Overview

- Last time
  - Subnetting and Supernetting
  - Classless addresses and CIDR
  - Hierarchical routing and BGP

- Today
  - Project Overview
  - Internet scale routing and the role of policy

Project Overview

1. Create topology (make udp sockets, one for routing and one for data)
2. Implement routing (control) thread
   - Track neighbors: Send hello packets to your direct neighbors, keep track of neighbors using TTL
   - Implement your routing protocol — e.g., if distance vector, periodically send your routing table to your neighbors; receive theirs and update your table if necessary
3. Implement your forwarding (data) thread
   - When you receive a data packet, check destination
   - If you are destination, receive it
   - If not, check routing table for next hop and forward the packet

Sleeping/Implementing timers

- You will have to use threads to implement your program
  - One thread for control (routing/hello) and one for data (forwarding packets)
- Use `usleep` and it sleeps only one thread (example to follow)
- I am guessing `sleep` works the same way
- Use `usleep`; it gives you finer control over time (microseconds instead of seconds)
- Need to `#include <unistd.h>`

## `usleep()` Example

```c
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>

void *slowthread(void *i) {
    int j=0;
    while(++j) { // print Hi once every 5 seconds
        usleep(5000000);
        printf("Hi%d from slowthread\n",j);
    }
    return NULL;
}

void *fastthread(void *i) {
    int j=0; // print Hi once every 1 second
    while(++j) {
        usleep(1000000);
        printf("Hi%d from fastthread\n",j);
    }
    return NULL;
}

main() {
    pthread_t *thread;
    int dummy;
    pthread_create(thread,NULL,slowthread,(void *) &dummy);
    fastthread((void *) &dummy);
    pthread_exit(NULL);
}
```
Recall Overall structure of Program

- Two threads, one for routing/control and one for forwarding/data
- Routing thread: (1) Exchanges hello messages; (2) Exchanges routing messages; (3) Builds routing table; (4) Interacts with control client (optional)
- Data thread: (1) Receives incoming data packets; (2) Forwards it if we are not destination according to routing table; (3) If you do not implement control client, should with low probability generate packets to random destinations

Routing Thread – Time Driven

- Also runs in a loop
- Use usleep with a suitable period (say 0.1 sec) to control loop interval
- Things happen periodically (e.g., should send hello packets every n seconds); keep a count for every event in terms of the loop granularity (a count of 50 will give you 5 seconds if your period is 0.1 seconds)
- Every loop
  1. Check if you need to generate control packets (hello or routing) based on their count expiring
  2. Check for control packets (hello, or routing) and update your routing table accordingly
- Behavior will vary depending on whether you use distance vector or link state

Data Thread – Event Driven

1. In a loop, send out any locally generated packets
2. Wait for incoming packets (use select, with a reasonable timeout value, say 0.1 sec)
3. If you receive an incoming data packet, check if you need to forward it or receive it locally and do so

Each thread sleeps independently without blocking the other
Hello Packet

- Hello protocol simply keeps hosts aware of their neighbors
  - Each host should have a data structure (array or linked list) of its immediate neighbors
  - Everytime they receive a hello packet from them, they refresh the TTL entry on the neighbor list (lets say starting value is 1000 timer pulses)
  - Everytime the routing thread wakes up reduce TTL by 1, if it reaches 0 remove the neighbor
  - If a link goes down, we stop receiving packets

- What should the packet format be? Hello Opcode and Source Id is enough

Routing Packet

- Depends on your routing protocol
  - Distance Vector: should include your full routing table
    - Example: [Opcode—Source—host1—Cost1—host2—cost2—...]
  - Link State: Send only your neighbor list
    - Example: [Opcode—Source—SequenceNumber—Neighbor1—Neighbor2—.. .]
    - Note that you have to flood this (send it to all other neighbors) the first time you receive this sequence number

- When your receive a routing packet, you should update your routing table according to your protocol

Packets and Bits

- Three types of packets:
  - Data packets do the work of carrying the data.
  - Hello packets (or heartbeat packets) keep neighbors informed that the link between them exists
  - Routing packets, exchange routing info according to your routing protocol
  - For the optional client part, you have other control packets

- Data packets are received on the data port, everything else on the routing port

Data Packet Format

- Should have fields for:
  - Packet Id, so that we can track it
  - Source and Destination Id
  - Previous hop (this allows us to tell what link it came on since all of the links share the same protocol, id, and sequence number)
  - Protocol Id (necessary to implement upper layer protocols used in bonus parts)
  - Time To Live (TTL)
  - Data

- Like project 1, you can use memcpy to assemble/desemble packets in character buffers to send them over

- You should generate packets periodically to random destinations if you do not implement the optional control client

- Should have fields for:
  - Source Id so that we can track it
  - Previous hop (this allows us to tell what link it came on since all of the links share the same protocol, id, and sequence number)
Recall Select System Call

- Provides a way to wait on an event on a designated set of sockets (or file descriptors)
- Problem
  - If you receive on a blocking socket, and there is no message, you are blocked (the whole process, not just a thread)
  - select provides a way to check if there is a message to avoid this blocking behavior
- It also provides a way to do event-driven processing; you wait until an event happens
  - Careful that select itself blocks – give it a short timeout value
  - may want to have a usleep(1) after the select to make sure the other thread gets a chance to run before the process blocks again (pthreads implementation is not preemptive by default)

Select by Example

```c
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#define STDIN 0 /* file descriptor for standard input */

main()
{
    struct timeval tv; // a structure used to represent time; it has seconds and usec components
    fd_set readfds; // set of file descriptors
    tv.tv_sec = 1;
    tv.tv_usec = 500000;
    FD_ZERO(&readfds); // zero out the set of descriptors
    FD_SET(STDIN, &readfds); // add STDIN to the set of descriptors
    select(STDIN+1, &readfds, NULL, NULL, &tv);
    if (FD_ISSET(STDIN, &readfds))
        printf("A key was pressed!\n");
    else printf("Timed out.\n");
}
```

Poor IP Address Utilization

- Allow multiple networks to share the same “physical network” space
- Two departments with 100 machines
  - Assign 2 class C networks?
  - Use subnetting to share one class C network
    - 128.226.123.xx is the network number
    - CS gets 128.226.123.0 to 128.226.123.127
    - EE gets 128.226.123.128 to 128.226.123.255
  - How?
    - Address mask is used to figure out what network you are on
      - Address Mask is 255.255.255.128
      - Any address in CS ANDED with the mask produces 128.226.123.0
      - Any address in EE ANDED with the mask produces 128.226.123.128

Supernetting

Instead of assigning a corporation that has 400 hosts a class B address, assign it two class C addresses
- If the networks are consecutive they can be abstracted away as a single network
- The combination of subnetting and supernetting gives us classless routing (Classless Internet Domain Routing/CIDR)
- The network portion of the address can be of any length
- Use masks to sort things out
- Use masks to help with routing
- As you get closer to the destination, more details become available
CIDR Routing

- Multi-regional 192.0.0.0 - 193.255.255.255
- Europe 194.0.0.0 - 195.255.255.255
- Others 196.0.0.0 - 197.255.255.255
- North America 198.0.0.0 - 199.255.255.255
- Central/South America 200.0.0.0 - 201.255.255.255
- Pacific Rim 202.0.0.0 - 203.255.255.255
- Others 204.0.0.0 - 207.255.255.255

- How can we make use of this information?
  - Shorthand is: network_number/length in bits
    - Example: 194/7 (Europe above)
    - Example: 128.226.123/24 for a 256 address network
    - Routers understand this shorthand directly

- No classes. Addresses assigned in any power of 2 granularity
  - 198.5.192/12 for a 4096 machine network

BGP – Very Briefly

- Challenges:
  - Scale – still a very large scale (10s of thousands of ASs)
  - How to specify cost metrics? Each AS is independent
  - Trusting advertisements
  - Specifying and enforcing flexible routing policies
  - Integration with Intradomain routing

- BGP – How it works
  - Each AS has a BGP speaker
  - Speaker exchanges reachability information with other speakers (no costs)
  - Distance vector but advertises full path to avoid count to infinity loops (no count since only reachability info is exchanged)
  - AS numbers centrally assigned and unique
  - Speaker can cancel previously advertised paths

Internet Past and Present

- Past – tree structure

- Present – multi backbone

- How to route it?

Example and Discussion

- AS 2 advertises it has a path to 128.96 (AS2, AS4); AS 1 advertises it has a path to 128.96 (AS1, AS2, AS4)

- Discussion
  - How optimal are the paths?
    - Always trying to maintain a balance between scalability and optimality
  - How scalable?
    - Number of ASs much smaller than number of networks
    - Updates of networks in ASs can be summarized as in OSPF
Discussion/Summary

- Routing scalability:
  - Get routing table sizes down – how?
  - Get routing overhead down – how?

- Role of hierarchy and address aggregation

- Commercial factors and relationships, and not performance/technical factors, often play the most important role in how routing works

BGP Table Size

- BGP table size at a sample Backbone routers (courtesy Oregon Route Views Project http://www.antc.uoregon.edu/route-views/)

Policies

- Routing internally and externally strongly influenced by policies. Example:
  - Prefer shortest path
  - Hot Potato Routing (get it off the AS as soon as possible)
- No longer a problem of finding the best path

BGP

- Allows ISPs/administrators to specify policy; but policy is not part of the protocol
- Example policy – hot potato – get the traffic off my network as fast as possible
- Policy shaped by commercial relationships between ISPs (in a second)
- Can lead to strange behavior
  - e.g., asymmetric routes
  - e.g., traffic between UAE and Saudi Arabia was getting routed through Washington DC
Peering vs. Transit

- Peering: Two ISPs provide connectivity for each other’s customers
  - Usually for free
  - Not a transitive relationship
- Transit: One ISP provides another with connectivity to every place it knows
  - $$$

IP in IP (IP tunneling)

- The IP Packet is encapsulated by another IP header with the tunnel address
Routing for Mobile Networks – Last Hop

- Mobile hosts can move to another network
  - What problems does this cause?
- Think about existing connects, and new connections
  - Basic problem: Network routes packets to the home network address
- How can we solve this?
- How does your mail continue to get to you when you move?

Tunneling (cont’d)

- Protocol field is set to IP-in-IP protocol
  - When tunnel destination receives the tunneled packet, it strips the header and sends the original packet on
- Applications?
  - Think of a situation where you can place a mail envelope in another
- For efficiency, an abbreviated IP header is defined for tunnelling

Mobile IP

- One solution – Mobile IP. Idea is, when a host moves:
  - It signs up with Foreign Agent (FA) on new network
  - Foreign Agent (FA) assigns a care-of-address
  - FA informs Home Agent (HA) of care-of-address
  - HA *tunnels* packets destined to the mobile to the care-of-address
  - FA strips tunnel header and delivers packet
Mobile IP

- Any problems?
  - What about hosts on the home network to whom you are talking, any problems once you go away?
  - How efficient? Can it be optimized?

Triangulation

- Consider a source node (S) talking to a mobile node M
  - Packets from S have to be forwarded through home agent
  - Packets from M can be sent directly to S
    * Will this work with ingress (and other) filtering?
  - Route looks like a triangle – inefficient
    * IETF draft proposes solution to this problem
    * Bad: requires changes to S's TCP/IP stack