**Connection-Oriented Nets**

VC # from A to D is 8, 92, 8, 6

**X.25**

- X.25 is just the interface to the network
- The only thing specified in standard is what goes on between DTE and DCE
- Three levels of standard
  - physical level — X.21
  - Frame level — HDLC variant
  - Packet level

**X.25 Packet Level**

**X.25 Call Request Packet**

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>type (call request=00001011)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>length calling addr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>called address</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>facilities length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>facilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>user data</td>
</tr>
</tbody>
</table>
**X.25 Data Packet**

<table>
<thead>
<tr>
<th>Q</th>
<th>D</th>
<th>Modulo</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ack #</td>
<td>M</td>
<td>Msg #</td>
<td>0</td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bigger Sequence Numbers

<table>
<thead>
<tr>
<th>Q</th>
<th>D</th>
<th>Modulo</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Msg #</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ack #</td>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**X.25 Message Numbers**

```
size x
dte ← dce
```
```
size 2x
dce ← dte
```

**ATM (Asynchronous Transfer Mode)**

- Datagram, but connection-oriented
- 48 byte cells, 5 byte (40 bit) header
- Header: 28 vc # (inside net, endnode sees 24 bits and 4 bits “generic flow control” (unused), 3 payload type (congestion, AAL5 eop, 1 cell loss priority (drop me), 8 bits hdr CRC

<table>
<thead>
<tr>
<th>gfc or VPI</th>
<th>VPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPI</td>
<td>VCI</td>
</tr>
<tr>
<td>VCI</td>
<td>payload type</td>
</tr>
<tr>
<td>header CRC</td>
<td></td>
</tr>
</tbody>
</table>

**VPI/VCI**

- Within left net, route based on entire VPI+VCI
- Within right net, route based solely on VPI
- Many VC’s multiplexed over one VP
- Smaller table size within backbone
**MPLS (Multiprotocol Label Swapping)**

- Basically same as ATM
- Add 32-bit ATM-like hdr on top of IP packet
  - 20 bit label
  - 3 bits unused
  - 1 bit “bottom of stack”
  - 8 bits TTL (hop count)
- stacked labels generalization of VPI/VCI
- Originally for faster forwarding than parsing IP header
- Later, “traffic engineering”
- Classify packets based on more than destination address

**ATM Adaptation Layers**

- 48 byte cells are not a convenient API
- AAL is a layer on top that makes ATM more convenient to the software
- AAL type 1 is constant bit rate. Application just sends a stream of bits. AAL sends it in 47 byte chunks. Extra byte is a 4 bit sequence number and 4 bit error check (3 bit CRC, one parity) on the sequence number.
- AAL type 5 adds a 4-byte CRC to a civilized size (up to 64K bytes) packet

**Connectionless Network Layers**

- Destination, Source addresses
- Fragmentation and reassembly info (in some)
- Options (source routing)
- Service requests (priority, special route calculation)
- Error reports
- Congestion indication
- Real difference between all the different “protocols” is size of address

**Addresses**

- 802 addresses have no geographic hints — like routing to social security number — known as “flat address”
- Generic hierarchical address: locator.node

| locator | node |

- IP, IPX, AppleTalk: locator is specific to a LAN
- CLNP, DECnet Ph4, (maybe IPv6) locator is entire region called an “area” — could be single LAN but can be bigger
Comparative Addresses

- **IP**: 4 bytes, boundary depends on mask
- **IPX**: 4 bytes, 6 bits area, 10 bits node
- **DECnet Ph IV**: 2 bytes total
- **AppleTalk**: 2 bytes
- **CLNP**: up to 14 bytes, 6 bytes
- **IPv6**: 8 bytes

Hierarchy within Locator

- The locator can have arbitrary many levels of hierarchy
- Assume addresses like this:
  - USMASSACTON.Radia
  - USCALSANJOSE.George
  - USSRRUSSIAMOSCOW.Boris
- Each level of hierarchy hides inside detail

Longest Matching Address Prefix

- Routing based on locator field is actually to longest matching address prefix
- We’ll discuss algorithms for doing this later
- How can you encode a prefix
  - IP way: address, mask
    - address: 10011101 11010010 00100000 00110101
    - mask: 11111111 11111111 11100000 00000000
    - CLNP way: length of prefix, prefix
    - AppleTalk way: address range (lo, hi)

Comparisons

- **Range**
  - most flexible -- doesn’t need to start and end on powers of 2
  - requires twice address in size
- **Mask**
  - allows non-contiguous subnet masks (not a feature, actually)
  - requires twice address in size
- **Length**
  - most compact encoding: address plus log address, or can be even more compact by not saying entire address when prefix is short
  - as flexible as mask, other than non-contiguous subnet masks
Address Assignment

- Own or rent addresses?
- Central administration (IANA) where you own address doesn’t allow aggregation
- Better for routing if you “rent” address, but then you have to renumber if you change providers
- People early on assumed they owned IP addresses, makes things difficult now
- It ought not to be that hard to renumber, but it is in IP
- The bigger the address the more manageable the problem

IP Addresses

- Ought to be (address, mask) but need to know about classes
- Originally the thought was that the address would consist of 1 byte of “net”, 3 bytes of “address within net”. This made sense when a “net” was something like the ARPANET. There weren’t very many “networks”.
- Then along came “Ethernet”. It had the word “net” in the title, so the assumption was that a net # would need to be assigned to each Ethernet
- Too many Ethernets (more than 256), so the following classes were invented:
  - class A: high bit 0, boundary after byte 1
  - class B: high 2 bits 10, boundary after 2
  - class C: high 3 bits 110, boundary after 3

IP Masks

- That’s just the historical way they handed out blocks of addresses.
- CIDR: Classless InterDomain Routing — just says always route to (address, mask) pair, ignore the class stuff
- Until routing protocols that infer mask based on class go away, we’re stuck with learning about classes
  - RIP
  - EGP
- Can have multiple levels of hierarchy: routers on outside of region advertise a shorter mask for the region, which summarizes all the addresses within the region

IPX Addresses

- Much bigger than IP (10 bytes vs 4), but 6 bytes for node, so locator is only 4 bytes (about 256 times as large as IP)
- 6 byte ID makes autoconfiguration trivial and removes overhead of mapping from layer 3 to layer 2 address
Nifty Idea

- Add 6 bytes to IPX address, which are autoconfigured into routers. Call that the “domain”
- Number all your routers once with 4 byte IPX “net number”. Learn “domain” number from router connected to “backbone”
- If move, domain changes, but that’s autoconfigured
- IPv6 may do similar things

| domain | net | node |

Interesting Privacy Issue

- From point of view of network protocol, IPX is ideal. Autoconfiguration trivial. no protocol needed for mapping from layer 3 to layer 2. (In IP this is done by ARP protocol, which we’ll get to)
- Recent privacy concern with visible unique IDs: can track location of computer, and perhaps human (with laptop that keeps same EUI-48)
- IETF wants the ability to hide such info
- Could be done with something that dynamically hands out “ID” portion of address (from local space)
- That means layer 3/layer 2 address mapping protocol required, at least when ID portion from local space

CLNP Addresses

- ISO hands out blocks of addresses to subauthorities, so a little mom and pop store in Brazil won’t have to go to ISO
- It’s easy to get a unique CLNP prefix
  - any telephone number will do
  - any point of attachment to a public provider net (using E.164 or X.121 addresses) will do
  - other types of addresses as well
- The “address formats” make it sound complicated, but all you really need to know is you get a prefix for your entire network, and then assign addresses from there, with a 6 byte ID field in the low order part of the address. All routers need to know is the locator is a pile of bits for which they route to longest matching prefix

My Favorite Use of CLNP

- Assume you connect branches of your network with a public provider net, say an SMDS or X.25 net
- Each branch uses its point of attachment as its prefix
**CLNP Addresses**

- All nodes’ addresses in R3’s portion would begin with δ, such as δ.S. If more hierarchy is required, then it would be δ.area.S.

- R3 advertises into its net either * or something indicating it belongs on that backbone net.

- The backbone just needs to route to δ, which it already knows how to do.

- Y looks up X in DS, packet gets to R2, which extracts telephone number and dials it. Magic! Y reaches X even if R2 and R3 didn’t know about each other.

**Alternatives (let’s say IP over SMDS)**

- R3 configured with R2’s address (β), and R2 with R3’s. They exchange routing information to let each other know the addresses they can reach.

- Convince the backbone to carry IP addresses as well as its own style addresses.

- Do some kind of broadcast over the backbone, as if the backbone were a LAN.

**IPv6**

- 8 + 8 bytes

- Bottom 8 bytes supposed to be a unique ID based on EUI-64

- IEEE defined “EUI-64” which are like 48 bit addresses, still with a 3 byte OUI, but now with a 5 byte block.

- (I’d have made the OUI 5 bytes. Oh well)

- Top 8 bytes is 3 bits (format indicator, currently only 001 really used), 13 bits for TLA (top level aggregation), with formal IANA assignments, 32 for “next level” and 16 for “site level”

- Maybe people will have the entire site autoconfigure the top level part.

**Fragmentation and Reassembly**

- X.25 has “more” flag. That won’t work here.

- Can you just chop packet in half?

- Rewrite hdr on each fragment, number 1, 2, 3?

- How do you recognize fragments of same pkt?

- Information for reassembly:
  - source, destination addresses
  - packet identifier
  - total length (in CLNP)
  - last fragment (in IP)
  - offset
Reassembly

- Same pkt if source, destination, packet id match
- Copy data into buffer
- How do you know when you’re done?
  - fragment marked “last fragment” arrives?
  - fragment that fills end of buffer (CLNP) arrives?
  - count fragment octets received?

Reassembly Strategies

1. bit mask — fragmentation must be on 8 octet boundary (both CLNP and IP) so bit mask can be 1/64 size of buffer
2. In place bookkeeping
   - ptr to 1st hole
   - ptr next hole
   - length this hole
   - ptr next hole
   - length this hole
   - ptr next hole
   - length this hole

Reassembly Deadlock

- Fragment arrives which doesn’t match any packets being assembled. Buffers are full. What should destination do?
  - discard new fragment?
  - discard oldest buffer?
  - discard longest one?
  - discard one at random?
  - crash?

Avoiding Network Layer Fragmentation

- IPX, AppleTalk, DECnet Ph 4 — no frag
- IPv6 has fields for fragmentation but the claim is that routers shouldn’t do fragmentation (only source network layer should)
- IP, CLNP has it but reassembly is awful. Dest and rtrs would prefer avoiding it if possible
- Strategies for choosing reasonable size:
  - analyze debris of fragmented pkt at dest
  - truncate
  - set “don’t fragment” flag. IP and CLNP give an error report “packet too big” to the source. Do binary search of potential sizes
  - add “but this size would have worked” to error report
Source Route Option

- IP and CLNP have two flavors: loose (partial) and strict (complete). IPv6 allows each hop to be specified as to whether it’s loose or strict.

<table>
<thead>
<tr>
<th>Destination</th>
<th>Source</th>
<th>Other stuff in fixed hdr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>source route option:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ptr into route</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hop 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hop 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hop 3</td>
</tr>
</tbody>
</table>

Source Route Option

- CLNP — only thing that changes is route pointer in option — it would be awkward overwriting DESTINATION because of variable length addresses

- IP
  - overwrites destination field in header with next destination
  - overwrites each hop with the forwarding router’s outgoing address
  - that way route can be used by destination, in reverse order (and indeed source route to S is required if in answer to packet received from S with source route)

Packet Header

Security Issue, Source Routing

- Suppose security assumes who you are based on your address in the IP header
- You are Boris in Russia, (with address X). You want to send a message to address Z, claiming to be President in US (address Y). You can transmit an IP packet with source address Y. But the return traffic won’t go to you...
- Unless you use the source routing option.
- Transmit with source = Y, destination = Z, and source route claiming it’s already done one hop of source route and visited X. When Z returns it, it will have source route X,Y. When you (X) receive packet you of course won’t forward it to Y.
- Also: may foil “source address filtering”
Source Routing

Z (according to original “host requirements” doc), must return traffic to Y via X

Quality of Service

- Priority
  - always drop lower priority packet if higher priority one around?
  - give different guarantees of buffers for each priority class?
  - how do you guarantee field will be set fairly?

- Routing Metric
  - possible metrics: bandwidth (default), delay, error rate, and money
  - CLNP has a bizarre set of 3 bits
  - IP had 3 bits: bandwidth, delay, error.
  - Now “diffserv” WG is haggling over what to do with the 8 bit TOS byte. “congestion indication”, “I react to CI”, and remaining 6 bits for ...?

CLNP Quality of Service

- E/C — error rate vs cost (money) (1 E wins)
- E/D — error rate vs delay
- D/C — delay vs cost

<table>
<thead>
<tr>
<th>E/C</th>
<th>E/D</th>
<th>D/C</th>
<th>metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>money</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>delay</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>money</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>default</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>default</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>delay</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>error metric</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>error metric</td>
</tr>
</tbody>
</table>

Options

- Should they be copied into every fragment'
  - IP has a flag for each option, set by the source, telling routers what to do with that option
  - CLNP has routers copy every option into every fragment
  - IPv6 (theoretically) routers won’t fragment
  - IPX, AppleTalk, DECnet (Phase IV and III) have no options

- IPv6 separates options into 4 classes:
  - end to end (not to be looked at by routers) -- only thing in there is fragmentation info
  - hop by hop — nothing defined
  - authentication
  - source routing
**IP Data Packet**

1. version
2. type of service
3. total length
4. pkt id
5. offset
6. offset (cont’d)
7. time to live
8. protocol
9. header chksum
10. source
11. destination
12. options
13. padding

**CLNP**

1. NLPID
2. hdr length
3. version
4. lifetime
5. sp ms er type
6. segment length
7. header checksum
8. dest add length
9. dest address
10. dest address
11. source address
12. pkt identifier
13. segment offset
14. total length
15. options

**AppleTalk**

1. hop ct
2. pkt length
3. checksum
4. dest net
5. src net
6. dest node
7. source node
8. dest socket
9. src socket
10. protocol type

**IPX**

1. checksum
2. pkt length
3. transport ctl (hop ct)
4. pkt type
5. dest net
6. dest node
7. dest socket
8. src net
9. src node
10. src socket

- Note: checksum isn’t implemented and is set to FFFF hex. Good thing. Why? See next slide.
**IPX on CSMA/CD**

- Ethernet format. Protocol type=8137 hex
- Raw 802.3 — leave out all multiplexing! Start IPX packet where DSAP should be, so checksum covers DSAP and SSAP
- SNAP — DSAP=SSAP=SNAP (aa hex), protocol type=0.0.0.81.37
- 802.2 — DSAP=LSAP=E0 hex

Cope with multiple formats by treating LAN as multiple logical LANs, and routers translate formats.

---

**DECnet Phase IV**

**Short Format**

Flags:
- p=padding, v=version, rts=this is an error message — pkt being returned to sender, rq=request error return if problem, format 2=short, 6=long, c=congestion experienced

![Diagram of DECnet Phase IV Short Format]

**Long Format**

- 2 bytes of Phase IV address go in bottom of address, with a constant (hi-ord) in next 4 bytes
- The resulting 6 byte address is what the Ethernet chip is overwritten with

![Diagram of DECnet Phase IV Long Format]

**IP v6**

![Diagram of IP v6]

- ver(4 bits)/service(8)/flow(20)
- v c flow label
- payld ln nxt hop
- source
- destination
- hop-hop header
- rtg header
- authentication
- end-end options


**IP v6 Extra Headers**

- Each start with 1 octet indicating type of next header (TCP, or one of the other options), followed by 1 octet giving length (in 8 octet chunks) of this header. Inside is TLV (type, length, value) encoded options. Not many options are defined. Things are still changing a lot.

<table>
<thead>
<tr>
<th>next</th>
<th>length</th>
<th>type</th>
<th>length</th>
<th>value</th>
<th>type</th>
<th>length</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>next</td>
<td>length</td>
<td>type</td>
<td>length</td>
<td>value</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Names/Addresses**

- Remembering and typing layer 3 addresses unpleasant
- Better to specify destination by “name”
- Why hierarchical names: east.sun.com
  - Hard to make sure all names unique, and meaningful, and reasonably short
  - Want to delegate administration (not have one organization have to deal with everyone to make sure the name is unique)
  - Phone book would get too big
- Like a hierarchical file system
- Don’t need to keep all information in one place. Instead, each directory can be stored on a different server

**Headers inside Headers**

As transmitted by S:

- layer 2 hdr
  - D=β
  - S=α

- layer 3 hdr
  - D=23.19
  - S=23.35

As transmitted by R1:

- layer 2 hdr
  - D=χ
  - S=δ

- layer 3 hdr
  - D=77.19
  - S=23.24

**Replication**

- Don’t want any of the information to be subject to a single point of failure
- So each directory can be stored on multiple servers
  - “Siblings” with respect to a directory all store that directory. All siblings store A.B.C.D
  - “Parents” store the parent of a directory. So parents of A.B.C.D would store B.C.D
  - “Child” of directory of A.B.C.D would be something like X.A.B.C.D
- Suppose each server that stores directory A.B.C.D keeps:
  - a list of addresses of siblings
  - a list of addresses of parents
  - for each child directory, a list of addresses
**Replication Strategies**

- One “master”, at which updates can be made. Secondaries either periodically check with master for updates, or master reliably updates the secondaries when there are changes.
- Suppose no master (all equal)
  - everyone checks with everyone
  - configured ring (Digital’s scheme)
  - configured tree (like telephone tree at daycare center to alert parents to snow days)
  - gossip (an idea that keeps getting reinvented, but has been around for many years)

**Gossip**

- Update described by (replicaID, time, info)
- Each replica keeps the following:
  - replica1, time1, time2
  - replica2, time1, time2
  - …
  - replica_n, time1, time2
- When replica_i contacts replica_j:
  - exchange data structures
  - for each replica#x, if i’s time1 for x < j’s time1 for x, then j sends i all x’s updates between i’s time1 for x and j’s time1 for x.
  - for each time2, take the max between i and j
  - when done, throw away all updates marked less than the smallest time2

**Navigating**

- One idea:
  - each directory knows how to find a replica of its parent, and how to find a replica for each child
  - Find any server that stores a directory, have it tell you how to get to the parent, and how to get from there to the next level...up to the least common ancestor
  - Then go downwards to the place you want
- Another idea: have everyone be configured with the address of a replica of the Root
- Who does the work? You say “I want to get to A.B.C.D” to a server that stores X.Y.Z.E.D” It could tell you the next step, or it could do all the work and tell you the answer