15-441

Computer Networking

Exam Feedback
Mar. 8, 2006

Traceroute slide incorporated March 24
Pigeon slide added May 10
Further additions not expected

Topics

- reading list
- the traceroute/netmask question
- finger client: errors & myths

Synchronization

Textbook

- Relevant now
  - Section 2.5 (Reliable Transfer)
  - Chapter 5: Transport (ok if you read 5.3 lightly)
  - Chapter 6: Congestion Control
- Looking Backward / Forward
  - Section 3.3 (ATM)
  - Section 4.4 (Multicast), 4.5 (MPLS)
  - Section 9.1 (DNS)

Outline

The pigeon question

The netmask question

The finger question

Myths

The Pigeon Question

Bob...

- ...inspects and makes decisions based on IP addresses.
- ...is a network-layer entity, typically called a router.

Kelly...

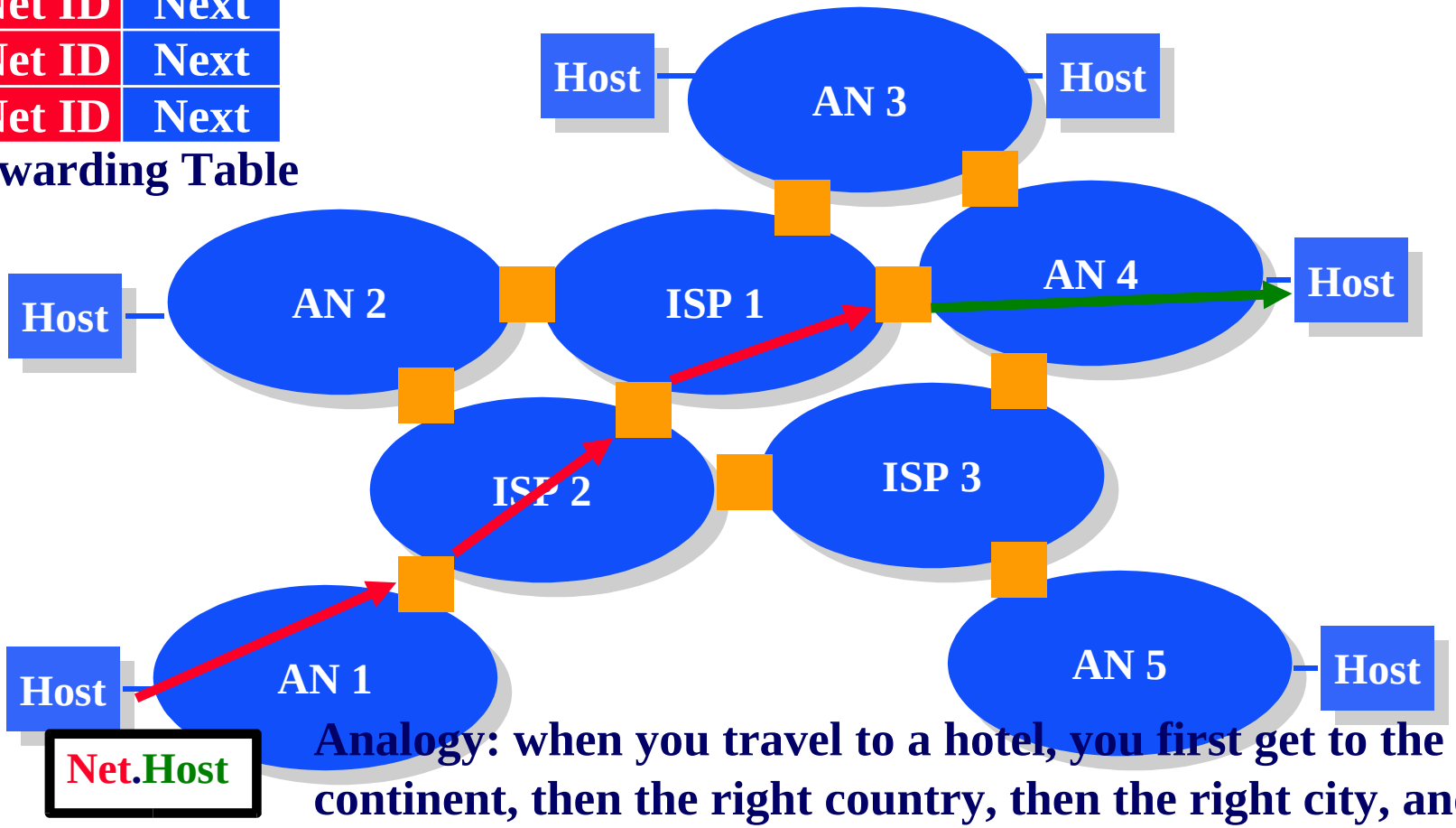
- ...“strengthens” packets which are “tired”
  - Kelly *could* be a repeater, but...Kelly...
- ...also decides where packets should go
  - Kelly can't be a physical-layer device
    - » Kelly's decisions are based on something “outside of” the IP addresses
      - o Kelly must be a link-layer device – in this case, a bridge.

Size of Netmask Increases

As you get closer to the destination, the set of hosts in your target area becomes smaller, i.e. the Net ID growns and the Host ID shrinks.

Net ID	Next
Net ID	Next
Net ID	Next
Net ID	Next

Forwarding Table



Analogy: when you travel to a hotel, you first get to the right continent, then the right country, then the right city, and finally, the right hotel.

Size of Netmask Increases

Cases

- Departing from a host with only one link
  - That link probably has a “default route” entry
    - » 0.0.0.0/0
- Heading toward “the backbone”
  - Probably more default-route entries (mask stays 0 bits long)
- Departing “the backbone” for target's ISP
  - One entry probably covers the ISP's address space
    - » x.y/16
- Departing the ISP to the target organization's network
  - One entry probably covers the target organization
    - » x.y.z/24
- Arriving at the destination host via point-to-point link
  - The table entry for that link on the penultimate host is like
    - » x.y.z.w/32

finger

Problem

- Here is a finger client
  - Connect to TCP port 79
  - send username
  - print out server's response
- Say what's wrong
  - This was a “target-rich environment”

finger.c

```
int main(int argc, char *argv[])
{
  int s, len;
  struct sockaddr_in server;
  struct hostent *hp;
  char c, buf[8192];

  if (argc != 3) {
    fprintf(stderr, "usage: %s host user\n", argv[0]);
    exit(9);
  }
  server.sin_family = AF_INET;
  server.sin_port = 79;
  server.sin_addr.s_addr = gethostbyname(argv[1]);
  s = socket(AF_INET, SOCK_DGRAM, 0);
  bind(s, (struct sockaddr *) &server, sizeof (server));
  write(s, argv[2], strlen(argv[2]));
  write(s, "\r\n", 2);
  if ((len = read(s, buf, sizeof (buf))) > 0)
    write(1, buf, len);
  exit(0);
}
```

finger.c

```
int main(int argc, char *argv[])
{
  int s, len;
  struct sockaddr_in server;
  struct hostent *hp;
  char c, buf[8192];

  if (argc != 3) {
    fprintf(stderr, "usage: %s host user\n", argv[0]);
    exit(9);
  }
  server.sin_family = AF_INET;
  server.sin_port = 79;
  server.sin_addr.s_addr = gethostbyname(argv[1]);
  s = socket(AF_INET, SOCK_DGRAM, 0);
  bind(s, (struct sockaddr *) &server, sizeof (server));
  write(s, argv[2], strlen(argv[2]));
  write(s, "\r\n", 2);
  if ((len = read(s, buf, sizeof (buf))) > 0)
    write(1, buf, len);
  exit(0);
}
```

finger.c

```
server.sin_family = AF_INET;
server.sin_port = 79;
server.sin_addr.s_addr = gethostbyname(argv[1]);
s = socket(AF_INET, SOCK_DGRAM, 0);
bind(s, (struct sockaddr *) &server, sizeof
 (server));
write(s, argv[2], strlen(argv[2]));
write(s, "\r\n", 2);
if ((len = read(s, buf, sizeof (buf))) > 0)
  write(1, buf, len);
```

Pretty much all of this is wrong

finger.c

```
server.sin_family = AF_INET;
server.sin_port = 79;
server.sin_addr.s_addr = gethostbyname(argv[1]);
s = socket(AF_INET, SOCK_DGRAM, 0);
bind(s, (struct sockaddr *) &server, sizeof
 (server));
write(s, argv[2], strlen(argv[2]));
write(s, "\r\n", 2);
if ((len = read(s, buf, sizeof (buf))) > 0)
  write(1, buf, len);
```

finger.c

Bad

```
server.sin_port = 79;
```

Good

```
server.sin_port = htons(79);
```

Bad

```
server.sin_addr.s_addr = gethostbyname(argv[1]);
```

Good

```
hp = gethostbyname(argv[1]);
memmove(&server.sin_addr, hp->h_addr, hp->h_length);
```

finger.c

Bad

```
s = socket(AF_INET, SOCK_DGRAM, 0);
```

Good

```
s = socket(AF_INET, SOCK_STREAM, 0);
```

Bad

```
bind(s, (struct sockaddr *) &server, sizeof
 (server));
```

Good

```
connect(s, (struct sockaddr *) &server, sizeof
 (server));
```

finger.c

Bad

```
  if ((len = read(s, buf, sizeof (buf))) > 0)
    write(1, buf, len);
```

Good

```
  while ((len = read(s, buf, sizeof (buf))) > 0)
    write(1, buf, len);
```

Myths

Must close sockets before exit()

- If that were true we'd all be in big trouble!
- exit()'s *job* is to clean up process resources

`sizeof(buf) == 4`

- That's *like* a real problem...
  - sizeof (pretty much any pointer) == 4 (on many machines)
  - sizeof (*array*) is, well, the size of the array, in bytes
    - » "Doesn't work" for array parameters to a function
    - » They're actually pointers (call by reference), not arrays

`write(stdout, ...)`

- That's mixing metaphors – file descriptors aren't stdio streams
- You could write `write(fileno(stdout), ...)`
- But if fileno(stdout) != 1 something very very odd is going on

Myths

Cannot use write() and read() on UDP sockets

- Sure you can!

read() doesn't block to wait for server response

- Yes, it does!

strings must be converted to network byte order

- The network byte order for strings is:
  - Send the first byte, then the second, then the third...
- “Byte order” is a problem when you have N-byte chunks
  - Integer is a 4-byte chunk
- You could have a string byte-order problem with Unicode
  - Out of scope

Myths

Buffer overflows!

```
write(s, argv[2], strlen(argv[2]));
```

- *We* aren't putting anything into a buffer!
  - Certainly not one of fixed size, without a length check
- The kernel might be putting these bytes in a buffer
  - If the kernel does that unsafely we have problems beyond finger
- The finger server might carelessly handle this request
  - But we can't save it from *other people* triggering that

```
read(s, buf, sizeof (buf))
```

- Ok, this is a buffer
- But we are very carefully *not* overflowing it!
  - If the kernel puts more than sizeof (buf) bytes into buf then we have problems bigger than finger

Not all buffer uses are buffer overflows!