Local Address Resolution

Peter Lo

What's the Issue?

- Principally, how do machines on a network obtain MAC addresses of other machines?
- Later, this unit considers how data is sent from one network to another
  - This requires a different level of addressing
- Now we consider how that higher level of addressing is converted to a machine address on the network
  - We need to resolve the addressing

The issue this week is to describe how nodes find out the MAC addresses of other nodes. In the NetSim lab this was presented as devising a scheme whereby we could use NetSim computer names for communication instead of MAC addresses.

In the Internet nodes are identified by multi-level addresses called **IP Addresses** so in an Internet context the IP address takes the place of the NetSim name and the issue is "how do we translate a local node's IP address into its MAC address" (In order that we can make a local delivery)

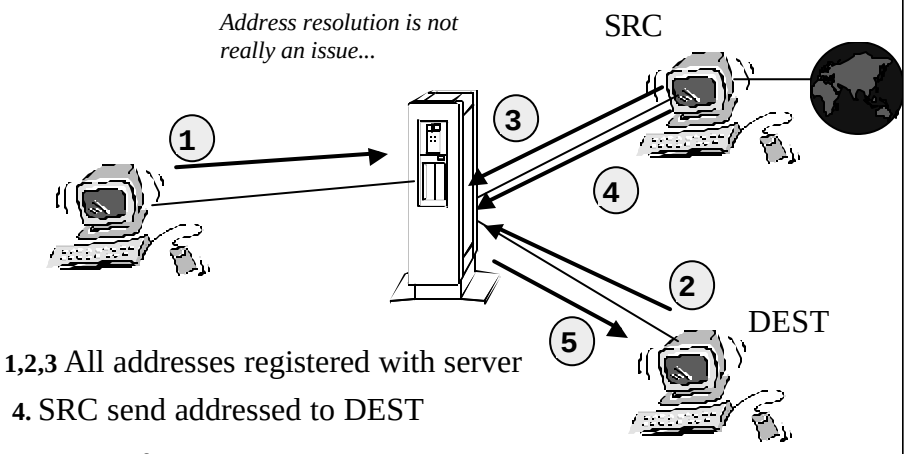
A more general name for this type of action is **resolution** of the IP address.

Some Broad Approaches

- Server controls communication
- Server aids address resolution
- MAC addresses are globally known
- Peer-to-peer resolution

According to the way that the local network functions there may be different ways of going about address resolution.

Communication via a Server



1,2,3 All addresses registered with server

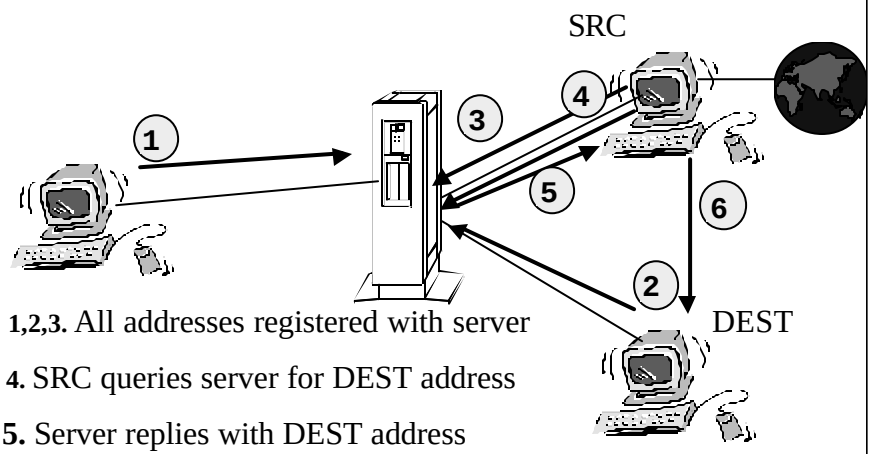
4. SRC send addressed to DEST

5. Server forwards to DEST

If the communication is actually **via** a server then there is no real issue of address resolution.

So long as all nodes register their addresses with the server the server itself will do all the required forwarding of packets.

Server Assisted Example



1,2,3. All addresses registered with server

4. SRC queries server for DEST address

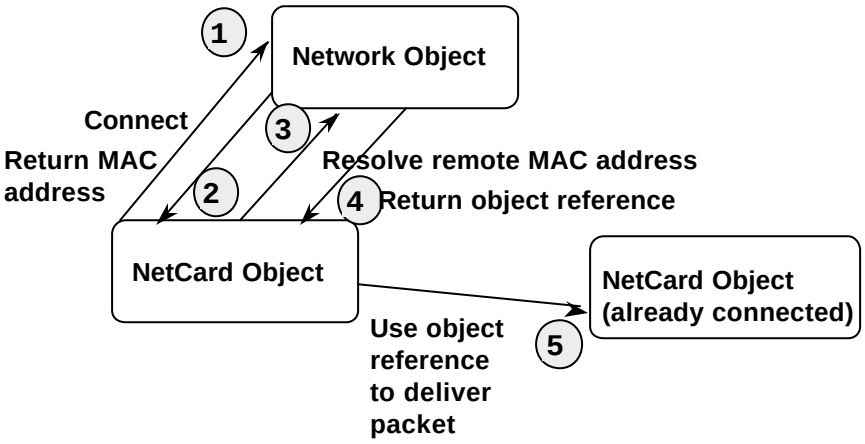
5. Server replies with DEST address

6. SRC forwards to DEST directly

In this scenario the server does not actually do the forwarding of packets. The only responsibility of the server is to help in address resolution in order that nodes can forward directly to each other.

This is precisely the architecture of NetSim as the next slide shows.

Overall Workflow



NetSim does the same!

MAC Addresses Globally Known

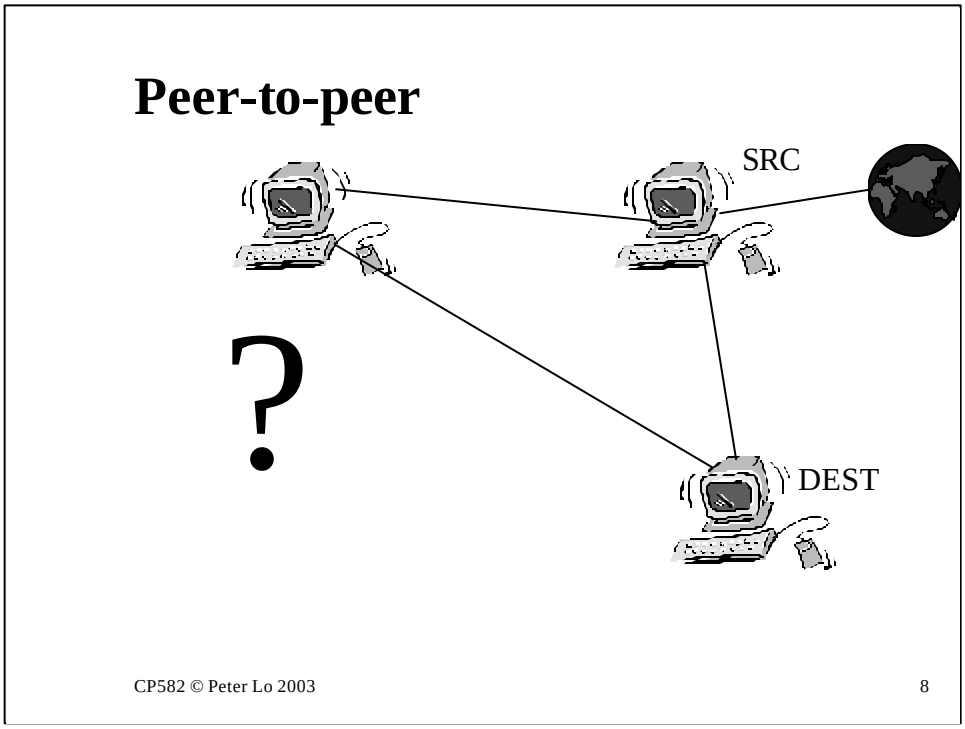
- Include the MAC address in the global multi-level address
  - IPX (Novell) does this
- Simple, but…
  - Global address becomes as "unpronounceable" as MAC address
  - "steals" addressing bits from the network address
    - 48 bits of MAC address goes everywhere
  - doesn't follow a nice hierarchical model
    - If the NIC changes the MAC address changes
    - What if I store another nodes address?

This solution, used by Novell in IPX, eliminated the problem of local address resolution by using the MAC address as part of the global, multi-level address. When a packet arrives from far away it already contains the MAC address of the destination so local delivery can proceed automatically.

Remote nodes are identified by a "global" address which consists of a number identifying their network combined with their MAC address. Nodes do not have names at all - they end up being identified in the internet work by numbers that are even longer and harder to type than MAC addresses.

The adoption of this solution reflects the fact that Novell networks do not use peer-to-peer communication. The basis for a Novell network is communication between client nodes and servers. Servers have short text names and so can easily be referred to - if a client needs to contact another client it does so via the server that knows the global addresses of all its clients.

So IPX eliminates the issue of local address resolution but to make this work the servers become central to **all** communication.



There are several reasons **not** to use a server for local address resolution:

• Single point of failure - how will any nodes send anything if the server is down?

• Centralised load - if there is a lot of resolution happening the server becomes a bottleneck.

• Administration - somebody needs to set up and manage the server.

Points such as these lead to the conclusion that it would be better to devise a peer-to-peer scheme of local address resolution and the rest of this lecture focuses on this style of solution.

Peer-to-peer

- Every machine holds a table
  - This consists of address pairs enabling a simple look-up to convert between them
  - These tables can be static…
    - an administrator maintains them
    - handles change only as well as the administrator
  - … or dynamic
    - the machines provide their own mechanism for discovery
    - this is the ARP approach
      - Address Resolution Protocol

For a peer-to-peer solution to work nodes must have a way to look up the addresses of their peers.

One approach (not recommended) it to store a **static** translation table at each node. This table is termed static because it needs to be updated manually to reflect changes. Since an administrator must do this the system fails if the administrator does.

It is clearly much more sensible to provide the nodes with a mechanism which they can use independently to build and rebuild their own translation tables.

This is the solution used in TCPIP networks and is known as ARP - Address Resolution Protocol.

Focus on ARP

- The Address Resolution Protocol
  - SRC looks for DEST entry in its local table
  - if found, fine. ARP is finished. Thanks for calling.
  - (Else) broadcast an ARP query on the network
    - query includes DEST higher-level address
  - Wait…
    - all machines read the query
    - if they see that the address matches theirs, send a response which includes their machine address
  - Add a new entry to the local table. Continue...

This slide spells out the "rules" of the ARP protocol. One might say the "protocol" of the protocol - here we are using "protocol" the first time in its plain English sense of what is the correct thing to do and when.

The basic rules of ARP are:

• Be prepared to maintain a local address translation table

• Always look first in this table

• If this does not help then broadcast, saying the name that you want resolved

• If the named node sees the broadcast it responds - including its MAC address.

• Use the MAC address to send a packet

• Store this new information in the table.

An aside on scalability

- To build scalable networks…
  - Minimise network traffic
    - Design network protocols with this in mind
  - Minimise storage needed at nodes
    - Avoid storing un-necessary information

If a network is to function just as well when it gets big as it did when it was small (be **scalable**) then there are at least two things to watch out for:

• Network traffic - if you design protocol that works well for 10 stations stop and think how it would be for 1000

• Local storage - similarly look at the effect of growth of the network on local storage needs that are needed to make the network function.

Spreading the news faster

- Can you see how this description can be added to in order to provide better dissemination of address pairs?
- Important to consider scalability
  - Let's not store it if we don't need it. So who needs it?
    - The DEST machine
    - any with current table entries for SRC, where they are different
- The enhancement is used by ARP.

The straightforward rules proposed would mean that each node would need to broadcast the first time it needed to send to another particular node. The consequence might be a large amount of broadcast traffic which could slow the network down.

The logical thing to do is for nodes, even if they do not need to respond to broadcast requests (it was not for them) to make use of the information in the request ("Oh, I see that the machine John is using the MAC address 024F5C33D29A - I'll note that down for later use")

This is very efficient use of the network but might be bad use of local storage. What if we never ever need to send a packets to John or Sue or Mary or Bill or Jane …….

The decision that ARP makes is:

• Store this info if we are actually who John is looking for. This will save us using ARP when we reply to the message John is preparing to send.

• Store this info if we already have a MAC address stored for John but it was different. In this way we keep bang up to date on the MAC addresses in the computers that we are involved in communicating with.

The ARP cache

- ARP uses an address mapping table
- This is stored in RAM on the local machine
- It is called the ARP cache
- Cache updates periodically - the dynamic approach
- Updating includes removal of old entries

The name given to this locally stored table is the **ARP Cache**

Each node manages its own table and an important part of this management is to make sure that entries do not stay in the table for too long.

There are two reasons for this:

• To keep the table reasonably small - hence it will be quick to look through and use less RAM

• To make sure that there is no stale information in the table such as mappings for nodes that have gone down or changed their NICs.

ARP command

- **On Unix or Windows**
- **arp -a**
  - Provides a look at the ARP cache
  - Reveals mappings between the higher-level address…
    - **"Internet address"**
  - … and machine address
    - **"physical address"**

```
C:\Netsim\newGUI>arp -a

Interface: 141.132.69.40
  Internet Address      Physical Address      Type
  141.132.64.2          08-00-20-73-c1-ac     dynamic
  141.132.64.13         00-03-e3-c1-02-f7     dynamic
```


Both Unix and Win95/98/NT provide a command for managing ARP

The most interesting option is **arp -a** which displays the contents of the local ARP cache. You can use this command repeatedly to figure out how old an entry must be before it is removed.

How do machines know???

- ARP requires machines to know both addresses.
  - How do they know this?
    - MAC address is in the NIC
    - Internet address is in a config file
  - What if a workstation is diskless??
    - Reads its MAC address from hardware
    - Sends a RARP (Reverse ARP) request to discover its Internet address
      - Who knows an Internet address that matches this MAC address??
      - Have a look in */etc/ethers* on a Unix host

It may seem obvious but it worth saying that in general nodes know both their own MAC address and Internet address. This means that they know when to respond to an ARP request and that they know the information to put in the reply.

The only exception to this is a machine which is diskless - with no local hard disk there is nowhere to store the config file which holds the Internet address.

The solution is an ARP-like protocol that does the opposite of ARP. Reverse Address Resolution Protocol (RARP) broadcasts the nodes MAC address. A dedicated RARP server sees the broadcast and looks up in a table (in Unix /etc/ethers) what Internet address is supposed to be given to this particular node.

Binary & Hexadecimal

- To understand the tools of the trade
  - NetXRay, Lanalyzer, etc.
- Because computers use a different number base to us
  - binary, as opposed to our decimal
- Because some problems are inherently easier to solve using different base number systems
- It's often actually easier to use
  - I guess we'll have to convince you of this!!

Binary & Hexadecimal Numbers

This slide sets out some reasons for getting into a whole new way of counting. The reasons all arise from the fact that computers are built upon binary counting.

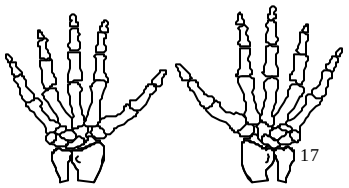
The earliest computers were mechanical and tried to count the way that humans do. Every cog and wheel within the computer had 10 positions and this introduced complexity into the design.

When electronic computers were being developed it was seen that electric circuits fundamentally have two "positions" or states - on and off - so it was decided to base the internal functioning of the computer on counting that had only two digits - counting to the base 2. There would be an overhead in communicating input and output with humans but the bulk of work was internal so the saving in complexity would be significant.

As a consequence, those who are concerning themselves with what happens inside computers need to learn to count the way computers do.

We are decimal critters...

- There are 10 “digits” in the system (0-9)
  - Because we have 10 fingers
- The number 1234 means:
  - (1 x 1000) + (2 x 100) + (3 x 10) + (4 x 1)
  - Each factor represents a power of 10
  - We need “0” to represent an empty column
    - 1234 - 234 = (1x1000) + ….nothing
    - How do we represent all that nothing?



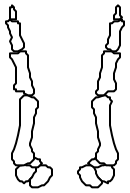
This is a history lesson. It was not always obvious how to use a **number base** to help in the representation of large numbers.

The Romans simply (!) extended the concept of counting with larger and larger “fingers” which made their numbers hard to read and also not scalable - what is “one billion, 73 million and 42” in Roman numerals?

The idea of representing numbers in **weighted columns** was Arabic and the concept of **zero**, which represented an empty column, was the revolutionary development that made this practicable.

But what if...

- There are 2 “digits” in the system (0-1)
  - Because we have 2 fingers
- The number 1010 means:
  - (1 x 8) + (0 x 4) + (1 x 2) + (0 x 1)
  - Each factor represents a power of 2

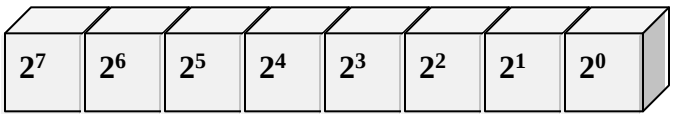


If we had been less complex creatures things might have been different…..

The idea of counting using 10 digits evolved from our physiology so if we had been a race of two-fingered critters we would have evolved counting based on 2 digits - counting to the base 2.

If we have two digits the first column can only represent “zero” or “one” and if a number gets bigger than this, as they are prone to, we must begin to **carry.** The first time we carry we are counting “twos” - when we carry from that column we are counting “fours” and so on. We use powers of 2 for the column headings.

Understanding binary



Can you see why the maximum binary value held by 5 bits is 2^5-1?
Use intuition not serious maths...

Binary counting is based on the columns having weights which are powers of 2 rather than powers of 10.

The question on the slide asks you to think about what "11111" (5 1's) is without adding it all up.

The key is to think about the number that is 1 bigger.

11111

\+ 1

0 carry 1

0 carry 1

0 carry 1

0 carry 1

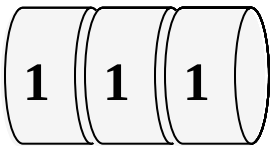
0 carry 1

100000

- what is this number? It's easy……32 (2^5) so the original number was 32 -1 = 31.

Intuitive Binary Counting

- Imagine a binary odometer for a three-bit system



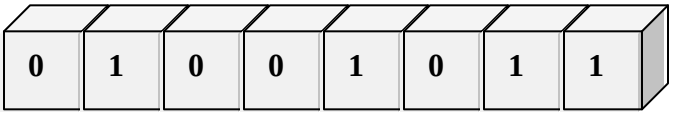
000
001
010
011
100
101
110
111

This animation shows binary counting with an odometer analogy.

All the possible combinations for a three **bit** number are shown.

Binary to Decimal

- The diagram represents a base 2 number
  - Is it even or odd?
  - What does it represent in base 10?



Even or odd?

The easiest question to answer about a binary number……..

Value in base 10?

Have to start adding up:

$1 \times 2^0 = 1$

$1 \times 2^1 = 2$

$0 \times 2^2 = 0$

$1 \times 2^3 = 8$ etc……..

Decimal to binary

- Harder
- Have to start guessing powers of 2
- For example: 97
  - Look for the largest power of 2 that fits in once
    - try 32 - goes 3 times - no good
    - try 64 - bingo! - 64 is 2^6 so we have 100000 and 97 -64 = 33 still to convert.
    - try 32 - bingo - 32 is 2^5 so we have 110000 and 33 - 32 = 1 still to convert
    - So 110001 is the answer
  - Now try 41, 77, 203 and 1111

There is a lot of fiddling around in this technique:

• First you have to guess powers of two which may not be easy is it is a large number and you are not good with your 2 time table.

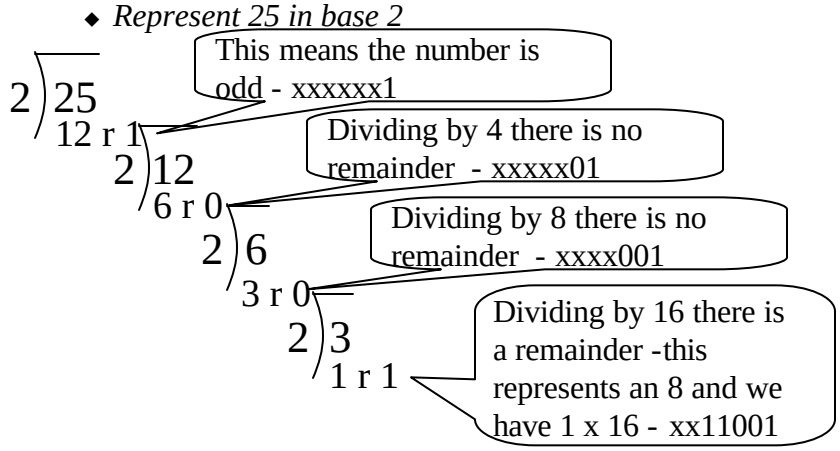
• Then you have to subtract before going through it all again.

The weakness of this method is that we are trying to handle the largest powers of 2 first. What about starting with the smallest powers - after all it is **very easy** to see if a number is odd or even.

This gives rise to the trick shown on the next slide.

Decimal to binary - an "odd" technique

- Easy, if you can divide by 2!!
  - *Represent 25 in base 2*



This is a "divide and conquer" (!) technique.

We start by dividing the number by 2. If there is a remainder we have a 1 in the one's column otherwise 0.

When we divide by two again we are actually dividing the original number by 4. If there is a remainder this time it represents a 2. We are testing the "oddness" of half our original number. Now we can fill out the "twos" column.

We repeat this until we have a 1 as the answer when we divide. At this point we have figured out the largest power of two that will fit into the original number which is what we started with in the previous technique. We have also completed the conversion to binary.

Let’s see that again...

- Try a harder one…
  - What is the base 2 representation of 78_{10}?

Try using the “odd” technique from the previous slide.

Look ma, no maths!

- What about these conversions from base 10 to base 2?
  - 63
  - 62
  - 129

Trying doing these without doing a complete step by step conversion. They are "easy" because they are close to a power of 2.

What is hexadecimal?

- Hexadecimal is a base 16 number system
  - Whatever you know about base 2 and base 10 you can apply to base 16
    - *with appropriate number substitutions*
  - The 16 digits are formed from 0-9A-F
    - *in increasing order of significance*
  - Because $16 = 2^4$, 4 bits can represent any hexadecimal digit.
    - *We'll see that this enables hex to provide a neat shorthand for binary*

All the 0's and 1's are hard for us humans so we have invented a way to gather the binary digits into groups of 4.

Four binary digits can represent numbers from 0 to 15 - consequently we need 16 different digits that we can write down -

0 - 9 takes care of the first 10

For the remaining 6 we use letters - A,B,C,D,E,F

Binary to hex and back

- If you are happy with the natural order of binary, you basically already know hex.
  - Just apply the hex digits in order

0000	0	0110	6	
0001	1	0111	7	
0010	2	1000	8	
0011	3	1001	9	
0100	4	1010	A	(10)
0101	5	1011	B	(11)
		1100	C	(12)
		1101	D	(13)
		1110	E	(14)
		1111	F	(15)

This a summary of the hexa-decimal notation scheme - this needs to become second nature to you!

Hex - binary conversions

- Convert from binary to hex:
  - 1101 0010 1111 1110 0000 0010 1011 1100
  - 1011001110110100
  - 1100011011100
  - 11101000100110110
- Convert from hex to binary:
  - FA207E
  - DEADBEEF
  - CAFECAFE

These should be much easier that anything involving decimal because of the fact that **one hex-digit = four binary digits.**

The only problem that can arise is where there is binary number that does not have a multiple of four digits. In this case you must remember to begin your conversion at the right hand end.

Some big numbers

- 1 KB = 1024 (≅ 1000)
  - = 2^{10}
  - = 10000000000 (10 zeros)
  - = 0x400 (notice the terminology for hex)
- 1 MB = 1KB x 1KB (≅ 1,000,000)
  - = 2^{20}
  - =100000000000000000000 (20 zeros)
  - = 0x100000

These are some important conversions to commit to memory.

Anyone for octal?

- No big deal… In fact, the same deal as hex.
- Binary to octal:
  - 011 111 010 101 110
- Octal to binary:
  - 70623

Very occasionally in computing you may come across **octal** representation. This is counting to base 8 which involves gathering the binary digits in groups of three.

An octal number can **never** have the digits 8 & 9 in it - these are represented as 10 and 11 when you count to base 8.

Those reasons again… # 1

- "Its often actually easier to use"
  - Which would you have more chance of remembering (or successfully repeating to someone else):
    - 1001 0110 1101 0100 0011 0100 0101 1011
    - 96d4345b

Hex is a clear winner over binary in terms of ease of use but you might say why not stick to decimal?

The reason we like to talk about the insides of computers, or packets on the network, in hex is that the underlying mechanism is based on binary arithmetic so the "boundaries" fall naturally into place. This is made clearer in the next slide.

Those reasons again… # 2

- "Some problems are inherently easier to answer in a different number base"
- eg Message on computer screen - which would be most useful?:
  - "Bad memory at address 1864ADB"
  - "Bad memory at address 25578203"

“Bad memory” example

Remember: 1MB = 0x100000 (5 zeros)
so 16 MB = 0x1000000 (6 zeros)

Which SIMM contains the bad address?
(0x1864ADB or 25578203)

SIMM #1 - 16MB: 0 - 0xFFFFFF (6 F’s)

SIMM #2 - 16MB: 0x1000000 - 0x1FFFFFF (6 F’s)

This shows why hex notation, which relates more closely to the way the computer is built, helps solve a real-world problem.

Those reasons again… # 3

- "Because computers use a different number base"
  - From Netware 4.11 Administration:
    - "A single Netware 4 print server can service up to 256 printers…"
  - What does a statement like this reveal to someone who understands the binary nature of computers?