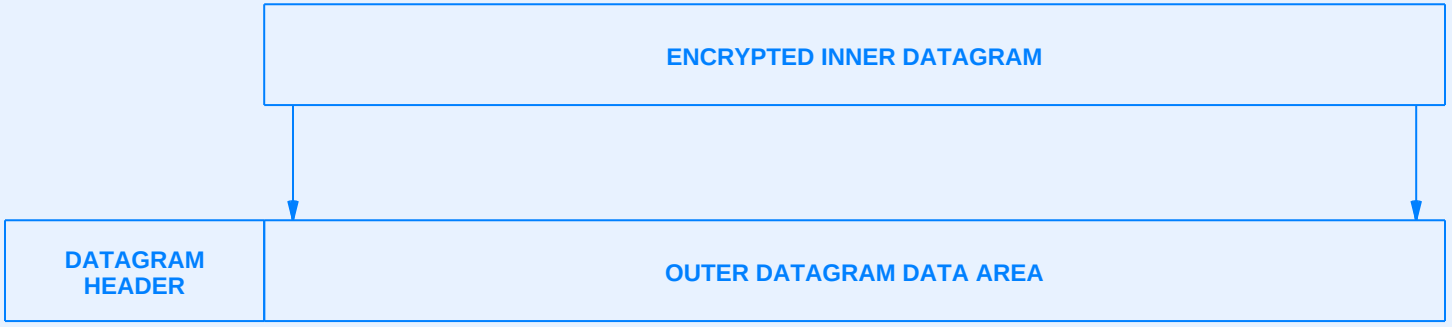
How can an organization that uses the global Internet to connect its sites keep its data private?

- Answer: *Virtual Private Network* (*VPN*)

Virtual Private Network

- Connect all sites to global Internet
- Protect data as it passes from one site to another
  - Encryption
  - IP-in-IP tunneling

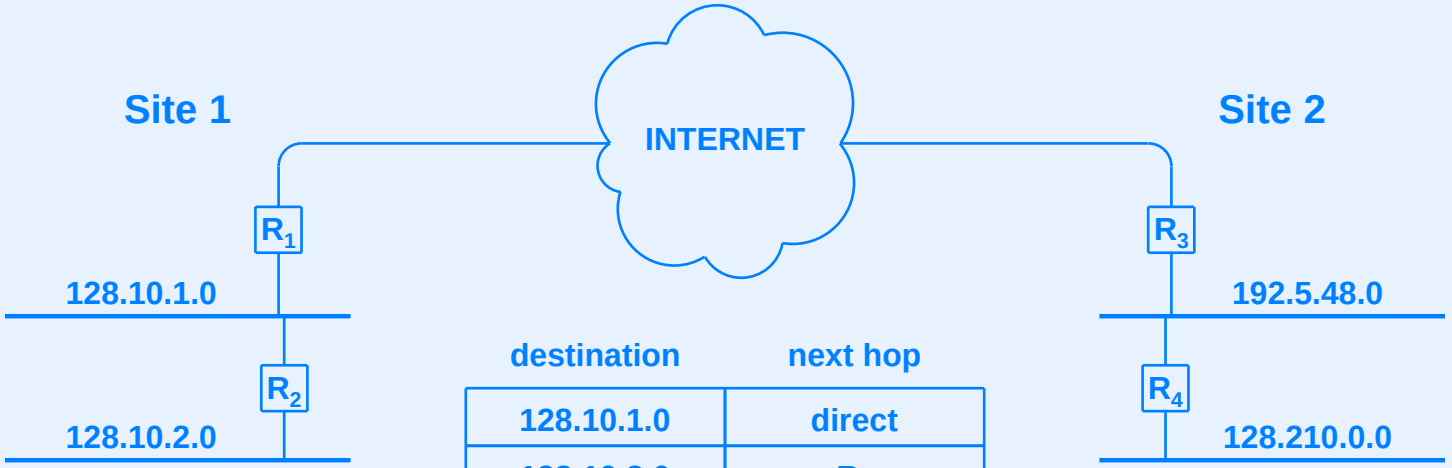
Illustration Of Encapsulation Used With VPN



The Point

A Virtual Private Network sends data across the Internet, but encrypts intersite transmissions to guarantee privacy.

Example Of VPN Addressing And Routing



destination	next hop
128.10.1.0	direct
128.10.2.0	R_2
192.5.48.0	tunnel to R_3
128.210.0.0	tunnel to R_3
default	ISP's router

Routing table in R_1

Example VPN With Private Addresses

- Advantage: only one globally valid IP address needed per site

General Access With Private Addresses

- Question: how can a site provide multiple computers at the site access to Internet services without assigning each computer a globally-valid IP address?
- Two answers
  - Application gateway (one needed for each service)
  - *Network Address Translation* (*NAT*)

Network Address Translation (NAT)

- Extension to IP addressing
- IP-level access to the Internet through a single IP address
- Transparent to both ends
- Implementation
  - Typically software
  - Usually installed in IP router
  - Special-purpose hardware for highest speed

Network Address Translation (NAT) (continued)

- Pioneered in Unix program *slirp*
- Also known as
  - *Masquerade* (Linux)
  - *Internet Connection Sharing* (Microsoft)
- Inexpensive implementations available for home use

NAT Details

- Organization
  - Obtains one globally valid address per Internet connection
  - Assigns nonroutable addresses internally (net 10)
  - Runs NAT software in router connecting to Internet
- NAT
  - Replaces source address in outgoing datagram
  - Replaces destination address in incoming datagram
  - Also handles higher layer protocols (e.g., pseudo header for TCP or UDP)

NAT Translation Table

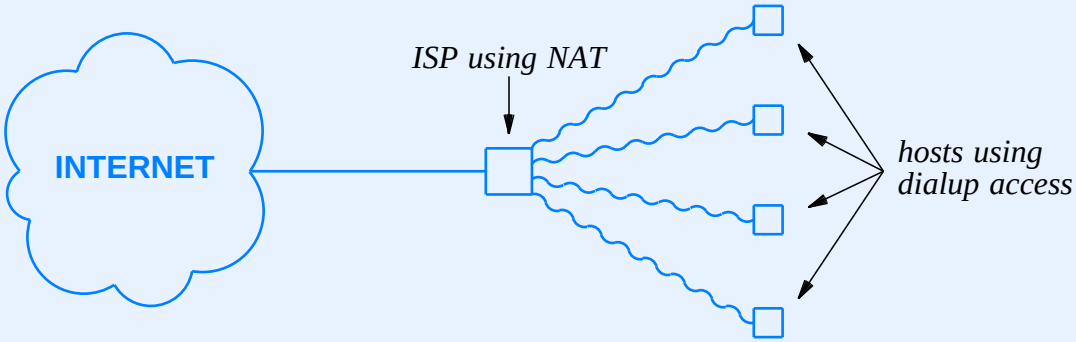
- NAT uses translation table
- Entry in table specifies local (private) endpoint and global destination.
- Typical paradigm
  - Entry in table created as side-effect of datagram leaving site
  - Entry in table used to reverse address mapping for incoming datagram

Example NAT Translation Table

Private Address	Private Port	External Address	External Port	NAT Port	Protocol Used
10.0.0.5	21023	128.10.19.20	80	14003	tcp
10.0.0.1	386	128.10.19.20	80	14010	tcp
10.0.2.6	26600	207.200.75.200	21	14012	tcp
10.0.0.3	1274	128.210.1.5	80	14007	tcp

- Variant of NAT that uses protocol port numbers is known as *Network Address and Port Translation* (*NAPT*)

Use Of NAT By An ISP



Higher Layer Protocols And NAT

- NAT must
  - Change IP headers
  - Possibly change TCP or UDP source ports
  - Recompute TCP or UDP checksums
  - Translate ICMP messages
  - Translate port numbers in an FTP session

Applications And NAT

NAT affects ICMP, TCP, UDP, and other higher-layer protocols; except for a few standard applications like FTP, an application protocol that passes IP addresses or protocol port numbers as data will not operate correctly across NAT.

Summary

- Virtual Private Networks (VPNs) combine the advantages of low cost Internet connections with the safety of private networks
- VPNs use encryption and tunneling
- Network Address Translation allows a site to multiplex communication with multiple computers through a single, globally valid IP address.
- NAT uses a table to translate addresses in outgoing and incoming datagrams


Questions?

PART XX

CLIENT-SERVER MODEL OF INTERACTION

Client-Server Paradigm

- Conceptual basis for virtually all distributed applications
- One program initiates interaction to which another program responds
- Note: ‘‘peer-to-peer’’ applications use client-server paradigm internally

Definitions

- Client
  - Any application program
  - Contacts a server
  - Forms and sends a request
  - Awaits a response
- Server
  - Usually a specialized program that offers a service
  - Awaits a request
  - Computes an answer
  - Issues a response

Server Persistence

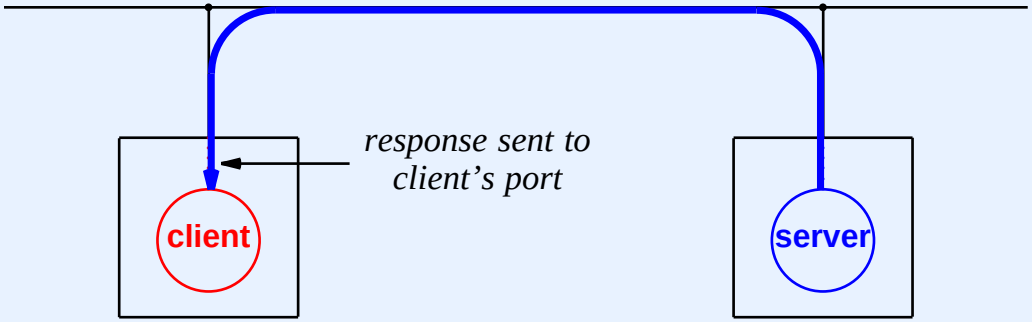
A server starts execution before interaction begins and (usually) continues to accept requests and send responses without ever terminating. A client is any program that makes a request and awaits a response; it (usually) terminates after using a server a finite number of times.

Illustration Of The Client-Server Paradigm

Client sends request

Illustration Of The Client-Server Paradigm

Client sends request



Server sends response

Use Of Protocol Ports

A server waits for requests at a well-known port that has been reserved for the service it offers. A client allocates an arbitrary, unused, nonreserved port for its communication.

Client Side

- Any application program can become a client
- Must know how to reach the server
  - Server's Internet address
  - Server's protocol port number
- Usually easy to build

Server Side

- Finds client's location from incoming request
- Can be implemented with application program or in operating system
- Starts execution before requests arrive
- Must ensure client is authorized
- Must uphold protection rules
- Must handle multiple, concurrent requests
- Usually complex to design and build

Concurrent Server Algorithm

- Open well-known port
- Wait for next client request
- Create a new socket for the client
- Create thread / process to handle request
- Continue with *wait* step

Complexity Of Servers

Servers are usually more difficult to build than clients because, although they can be implemented with application programs, servers must enforce all the access and protection policies of the computer system on which they run and must protect themselves against all possible errors.

Summary

- Client-server model is basis for distributed applications
- Server is specialized, complex program (process) that offers a service
- Arbitrary application can become a client by contacting a server and sending a request
- Most servers are concurrent


Questions?

PART XXI

THE SOCKET INTERFACE

Using Protocols

- Protocol software usually embedded in OS
- Applications run outside OS
- Need an *Application Program Interface* (*API*) to allow application to access protocols

API

- TCP/IP standards
  - Describe general functionality needed
  - Do not give details such as function names and arguments
- Each OS free to define its own API
- In practice: *socket interface* has become de facto standard API

Socket API

- Defined by U.C. Berkeley as part of BSD Unix
- Adopted (with minor changes) by Microsoft as *Windows Sockets*

Characteristics Of Socket API

- Follows Unix's open-read-write-close paradigm
- Uses Unix's *descriptor* abstraction
  - First, create a socket and receive an integer descriptor
  - Second, call a set of functions that specify all the details for the socket (descriptor is argument to each function)
- Once socket has been established, use *read* and *write* or equivalent functions to transfer data
- When finished, close the socket

Creating A Socket

result = socket(pf, type, protocol)

- Argument specifies protocol family as TCP/IP

Terminating A Socket

close(socket)

- Closing a socket permanently terminates the interaction

Specifying A Local Address For The Socket

bind(socket, localaddr, addrlen)

Connecting A Socket To A Destination Address

connect(socket, destaddr, addrlen)

- Can be used with UDP socket to specify remote endpoint address

Sending Data Through A Socket

send(socket, message, length, flags)

- Note
  - Function *write* can also be used
  - Alternatives exist for connectionless transport (UDP)

Receiving Data Through A Socket

recv(socket, buffer, length, flags)

- Note
  - Function *read* can also be used
  - Alternatives exist for connectionless transport (UDP)

Obtaining Remote And Local Socket Addresses

getpeername(socket, destaddr, addrlen)

and

getsockname(socket, localaddr, addrlen)

Set Maximum Queue Length (Server)

listen(socket, qlength)

- Maximum queue length can be quite small

Accepting New Connections (Server)

newsock = accept(socket, addr, addrlen)

- Note:
  - Original socket remains available for accepting connections
  - New socket corresponds to one connection
  - Permits server to handle requests concurrently

Handling Multiple Services With One Server

- Server
  - Creates socket for each service
  - Calls *select* function to wait for any request
  - Select specifies which service was contacted
- Form of select

  nready = select(ndesc, indesc, outdesc, excdesc, timeout)

Socket Functions Used For DNS

- Mapping a host name to an IP address

  gethostname(name, length)

- Obtaining the local domain

  getdomainname(name, length)

Byte Order Conversion Routines

- Convert between network byte order and local host byte order
- If local host uses big-endian, routines have no effect

```
localshort = ntohs(netshort)
locallong = ntohl(netlong)
netshort = htons(localshort)
netlong = htonl(locallong)
```

IP Address Manipulation Routines

- Convert from dotted decimal (ASCII string) to 32-bit binary value
- Example:

address = inet_addr(string)

Other Socket Routines

- Many other functions exist
- Examples: obtain information about
  - Protocols
  - Hosts
  - Domain name

Example Client Program

```
/* whoisclient.c - main */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

/*------------------------------------------------------------------------
 * Program:     whoisclient
 *
 * Purpose:     UNIX application program that becomes a client for the
 *              Internet "whois" service.
 *
 * Use:         whois hostname username
 *
 * Author:      Barry Shein, Boston University
 *
 * Date:        Long ago in a universe far, far away
 *
 *------------------------------------------------------------------------
 */
```

Example Client Program (Part 2)

```
main(argc, argv)
int argc;                       /* standard UNIX argument declarations  */
char *argv[];
{
  int s;                        /* socket descriptor                    */
  int len;                      /* length of received data              */
  struct sockaddr_in sa;        /* Internet socket addr. structure      */
  struct hostent *hp;           /* result of host name lookup           */
  struct servent *sp;           /* result of service lookup             */
  char buf[BUFSIZ+1];           /* buffer to read whois information     */
  char *myname;                 /* pointer to name of this program      */
  char *host;                   /* pointer to remote host name          */
  char *user;                   /* pointer to remote user name          */

  myname = argv[0];
```

Example Client (Part 3)

```
/*
 * Check that there are two command line arguments
 */
if(argc != 3) {
      fprintf(stderr, "Usage: %s host username\n", myname);
      exit(1);
}
host = argv[1];
user = argv[2];
/*
 * Look up the specified hostname
 */
if((hp = gethostbyname(host)) == NULL) {
      fprintf(stderr,"%s: %s: no such host?\n", myname, host);
      exit(1);
}
/*
 * Put host's address and address type into socket structure
 */
bcopy((char *)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
sa.sin_family = hp->h_addrtype;
```

Example Client (Part 4)

```
/*
 * Look up the socket number for the WHOIS service
 */
if((sp = getservbyname("whois","tcp")) == NULL) {
      fprintf(stderr,"%s: No whois service on this host\n", myname);
      exit(1);
}
/*
 * Put the whois socket number into the socket structure.
 */
sa.sin_port = sp->s_port;
/*
 * Allocate an open socket
 */
if((s = socket(hp->h_addrtype, SOCK_STREAM, 0)) < 0) {
      perror("socket");
      exit(1);
}
```

Example Client (Part 5)

```
  /*
   * Connect to the remote server
   */
  if(connect(s, &sa, sizeof sa) < 0) {
        perror("connect");
        exit(1);
  }
  /*
   * Send the request
   */
  if(write(s, user, strlen(user)) != strlen(user)) {
        fprintf(stderr, "%s: write error\n", myname);
        exit(1);
  }
  /*
   * Read the reply and put to user's output
   */
  while( (len = read(s, buf, BUFSIZ)) > 0)
        write(1, buf, len);
  close(s);
  exit(0);
}
```

Example Server Program

```
/* whoisserver.c - main */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <pwd.h>

/*------------------------------------------------------------------------
 * Program:     whoisserver
 *
 * Purpose:     UNIX application program that acts as a server for
 *              the "whois" service on the local machine.  It listens
 *              on well-known WHOIS port (43) and answers queries from
 *              clients.  This program requires super-user privilege to
 *              run.
 *
 * Use:         whois hostname username
 *
```

Example Server (Part 2)

```
 * Author:      Barry Shein, Boston University
 *
 * Date:        Long ago in a universe far, far away
 *
 *-----------------------------------------------------------------------
 */

#define BACKLOG         5       /* # of requests we're willing to queue */
#define MAXHOSTNAME     32      /* maximum host name length we tolerate */

main(argc, argv)
int argc;                       /* standard UNIX argument declarations  */
char *argv[];
{
  int s, t;                     /* socket descriptors                   */
  int i;                        /* general purpose integer              */
  struct sockaddr_in sa, isa;   /* Internet socket address structure    */
  struct hostent *hp;           /* result of host name lookup           */
  char *myname;                 /* pointer to name of this program      */
  struct servent *sp;           /* result of service lookup             */
  char localhost[MAXHOSTNAME+1];/* local host name as character string  */
```

Example Server (Part 3)

```
myname = argv[0];
/*
 * Look up the WHOIS service entry
 */
if((sp = getservbyname("whois","tcp")) == NULL) {
      fprintf(stderr, "%s: No whois service on this host\n", myname);
      exit(1);
}
/*
 * Get our own host information
 */
gethostname(localhost, MAXHOSTNAME);
if((hp = gethostbyname(localhost)) == NULL) {
      fprintf(stderr, "%s: cannot get local host info?\n", myname);
      exit(1);
}
```

Example Server (Part 4)

```
/*
 * Put the WHOIS socket number and our address info
 * into the socket structure
 */
sa.sin_port = sp->s_port;
bcopy((char *)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
sa.sin_family = hp->h_addrtype;
/*
 * Allocate an open socket for incoming connections
 */
if((s = socket(hp->h_addrtype, SOCK_STREAM, 0)) < 0) {
      perror("socket");
      exit(1);
}
/*
 * Bind the socket to the service port
 * so we hear incoming connections
 */
if(bind(s, &sa, sizeof sa) < 0) {
      perror("bind");
      exit(1);
}
```

Example Server (Part 5)

```
  /*
   * Set maximum connections we will fall behind
   */
  listen(s, BACKLOG);
  /*
   * Go into an infinite loop waiting for new connections
   */
  while(1) {
        i = sizeof isa;
        /*
         * We hang in accept() while waiting for new customers
         */
        if((t = accept(s, &isa, &i)) < 0) {
          perror("accept");
          exit(1);
        }
        whois(t);           /* perform the actual WHOIS service */
        close(t);
  }
}
```

Example Server (Part 6)

```
/*
 * Get the WHOIS request from remote host and format a reply.
 */
whois(sock)
int sock;
{
  struct passwd *p;
  char buf[BUFSIZ+1];
  int i;

  /*
   * Get one line request
   */
  if( (i = read(sock, buf, BUFSIZ)) <= 0)
        return;
  buf[i] = '\0';        /* Null terminate */
```

Example Server (Part 7)

```
  /*
   * Look up the requested user and format reply
   */
  if((p = getpwnam(buf)) == NULL)
        strcpy(buf,"User not found\n");
  else
        sprintf(buf, "%s: %s\n", p->pw_name, p->pw_gecos);
  /*
   * Return reply
   */
  write(sock, buf, strlen(buf));
  return;
}
```

Summary

- Socket API
  - Invented for BSD Unix
  - Not official part of TCP/IP
  - De facto standard in the industry
  - Used with TCP or UDP
  - Large set of functions
- General paradigm: create socket and then use a set of functions to specify details


Questions?