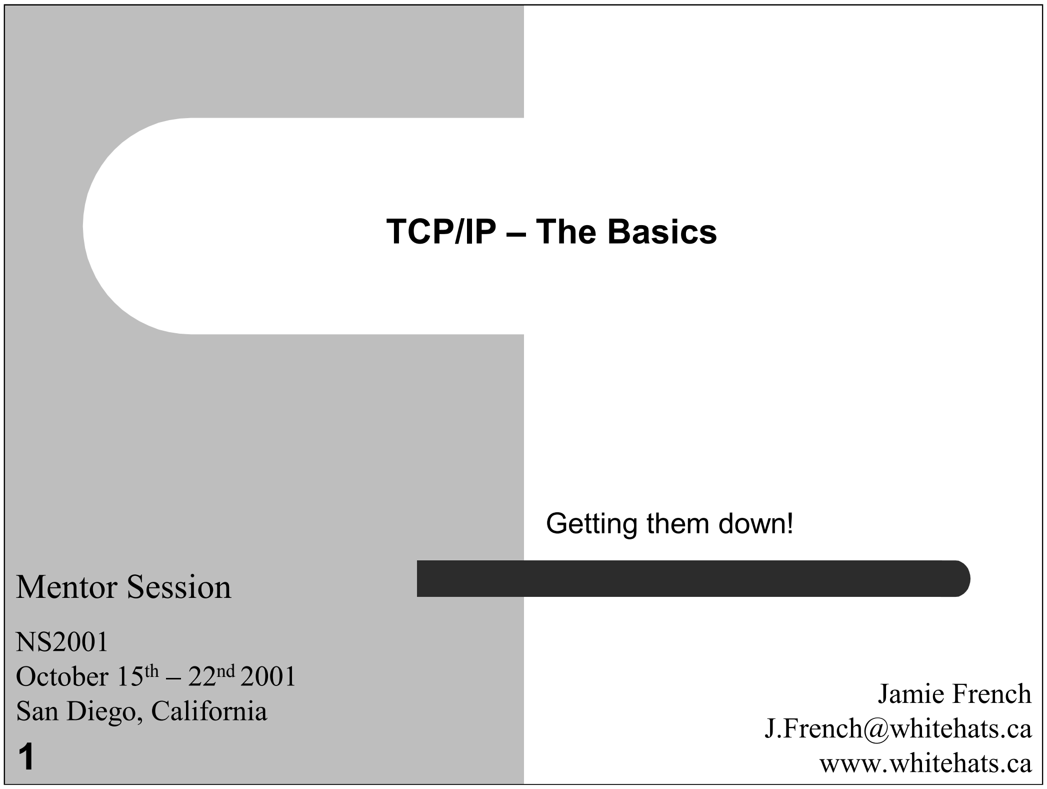


TCP/IP – The Basics

Getting them down!

Mentor Session

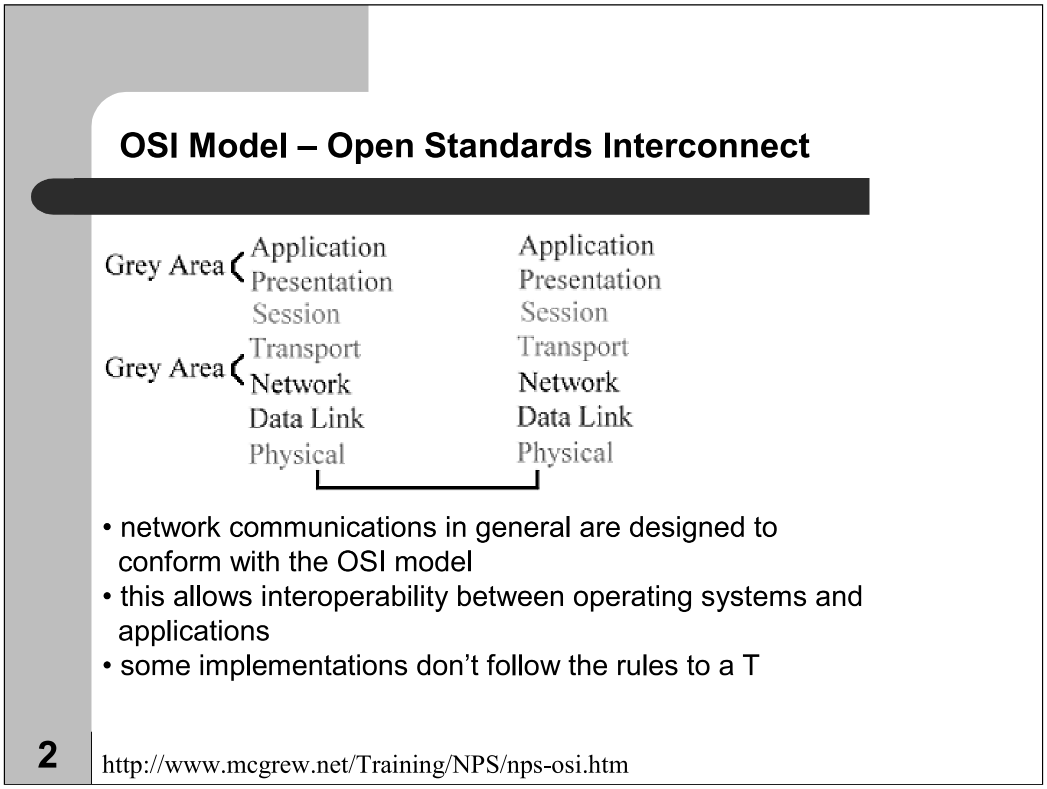
NS2001
October 15th – 22nd 2001
San Diego, California

1

Jamie French
J.French@whitehats.ca
www.whitehats.ca

Having taught networking issues and IDS techniques over the past two years to co-workers, I have often found the source of their troubles is that they don’t understand fundamentally how network communications work.

Lets begin with the fundamentals.



OSI Model – Open Standards Interconnect

Grey Area { Application | Application
Presentation | Presentation
Session | Session
Grey Area { Transport | Transport
Network | Network
Data Link | Data Link
Physical | Physical

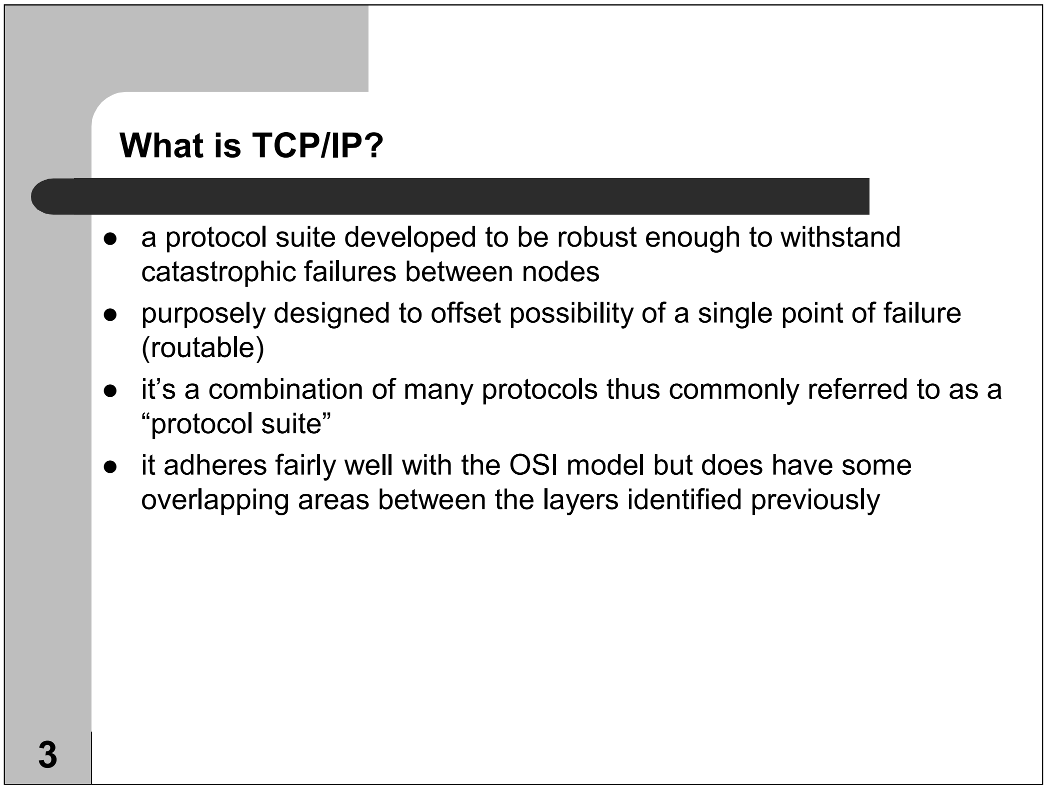
- network communications in general are designed to conform with the OSI model
- this allows interoperability between operating systems and applications
- some implementations don’t follow the rules to a T

2

http://www.mcgrew.net/Training/NPS/nps-osi.htm

1. Take input (any form – i.e. file, keyboard) and pass it to the application to process.
2. The application formats the data into a standardized presentation, such as Hypertext Mark Up Language (HTML).
3. This is then passed to the Operating System (OS) with a requests to open up a socket. If granted, this serves as the session (link) between the application and the networking stack and its supported protocols. The network stack kicks in on the transport layer. Don’t confuse this as thinking we skipped to the network layer.
4. The transport layer puts this data into a package and prepares it for shipping. Basically it places some necessary data in-front of and behind what was passed to it originally by the presentation layer. This added data will become known as the layer 4 header.
5. This data is then taken and packaged again by the network layer. This added data will become known as the layer 3 header.
6. Next the “package” is passed to the data link layer. At this point your data has made it to your network interface card. Another bunch of data is added which will become known as the LLC or logical link control header. In the case of Ethernet networks, people refer to it as the Ethernet header. This is then passed to the hardware and modulated or transformed into a signal that is transmitted. The method of modulation will coincide with the physical link layer (i.e. fiber optic, radio frequency). For our purposes we now have a packet and it is comprised of a stream of 1’s and/or 0’s.
7. This is then placed onto the physical layer and sent on its way. All the headers added are required for other hosts OSI models to understand what to do with the packet when they receive it and to be able to figure out whether it was for them or not. The implementation of the OSI model on an OS is often referred to as the Network Stack.

The implementation of this model by TCP/IP often melds the application and presentation layers together as one function. Another important point is that data is unidirectional within the layers. For instance data will not flow from layer 3 to layer 2 and then back to layer 3. The data will come in or go out and continue in its current direction until it is processed or discarded.



What is TCP/IP?

- a protocol suite developed to be robust enough to withstand catastrophic failures between nodes
- purposely designed to offset possibility of a single point of failure (routable)
- it's a combination of many protocols thus commonly referred to as a "protocol suite"
- it adheres fairly well with the OSI model but does have some overlapping areas between the layers identified previously

3

During the cold war the threat to command and control communications (CCC) was considered great. If a strategic strike were performed by the Soviet Union on the communications network, the fog of war could drastically hinder US reactions and defensive coordination.

1957 ARPA - Advanced Research Projects Agency was formed.

1972 ARPA was renamed to DARPA – Defence Advanced Research Projects Agency.

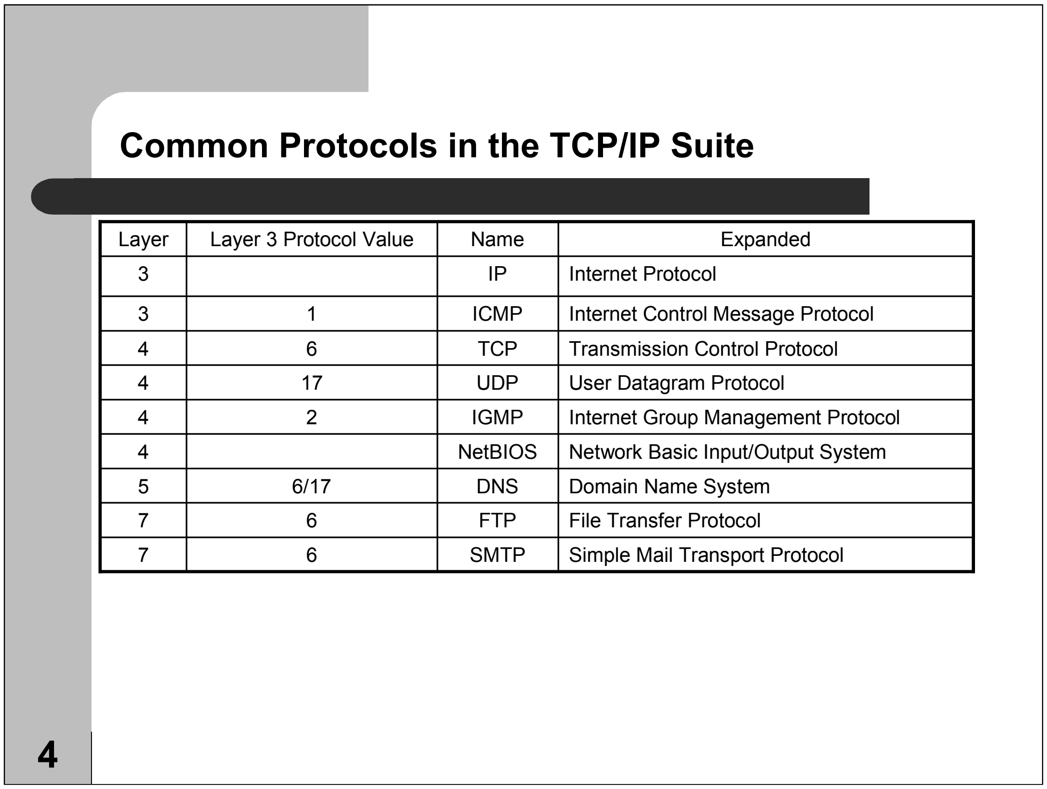
1973 Vinton Cerf from Stanford and Bob Kahn from DARPA begin work on what will become TCP/IP.

1976 Ethernet developed by Dr. Robert M. Metcalfe.

Definition of protocol from Merriam-Webster's Dictionary – both are applicable in context:

a: a code prescribing strict adherence to correct etiquette and precedence (as in diplomatic exchange and in the military services)

b : a set of conventions governing the treatment and especially the formatting of data in an electronic communications system



Common Protocols in the TCP/IP Suite

Layer	Layer 3 Protocol Value	Name	Expanded
3		IP	Internet Protocol
3	1	ICMP	Internet Control Message Protocol
4	6	TCP	Transmission Control Protocol
4	17	UDP	User Datagram Protocol
4	2	IGMP	Internet Group Management Protocol
4		NetBIOS	Network Basic Input/Output System
5	6/17	DNS	Domain Name System
7	6	FTP	File Transfer Protocol
7	6	SMTP	Simple Mail Transport Protocol

4

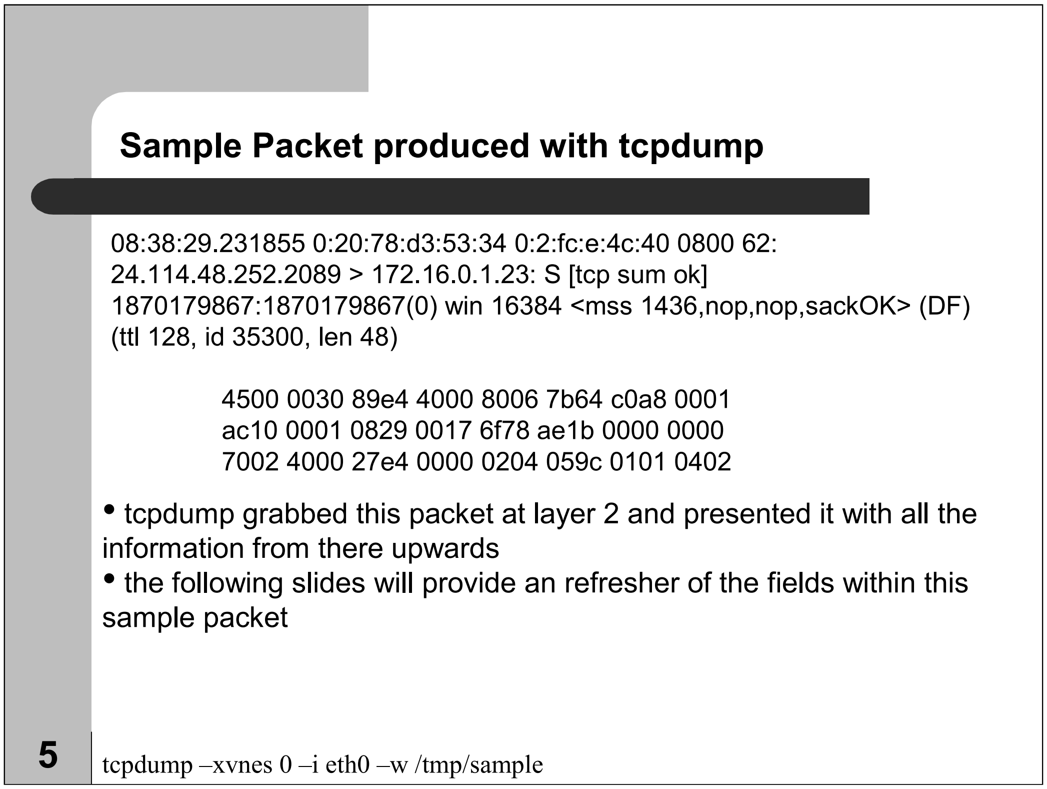
Lets look at this a little more closely. See if you can follow the logic of the layer 3 protocol values. These values are in the IP header and tell the receiving end how to process the packaged data.

From the list above you will note that IP doesn't have a layer 3 value. That is because it is the shipping/receiving department protocol. Anything passed down the stack from IP gets its own header, therefore it does not need a number.

ICMP is an exception to the rules. It is listed as a layer 3 protocol but it has a value assigned to it. This is because the protocols main purpose is to help solve network connectivity problems. It is not directly interfaced with above the network layer but creates a grey line as it does go above this layer often to determine information or deliver information to an application.

Layer 4 protocols have distinct values when they are addressed by IP. Note that NetBIOS does not have a specific value. It is a separate protocol within the TCP/IP suite and should be distinguished that way. NetBIOS over TCP/IP is another story. It can be either protocol as decided by the application and/or presentation layer.

All the protocols above layer 4 are decided upon by the application. Depending on application specific parameters that were programmed, it will likely be sent down the stack as either UDP or TCP depending on which protocol the programmer deemed to be required for the specific application.



Sample Packet produced with tcpdump

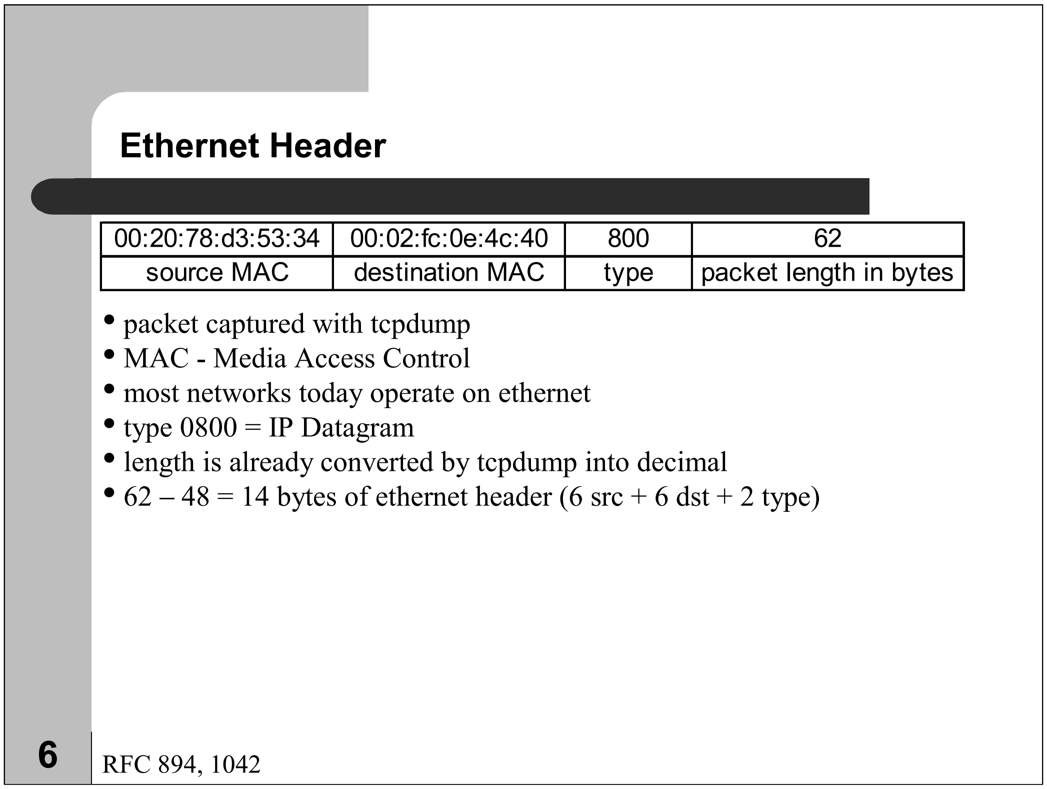
```
08:38:29.231855 0:20:78:d3:53:34 0:2:fc:e:4c:40 0800 62:
24.114.48.252.2089 > 172.16.0.1.23: S [tcp sum ok]
1870179867:1870179867(0) win 16384 <mss 1436,nop,nop,sackOK> (DF)
(ttl 128, id 35300, len 48)

          4500 0030 89e4 4000 8006 7b64 c0a8 0001
          ac10 0001 0829 0017 6f78 ae1b 0000 0000
          7002 4000 27e4 0000 0204 059c 0101 0402
```

- tcpdump grabbed this packet at layer 2 and presented it with all the information from there upwards
- the following slides will provide an refresher of the fields within this sample packet

5

```
tcpdump –xvnes 0 –i eth0 –w /tmp/sample
```

This space intentionally left blank.



Ethernet Header

00:20:78:d3:53:34	00:02:fc:0e:4c:40	800	62
source MAC	destination MAC	type	packet length in bytes

- packet captured with tcpdump
- MAC - Media Access Control
- most networks today operate on ethernet
- type 0800 = IP Datagram
- length is already converted by tcpdump into decimal
- 62 – 48 = 14 bytes of ethernet header (6 src + 6 dst + 2 type)

6

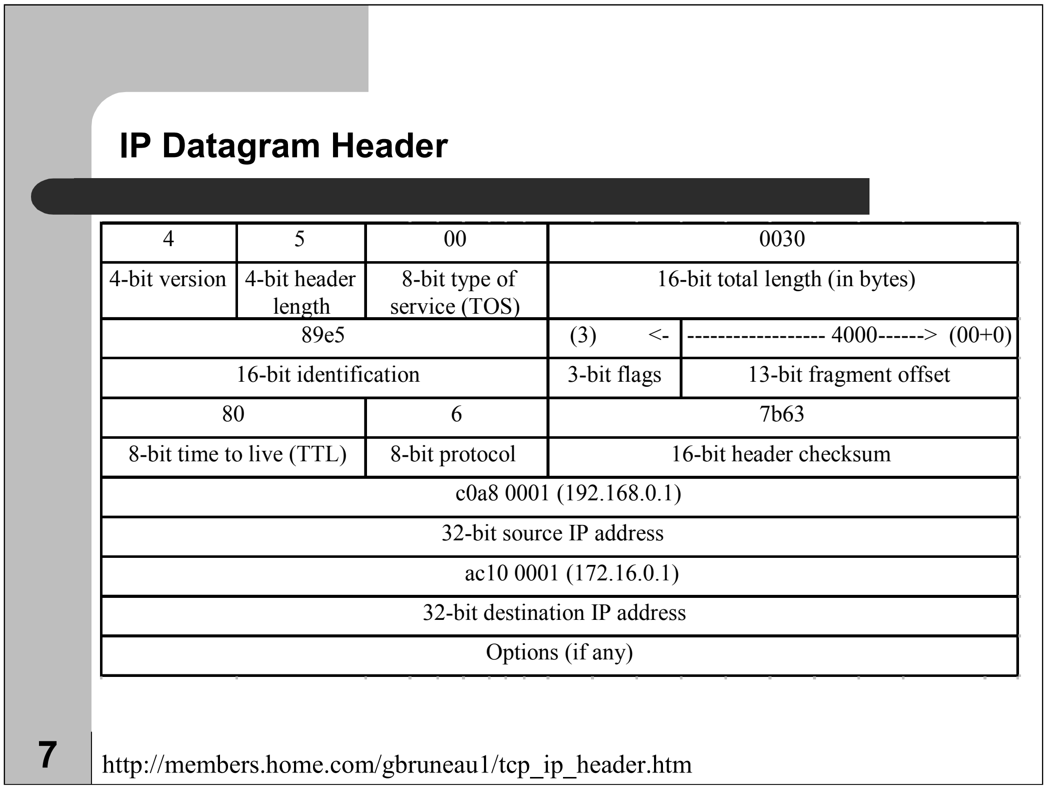
RFC 894, 1042

The whole packet has been included in the sample to familiarize you with the fields and where they sit in relation to each other.

Example Packet:

```
08:38:29.231855 0:20:78:d3:53:34 0:2:fc:e:4c:40 0800 62: 192.168.0.1.2089 > 172.16.0.1.23: S [tcp sum
ok]
1870179867:1870179867(0) win 16384 <mss 1436,nop,nop,sackOK> (DF) (ttl 128, id 35300, len 48)
   4500 0030 89e4 4000 8006 7b64 c0a8 0001
    ac10 0001 0829 0017 6f78 ae1b 0000 0000
    7002 4000 27e4 0000 0204 059c 0101 0402
```

Notice the bolded len 48. This is a value calculated at layer 3 and presented by tcpdump.



IP Datagram Header

4	5	00	0030	
4-bit version	4-bit header length	8-bit type of service (TOS)	16-bit total length (in bytes)	
89e5			(3) <-	------------------ 4000------> (00+0)
16-bit identification			3-bit flags	13-bit fragment offset
80		6	7b63	
8-bit time to live (TTL)		8-bit protocol	16-bit header checksum	
c0a8 0001 (192.168.0.1)				
32-bit source IP address				
ac10 0001 (172.16.0.1)				
32-bit destination IP address				
Options (if any)				

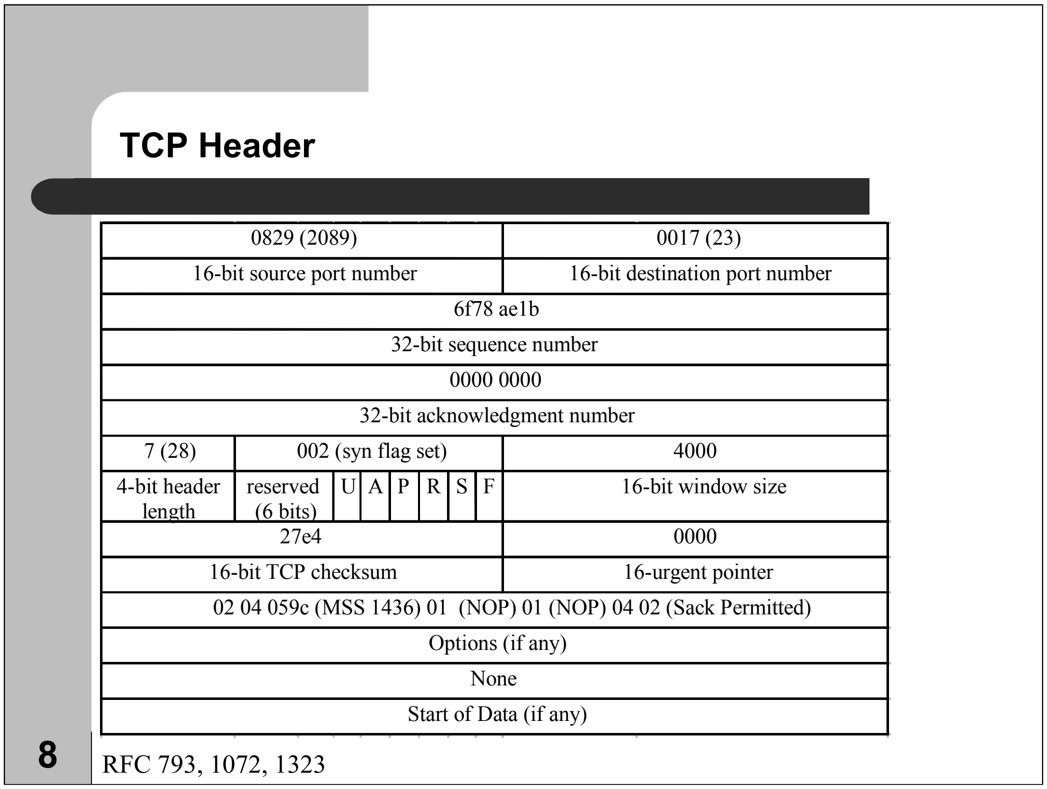
7

http://members.home.com/gbruneau1/tcp_ip_header.htm

Example Packet:

```
08:38:29.231855 0:20:78:d3:53:34 0:2:fc:e:4c:40 0800 62: 192.168.0.1.2089 > 172.16.0.1.23: S [tcp sum
ok]
1870179867:1870179867(0) win 16384 <mss 1436,nop,nop,sackOK> (DF) (ttl 128, id 35300, len 48)
   4500 0030 89e4 4000 8006 7b64 c0a8 0001
   ac10 0001 0829 0017 6f78 ae1b 0000 0000
   7002 4000 27e4 0000 0204 059c 0101 0402
```

Byte 6 and 7 into the IP header present a challenge. The method of arriving at the real value of these fields is by using base 2 math. The hexadecimal value of 4000 must first be converted into binary to do this. We will look more closely at base 2 math, conversions, and binary in the coming slides.



TCP Header

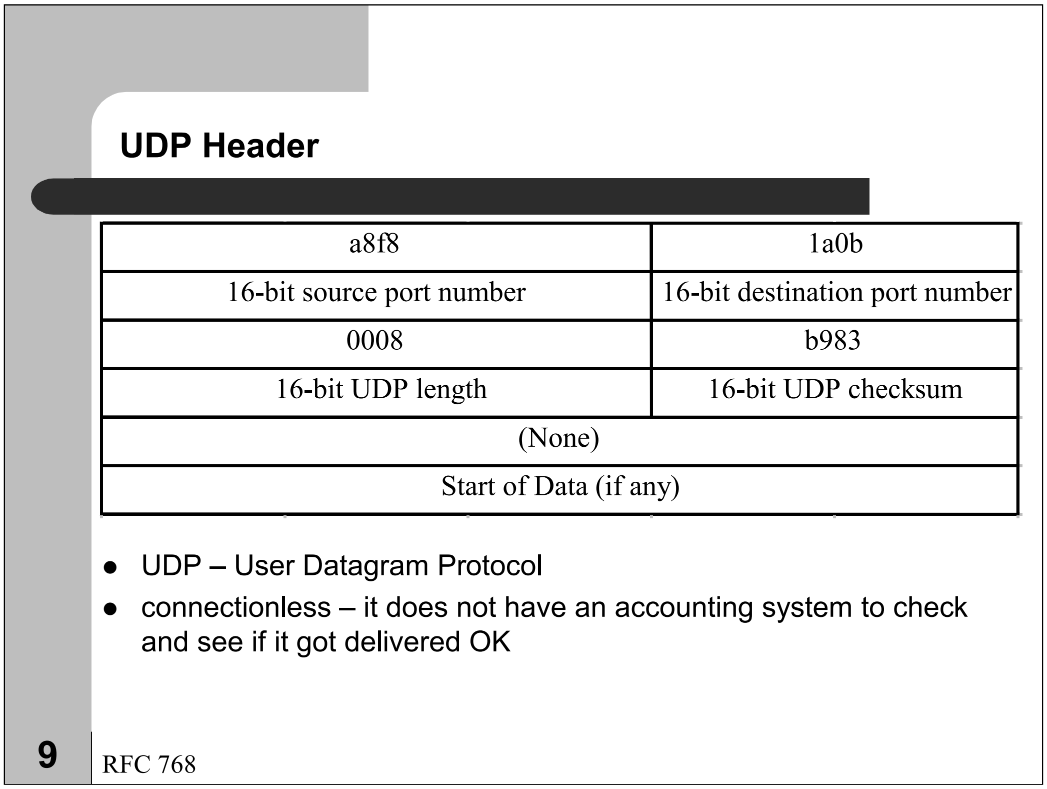
0829 (2089)								0017 (23)
16-bit source port number								16-bit destination port number
6f78 ae1b								
32-bit sequence number								
0000 0000								
32-bit acknowledgment number								
7 (28)	002 (syn flag set)							4000
4-bit header length	reserved (6 bits)	U	A	P	R	S	F	16-bit window size
27e4								0000
16-bit TCP checksum								16-urgent pointer
02 04 059c (MSS 1436) 01 (NOP) 01 (NOP) 04 02 (Sack Permitted)								
Options (if any)								
None								
Start of Data (if any)								

8 RFC 793, 1072, 1323

Example Packet:

```
08:38:29.231855 0:20:78:d3:53:34 0:2:fc:e:4c:40 0800 62: 192.168.0.1.2089 > 172.16.0.1.23: S [tcp sum ok]
1870179867:1870179867(0) win 16384 <mss 1436,nop,nop,sackOK> (DF) (ttl 128, id 35300, len 48)
   4500 0030 89e4 4000 8006 7b64 c0a8 0001
   ac10 0001 0829 0017 6f78 ae1b 0000 0000
   7002 4000 27e4 0000 0204 059c 0101 0402
```

The tricky field here is the 12th and 13th byte positions. We will discuss how to break up fields that don't end on a byte boundary in the following slides.



UDP Header

a8f8	1a0b
16-bit source port number	16-bit destination port number
0008	b983
16-bit UDP length	16-bit UDP checksum
(None)	
Start of Data (if any)	

- UDP – User Datagram Protocol
- connectionless – it does not have an accounting system to check and see if it got delivered OK

9 RFC 768

New Example Packet:

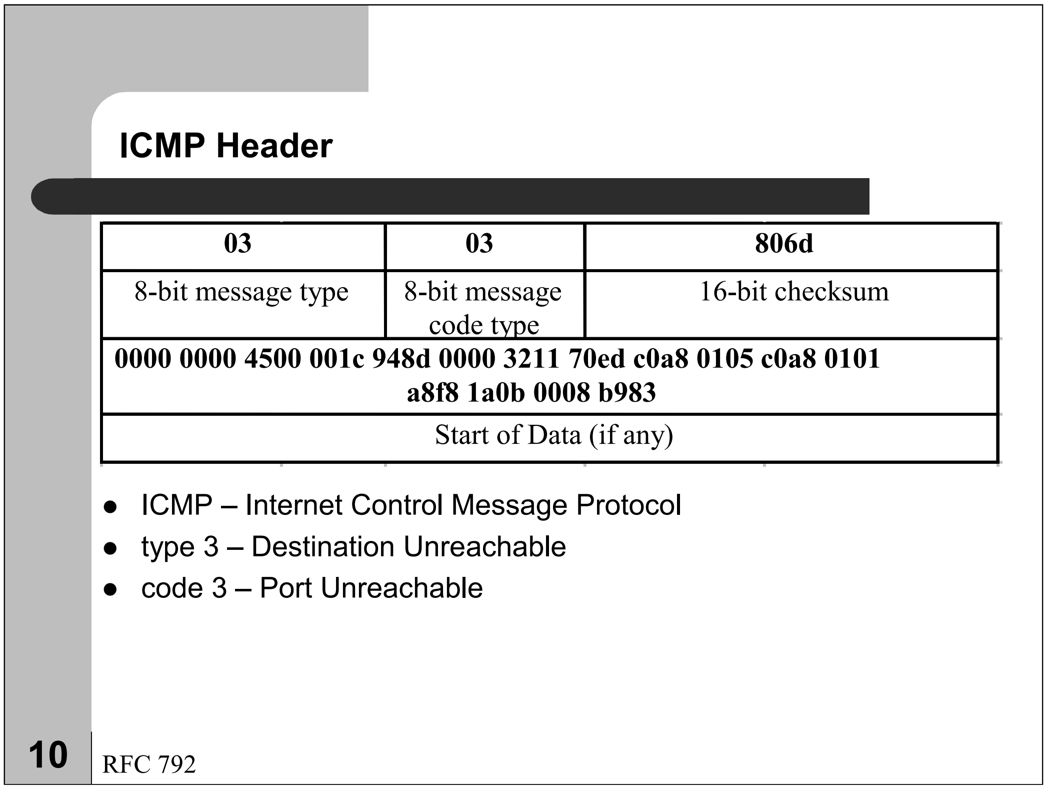
09:57:53.097634 0:d0:59:2b:46:96 0:20:78:d3:53:33 0800 42: 192.168.1.5.43256 > 192.168.1.1.6667: [udp sum ok]

udp 0 (ttl 50, id 38029, len 28)

  4500 001c 948d 0000 32**11** 70ed c0a8 0105

  c0a8 0101 **a8f8 1a0b 0008 b983**

Notice the bold 11. This is the protocol field in the IP header. Hex 11 converted to decimal is 17 which is the protocol value of UDP.



ICMP Header

03	03	806d
8-bit message type	8-bit message code type	16-bit checksum
0000 0000 4500 001c 948d 0000 3211 70ed c0a8 0105 c0a8 0101 a8f8 1a0b 0008 b983		
Start of Data (if any)		

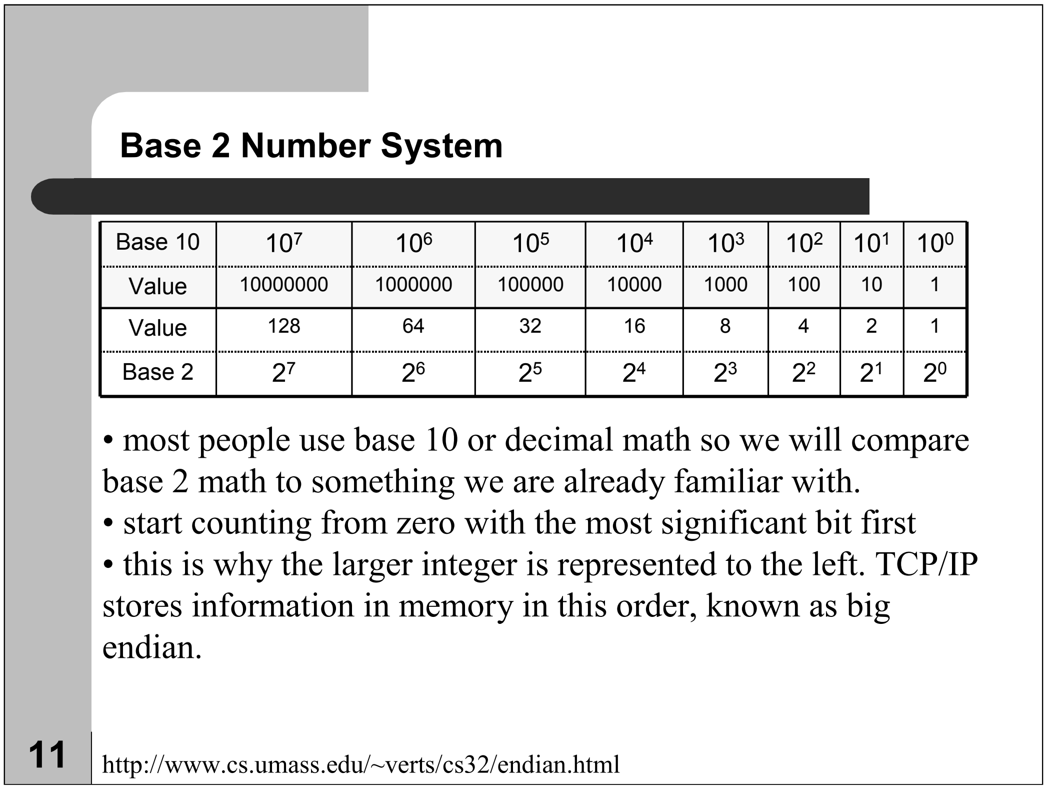
- ICMP – Internet Control Message Protocol
- type 3 – Destination Unreachable
- code 3 – Port Unreachable

10 RFC 792

New Sample Packet:

```
09:57:53.097998 0:20:78:d3:53:33 0:d0:59:2b:46:96 0800 70: 192.168.1.1 > 192.168.1.5: icmp:
192.168.1.1 udp port
6667 unreachable (ttl 64, id 38029, len 56)
    4500 0038 948d 0000 4001 62e1 c0a8 0101
    c0a8 0105 0303 806d 0000 0000 4500 001c
    948d 0000 3211 70ed c0a8 0105 c0a8 0101
    a8f8 1a0b 0008 b983
```

Notice the bold 01 in the IP header. This is the protocol value for ICMP. Also notice the original IP header and part of the UDP header are included as the data payload of the ICMP message. You can dig into this and get more information about what caused the ICMP message to be generated.



Base 2 Number System

Base 10	10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
Value	10000000	1000000	100000	10000	1000	100	10	1
Value	128	64	32	16	8	4	2	1
Base 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

- most people use base 10 or decimal math so we will compare base 2 math to something we are already familiar with.
- start counting from zero with the most significant bit first
- this is why the larger integer is represented to the left. TCP/IP stores information in memory in this order, known as big endian.

http://www.cs.umass.edu/~verts/cs32/endian.html

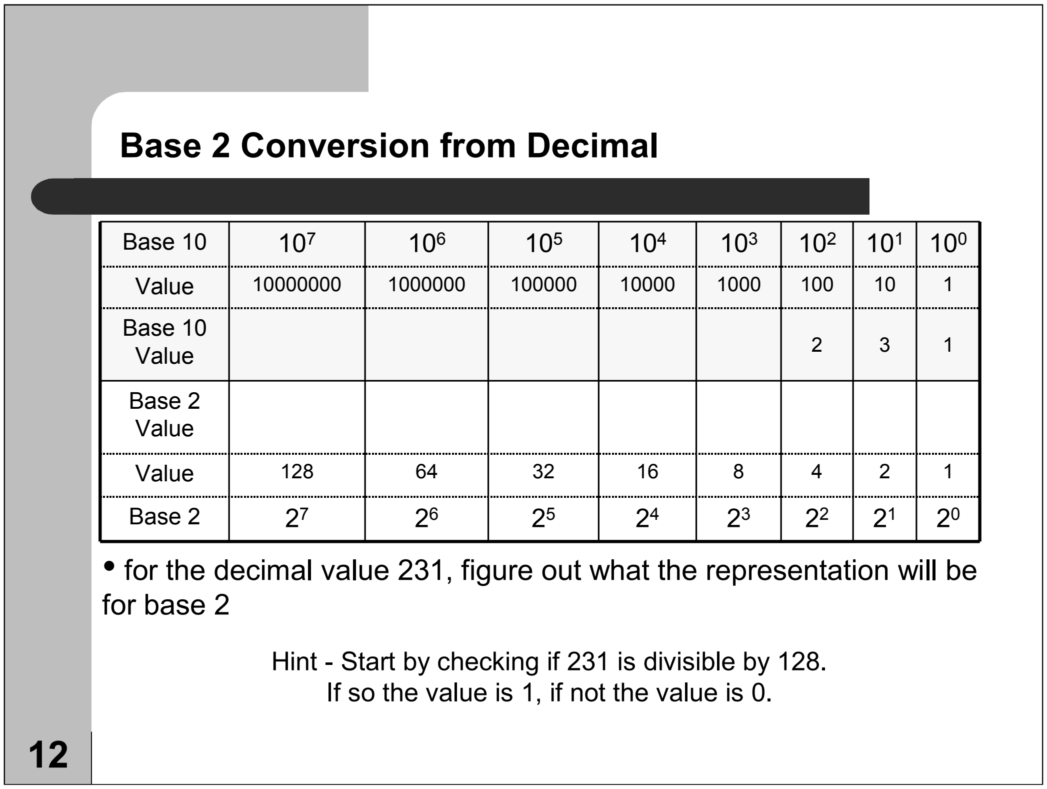
For every decimal position you increase the maximum value attainable by a multiple of the base for that number system. 10^4 has a value of 10000 or 10x10x10x10.

For instance if you take the number 6063 in decimal it would be broken down as follows:

1000’s = 6, 100’s = 0, 10’s = 6, Plus a remainder of 3.

The integer 3 was not high enough to increase the exponent of the 10’s therefore it is a remainder. This is because it is less than a whole unit. A unit in decimal is = 10.

The representation of decimal is flat so it would be represented as 6063 and that is why humans like base 10 math. Apply the same concepts to base 2 and we should be able to work through it.



Base 2 Conversion from Decimal

Base 10	10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
Value	10000000	1000000	100000	10000	1000	100	10	1
Base 10 Value						2	3	1
Base 2 Value								
Value	128	64	32	16	8	4	2	1
Base 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

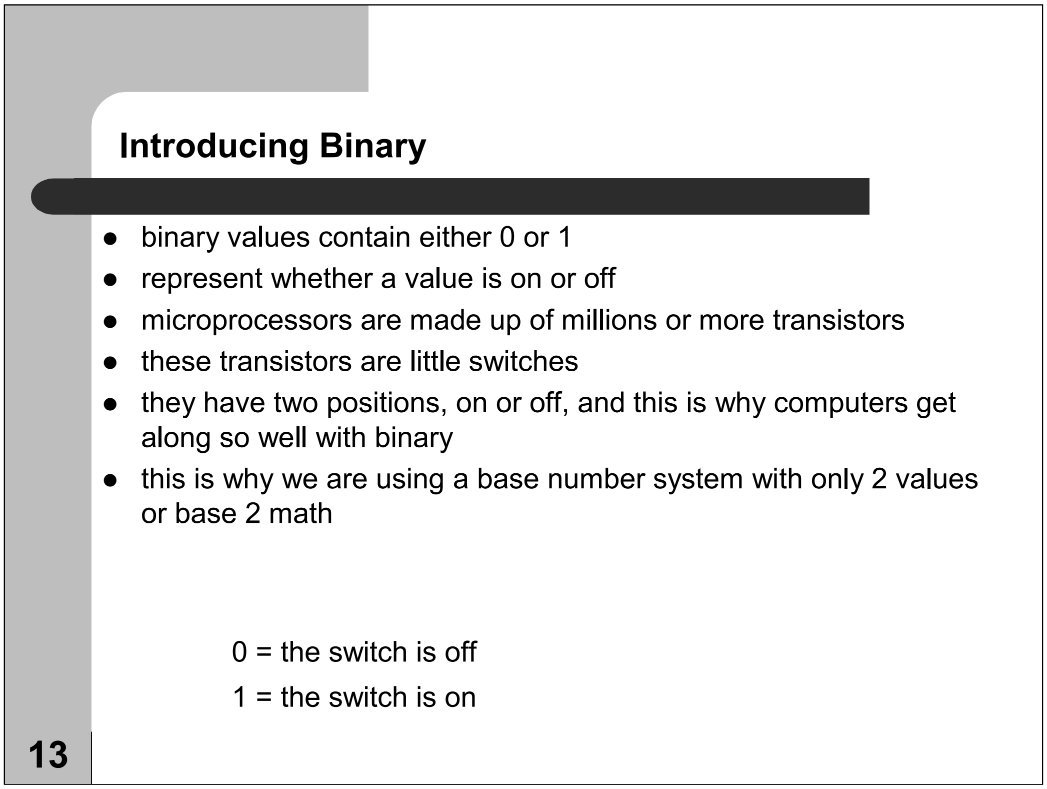
- for the decimal value 231, figure out what the representation will be for base 2

Hint - Start by checking if 231 is divisible by 128.
If so the value is 1, if not the value is 0.

12

Take the decimal value 231. Following the same steps as the previous slide lets break this down. Remember we start at 0 so each placeholder can only hold one “1” of its value. For instance we cannot have a value of 231 in the 2^0 placeholder because the exponent does not allow us to store a value higher than 1. The same reason applies to base 10. We could not express decimal 231 as 231 in the 10^0 placeholder because the value is higher than 10 and the placeholder can hold a maximum value equal to the base of the number system, in the case of decimal or base 10 the value is 10 (0 thru 9).

Answer is 11100111, therefore the representation of decimal 231 is 11100111. Hmmm, what does that look like?



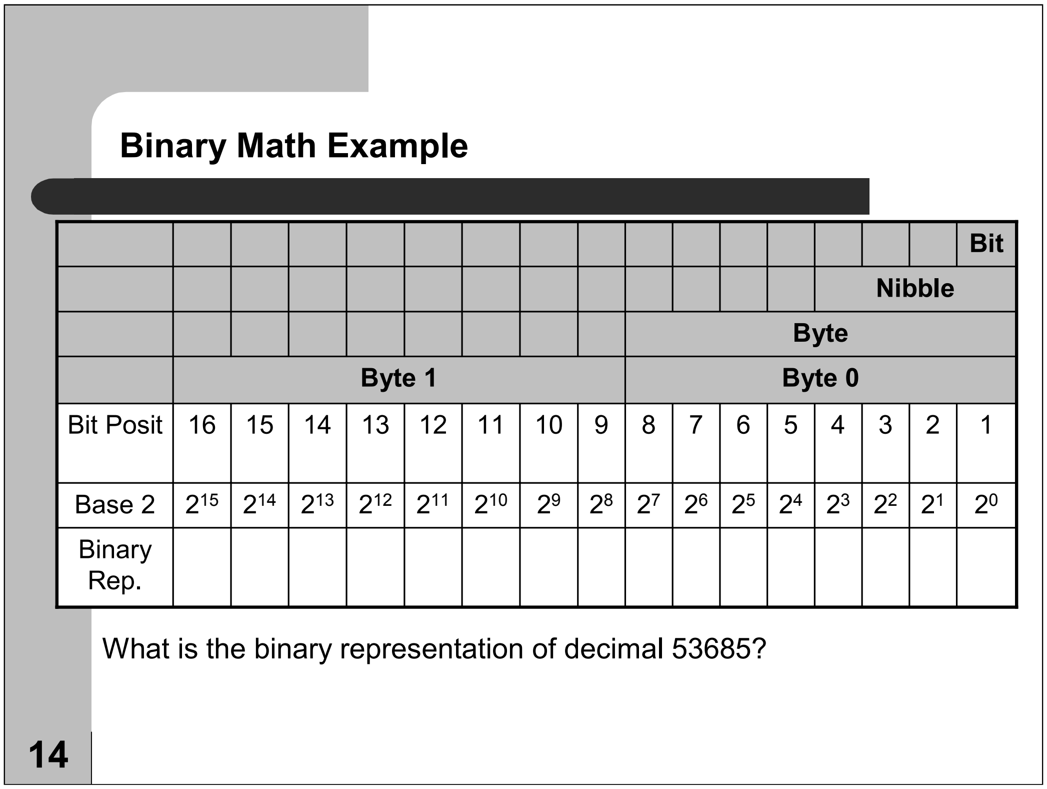
Introducing Binary

- binary values contain either 0 or 1
- represent whether a value is on or off
- microprocessors are made up of millions or more transistors
- these transistors are little switches
- they have two positions, on or off, and this is why computers get along so well with binary
- this is why we are using a base number system with only 2 values or base 2 math

0 = the switch is off

1 = the switch is on

This space intentionally left blank.



Binary Math Example

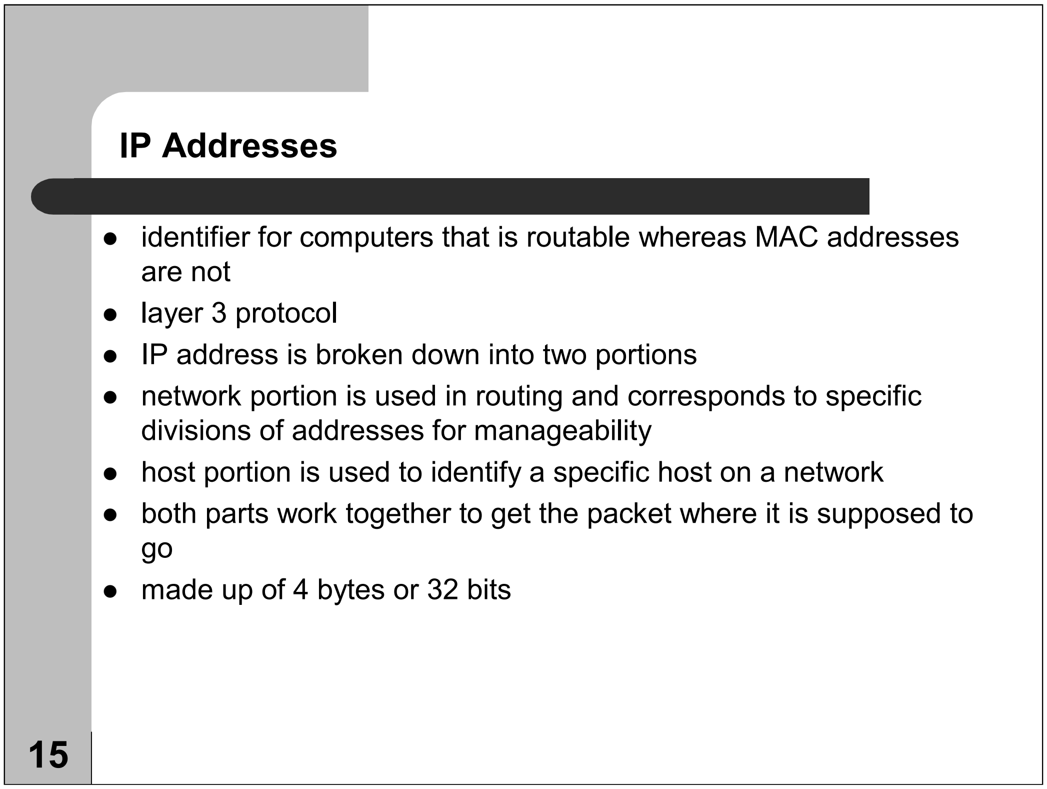
																Bit
													Nibble			
									Byte							
	Byte 1								**Byte 0**							
Bit Posit	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Base 2	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Binary Rep.																

What is the binary representation of decimal 53685?

14

$2^{15} = 32768$	Is 53685 divisible by 32768? Yes. 53685 - 32768 = 20917
$2^{14} = 16384$	Is 20917 divisible by 16384? Yes. 20917 - 16384 = 4533
$2^{13} = 8192$	Is 4533 divisible by 8192? No. Bit value 0. Next…
$2^{12} = 4096$	Is 4533 divisible by 4096? Yes. 4533 - 4096 = 437
$2^{11} = 2048$	Is 437 divisible by 2048? No. Bit value 0. Next…
$2^{10} = 1024$	Is 437 divisible by 1024? No. Bit value 0. Next…
$2^{9} = 512$	Is 437 divisible by 512? No. Bit value 0. Next…
$2^{8} = 256$	Is 437 divisible by 256? Yes. 437 - 256 = 181
$2^{7} = 128$	Is 181 divisible by 128? Yes. 181 - 128 = 53
$2^{6} = 64$	Is 53 divisible by 64? No. Bit value 0. Next…
$2^{5} = 32$	Is 53 divisible by 32? Yes. 53 - 32 = 21
$2^{4} = 16$	Is 21 divisible by 16? Yes. 21 - 16 = 5
$2^{3} = 8$	Is 5 divisible by 8? No. Bit value 0. Next…
$2^{2} = 4$	Is 5 divisible by 4? Yes. 5 - 4 = 1
$2^{1} = 2$	Is 1 divisible by 2? No. Bit value 0. Next…
$2^{0} = 1$	Is 1 divisible by 1? Yes. 1 - 1 = 0.

This should give us 1101000110110101 in binary.



IP Addresses

- identifier for computers that is routable whereas MAC addresses are not
- layer 3 protocol
- IP address is broken down into two portions
- network portion is used in routing and corresponds to specific divisions of addresses for manageability
- host portion is used to identify a specific host on a network
- both parts work together to get the packet where it is supposed to go
- made up of 4 bytes or 32 bits

IP addresses are broken down into network and host portions based on byte boundaries. These bytes are separated by a decimal. The addresses are broken down into these numbers because the high order bit of the binary address are turned on and separated from the remainder of the address by a 0 (except class A where the highest order bit must be 0). See below for an example.

Class A	0.0.0.0	-	127.255.255.255	(7 network bits/24 host bits)
Class B	128.0.0.0	-	191.255.255.255	(14 network bits/16 host bits)
Class C	192.0.0.0	-	223.255.255.255	(21 network bits/8 host bits)
Class D	224.0.0.0	-	239.255.255.255	(multicast 28 bits for ID)
Class E	240.0.0.0	-	255.255.255.255	(reserved 28 bits)

Class A **0111**1111 = 127, highest decimal value of a class A network

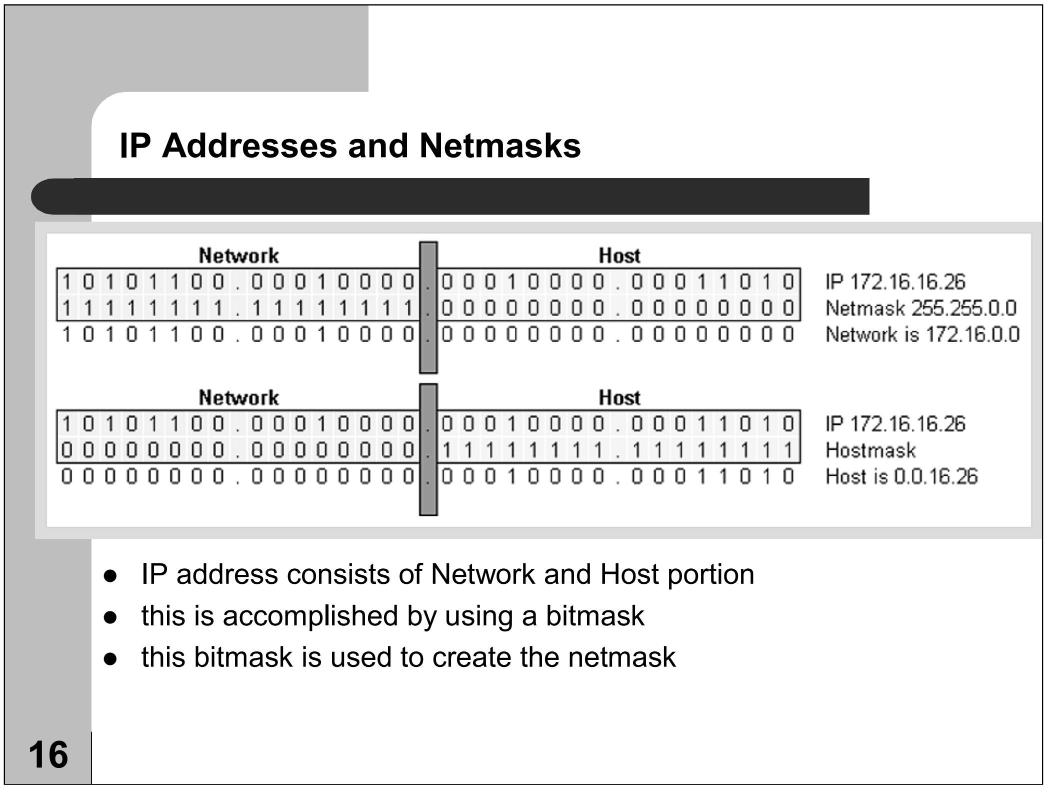
Class B **1011**1111 = 191, highest decimal value of a class A network

Class C **1101**1111 = 223, highest decimal value of a class A network

Class D **1110**1111 = 239, highest decimal value of a class A network

Class E **1111**1111 = 255, highest decimal value of a class A network

Notice the pattern in the bold section above. The “0” slides to the right through the 4 routable address ranges.



IP Addresses and Netmasks

Network	Host	
1 0 1 0 1 1 0 0 . 0 0 0 1 0 0 0 0	. 0 0 0 1 0 0 0 0 . 0 0 0 1 1 0 1 0	IP 172.16.16.26
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1	. 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	Netmask 255.255.0.0
1 0 1 0 1 1 0 0 . 0 0 0 1 0 0 0 0	. 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	Network is 172.16.0.0

Network	Host	
1 0 1 0 1 1 0 0 . 0 0 0 1 0 0 0 0	. 0 0 0 1 0 0 0 0 . 0 0 0 1 1 0 1 0	IP 172.16.16.26
0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	. 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1	Hostmask
0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0	. 0 0 0 1 0 0 0 0 . 0 0 0 1 1 0 1 0	Host is 0.0.16.26

- IP address consists of Network and Host portion
- this is accomplished by using a bitmask
- this bitmask is used to create the netmask

16

Modulo Math

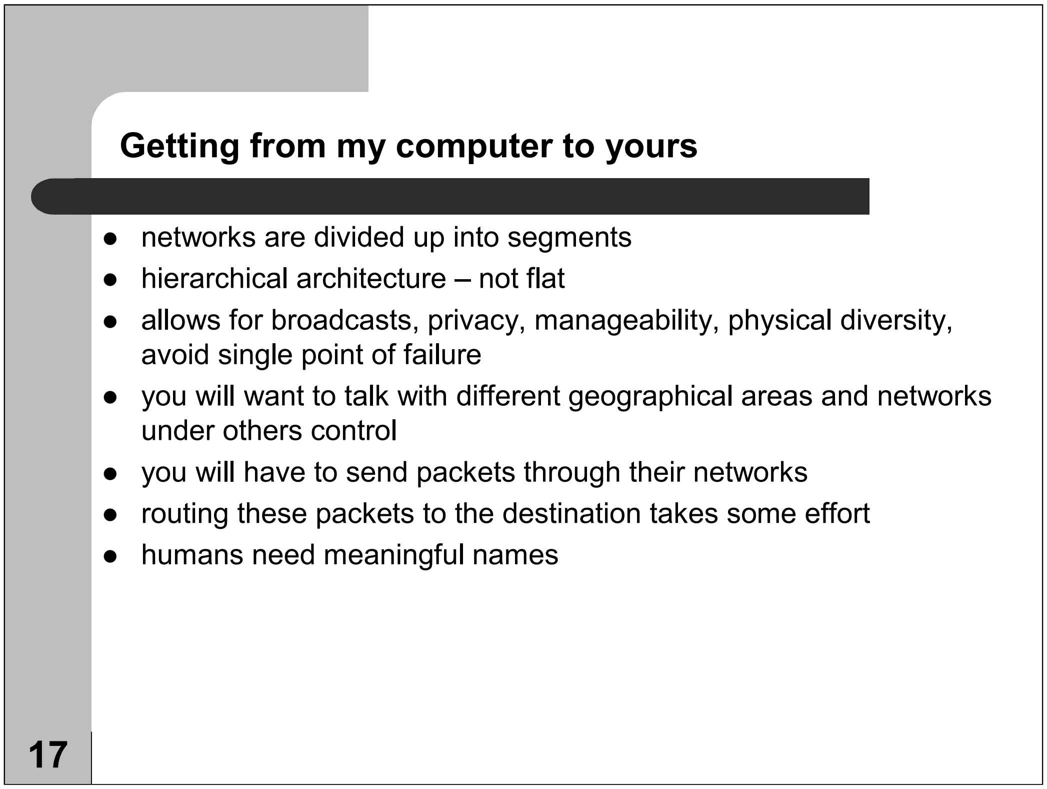
Bits are and'ed together (Boolean and operator). The result will mask what you want to keep and discard what you don't want to keep. In the examples above the 1's in the original IP address are transferred down while the bitmask was 1's. Where the mask had 0's, the original bits were dropped, leaving a result the desired result – indicating what the Network and Host ID's were.

0 + 0 = 0 example 2: 11101100

0 + 1 = 0 +11110000

1 + 0 = 0 11100000

1 + 1 = 1



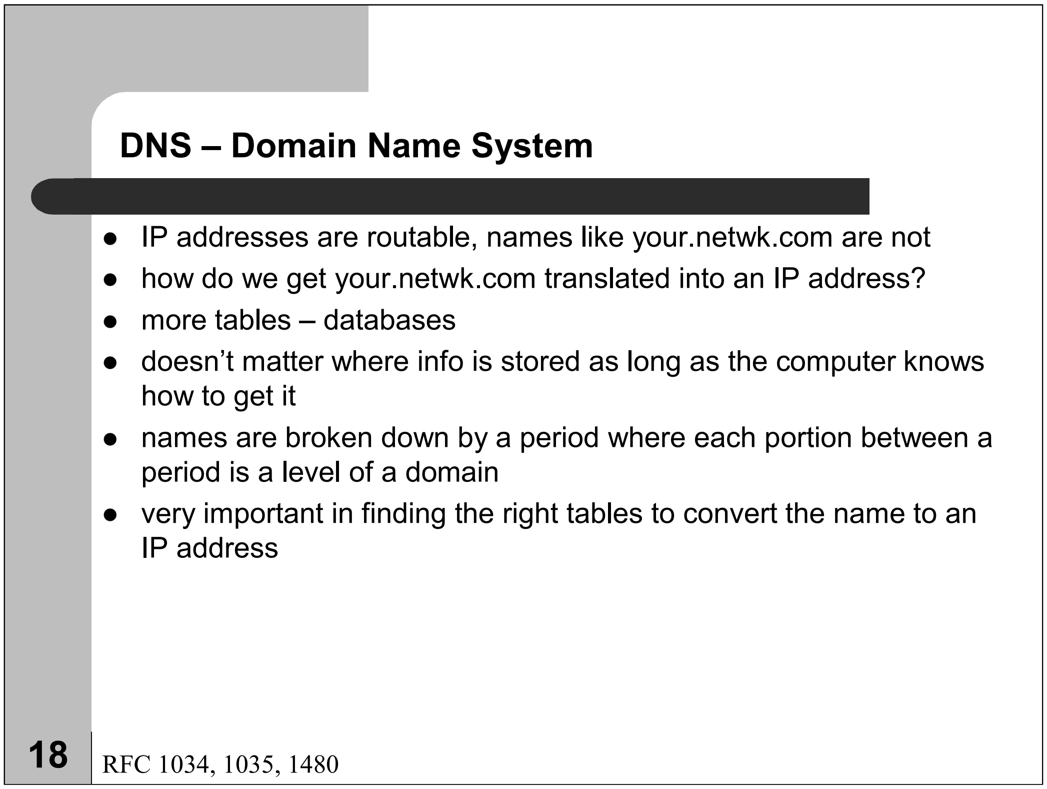
Getting from my computer to yours

- networks are divided up into segments
- hierarchical architecture – not flat
- allows for broadcasts, privacy, manageability, physical diversity, avoid single point of failure
- you will want to talk with different geographical areas and networks under others control
- you will have to send packets through their networks
- routing these packets to the destination takes some effort
- humans need meaningful names

17

How many URL's can you think of off the top of your head? Do you think you would be able to remember them if they were in the form of a number between 1 and 4294967295? If you could remember all the numbers, it would be quite a feat not to mix them up with the wrong site.

Enter Domain Name System…



DNS – Domain Name System

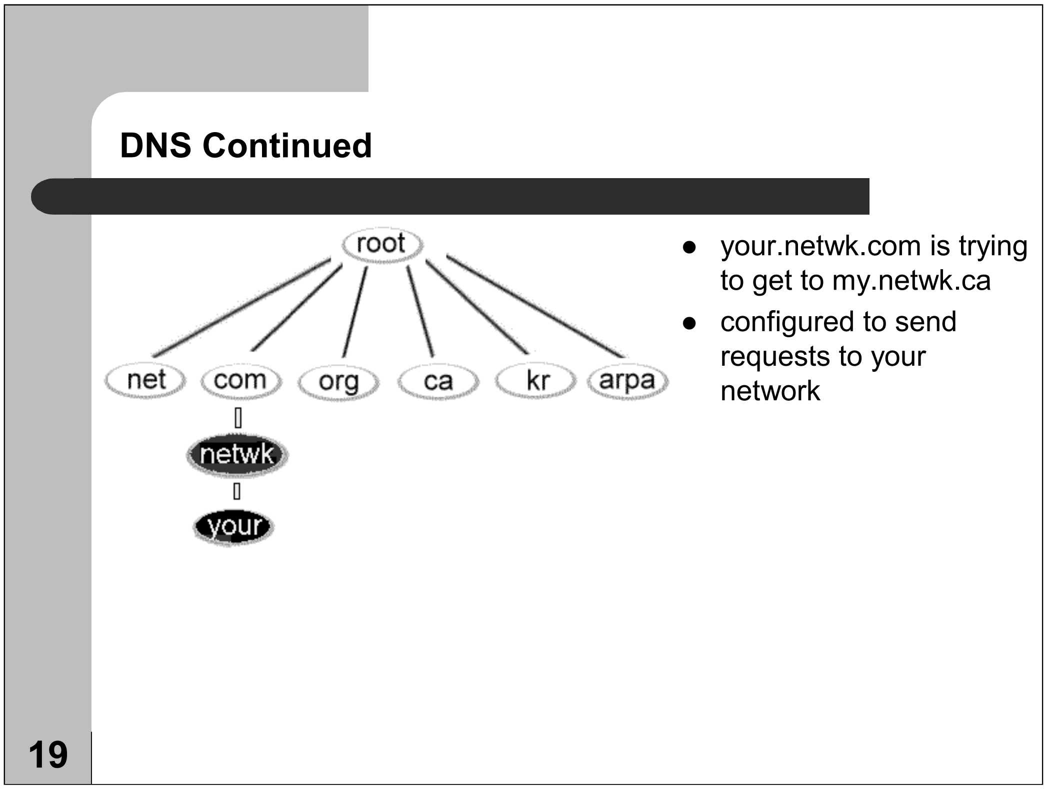
- IP addresses are routable, names like your.netwk.com are not
- how do we get your.netwk.com translated into an IP address?
- more tables – databases
- doesn’t matter where info is stored as long as the computer knows how to get it
- names are broken down by a period where each portion between a period is a level of a domain
- very important in finding the right tables to convert the name to an IP address

18 RFC 1034, 1035, 1480

Name Resolution is done in numerous ways. A table can reside on your.netwk.com and can specify what addresses resolve to what IPs’. This is particularly fast as nothing needs to be sent over the network.

The next slide describes how to get this information back when you do not know locally what the IP of a host is but you have the fully qualified domain name (FQDN).

FQDN example: www.sans.org



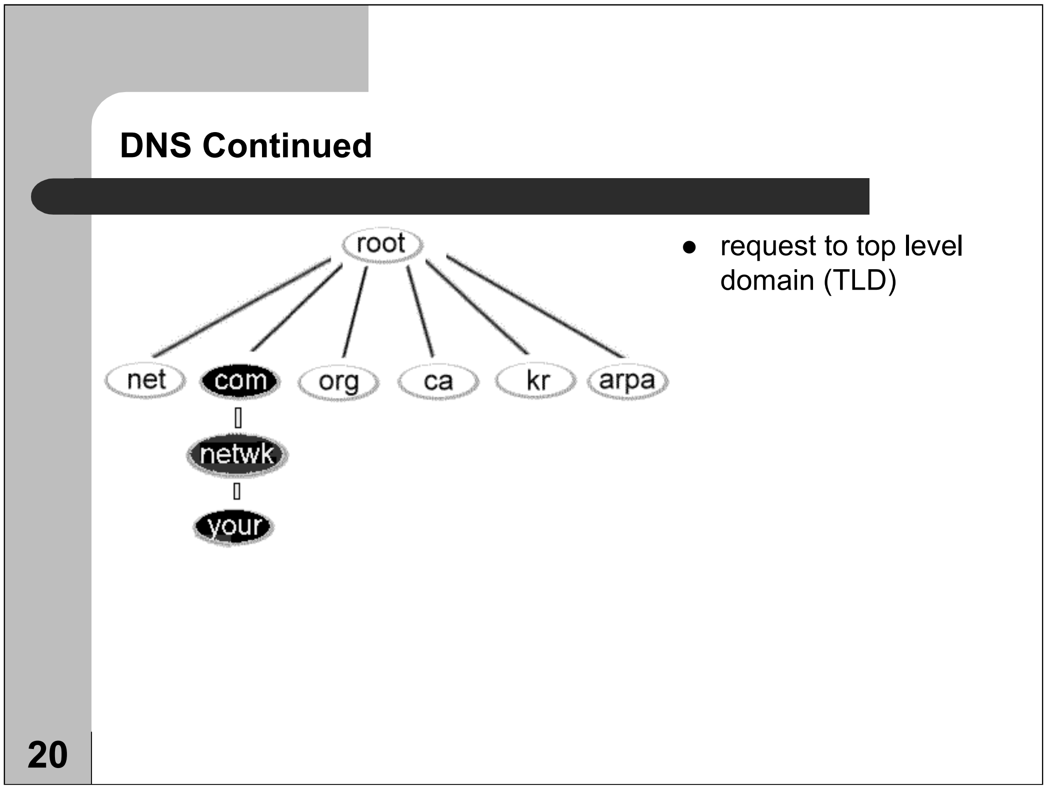
DNS Continued

- your.netwk.com is trying to get to my.netwk.ca
- configured to send requests to your network

19

Top Level Domains (TLD’s)

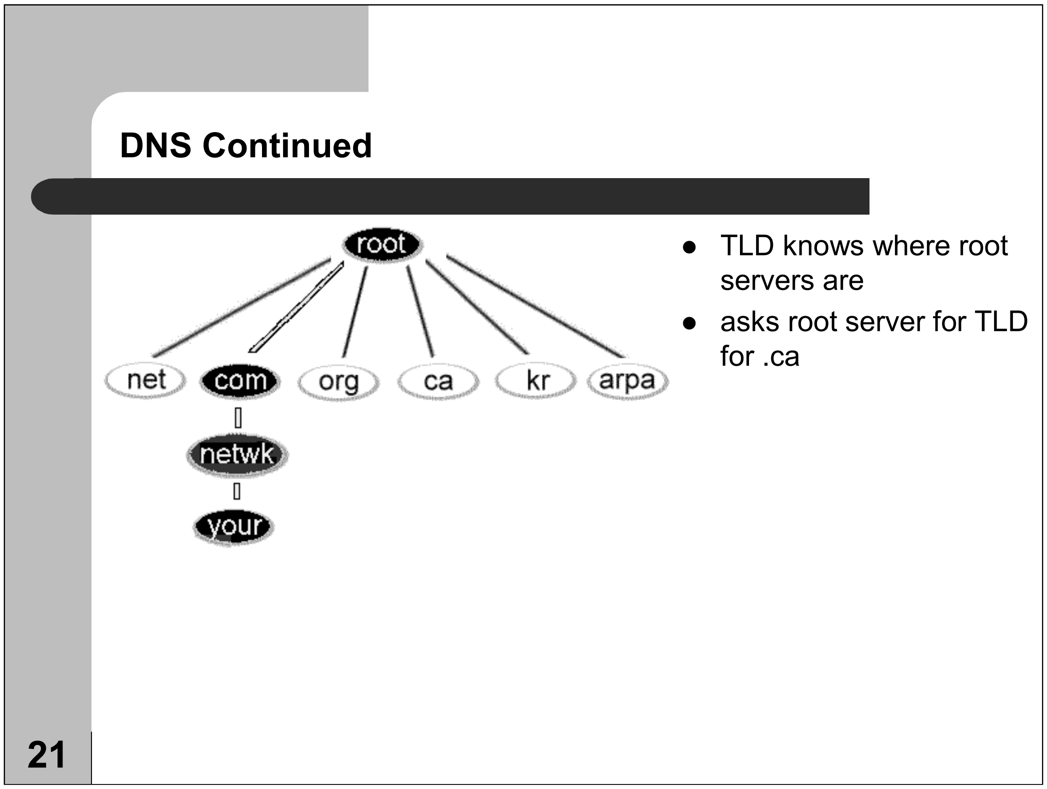
com, edu, gov, int, mil, net, org, arpa, and country specific which are two characters vice three.



DNS Continued

- request to top level domain (TLD)

This space intentionally left blank.

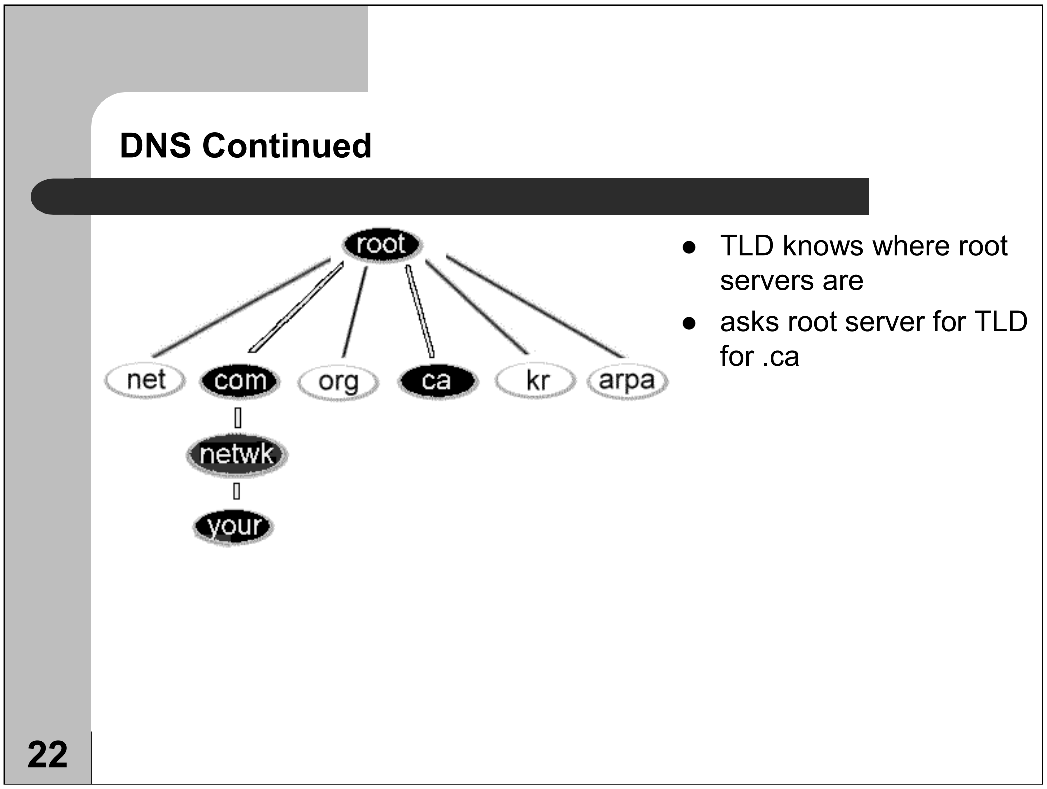


DNS Continued

- TLD knows where root servers are
- asks root server for TLD for .ca

21

This space intentionally left blank.

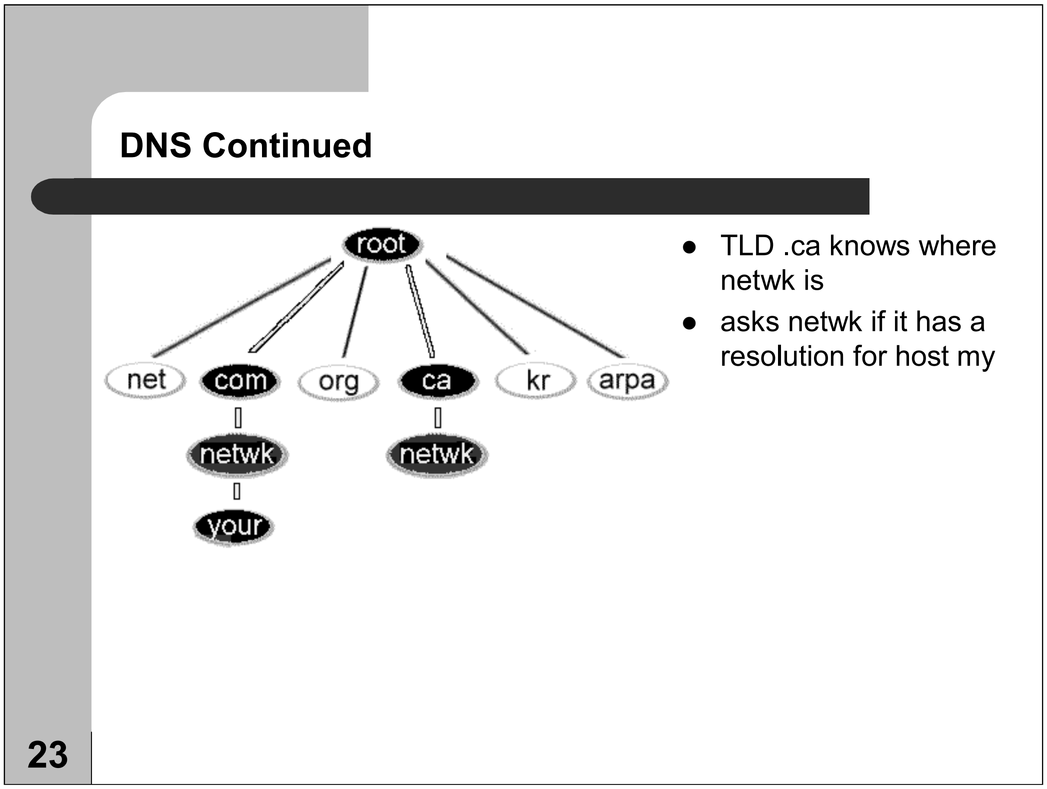


DNS Continued

- TLD knows where root servers are
- asks root server for TLD for .ca

22

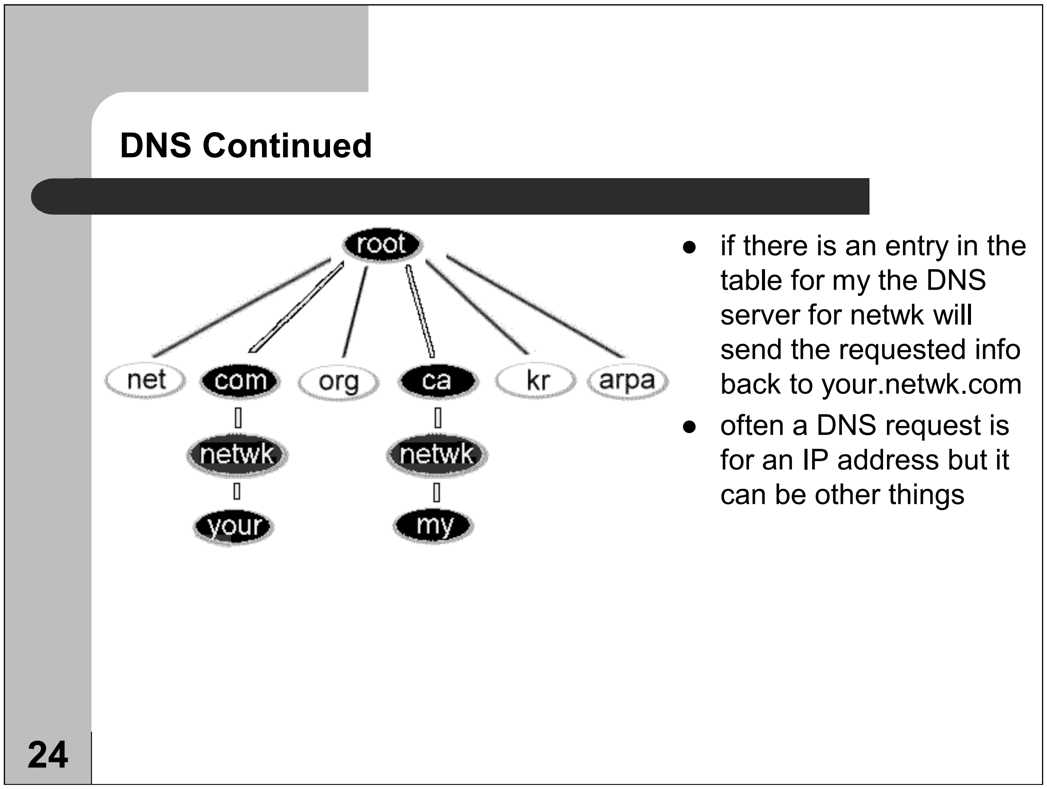
This space intentionally left blank.



DNS Continued

- TLD .ca knows where netwk is
- asks netwk if it has a resolution for host my

This space intentionally left blank.

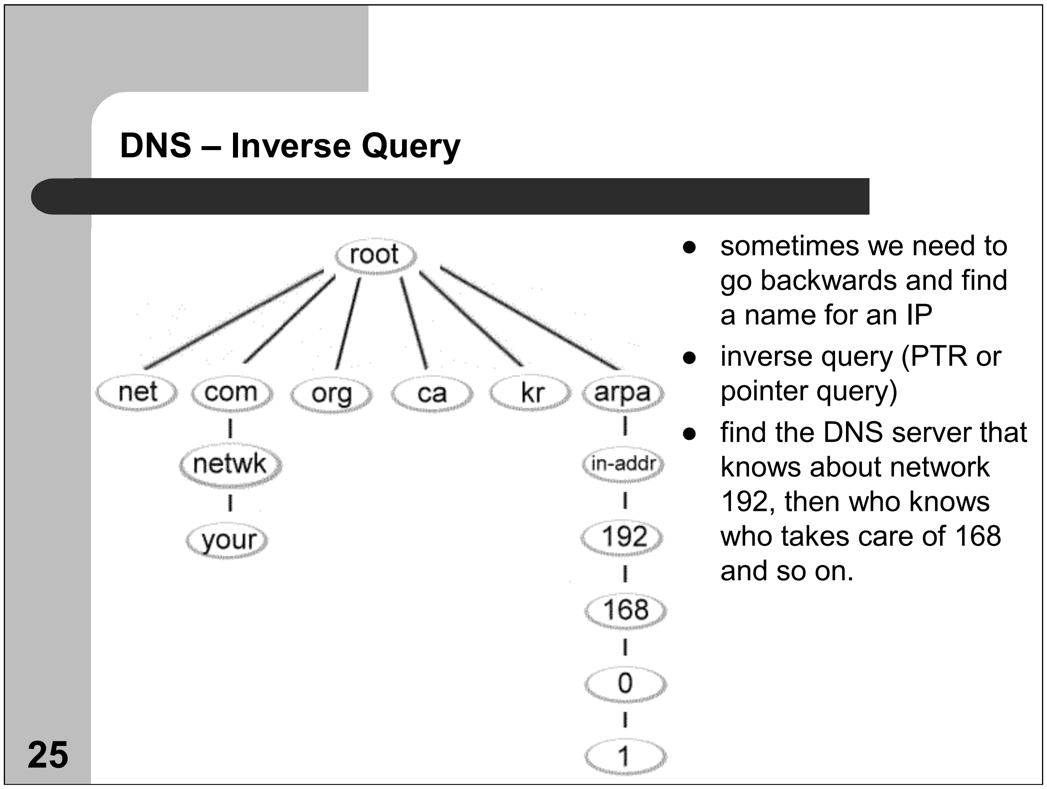


DNS Continued

- if there is an entry in the table for my the DNS server for netwk will send the requested info back to your.netwk.com
- often a DNS request is for an IP address but it can be other things

24

This space intentionally left blank.

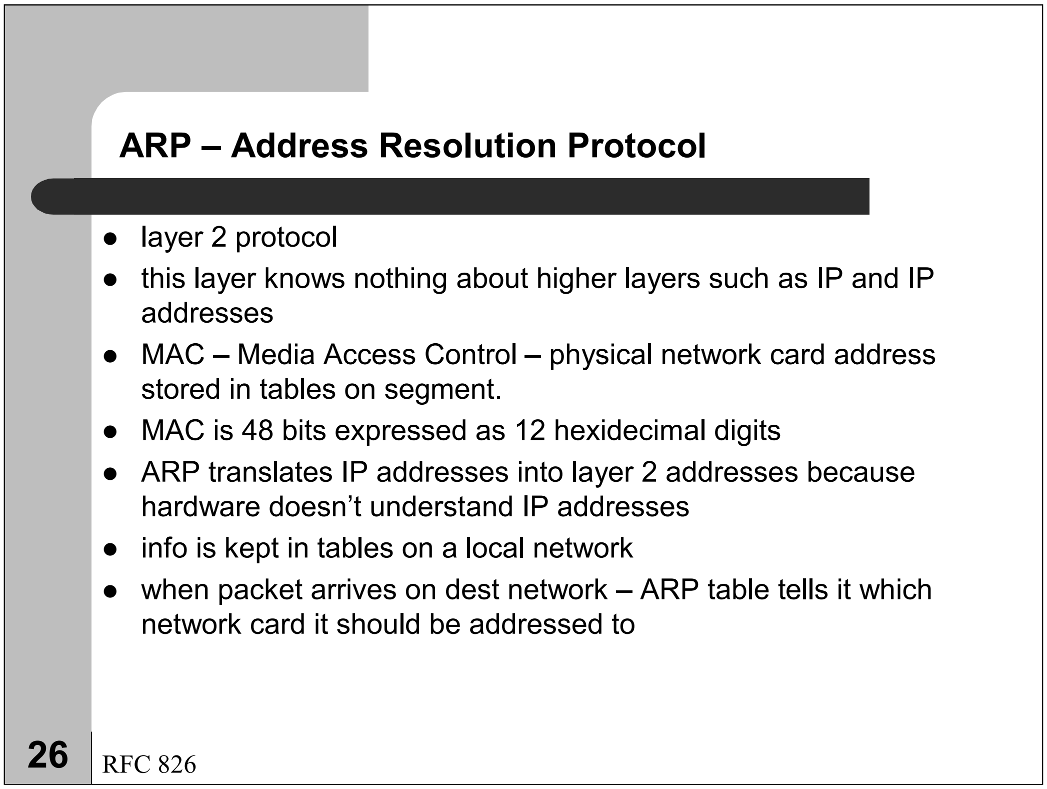


DNS – Inverse Query

- sometimes we need to go backwards and find a name for an IP
- inverse query (PTR or pointer query)
- find the DNS server that knows about network 192, then who knows who takes care of 168 and so on.

25

The response back from an in-addr query will be in the form host.network.tld.in-addr.arpa. This is because each server along the way will resolve their own portion and forward it on to the next one in line who will then add their information. Your IP address is already in the IP header so it will know how to get back to you. The DNS server that knows about your host will send a reply back to the originator of the query. If nothing is known then an error is returned from the DNS server actioning the query indicating the IP is a non-existent domain.



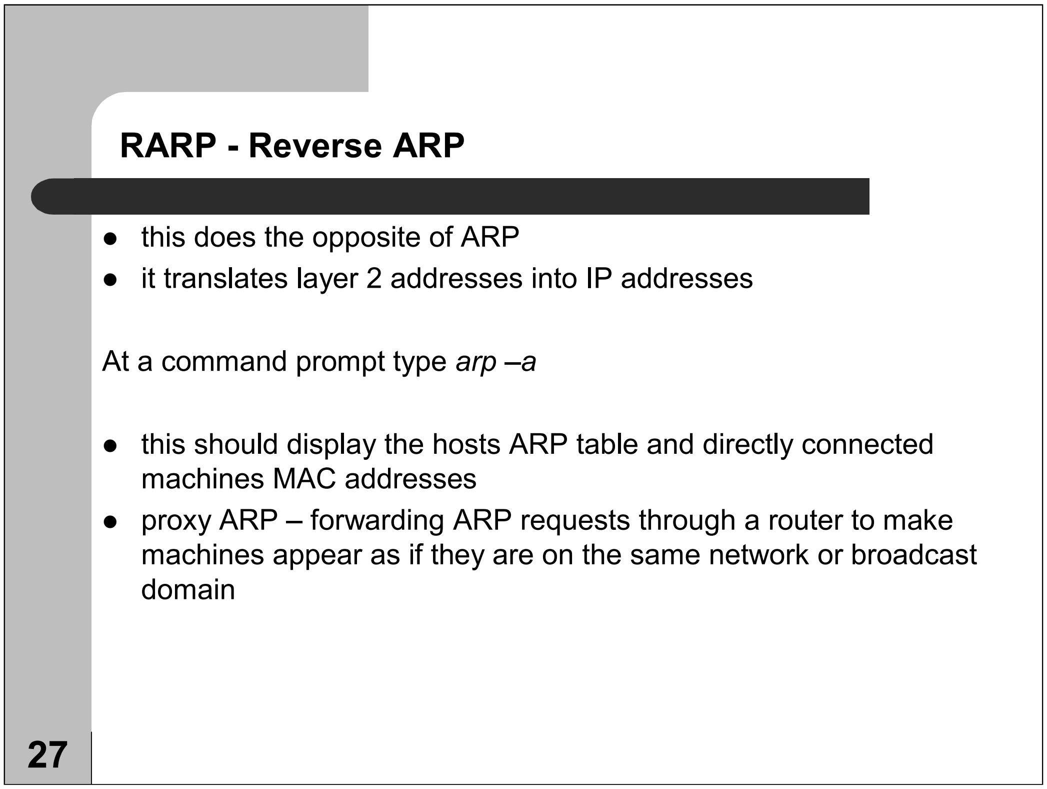
ARP – Address Resolution Protocol

- layer 2 protocol
- this layer knows nothing about higher layers such as IP and IP addresses
- MAC – Media Access Control – physical network card address stored in tables on segment.
- MAC is 48 bits expressed as 12 hexidecimal digits
- ARP translates IP addresses into layer 2 addresses because hardware doesn't understand IP addresses
- info is kept in tables on a local network
- when packet arrives on dest network – ARP table tells it which network card it should be addressed to

26 RFC 826

At each router, the MAC address is changed. The previous layer 2 header is stripped off and disguarded, a new header is placed on the packet, checksum calculated, and it is sent on its way. This is an important point to understand when working with logs containing layer 2 headers.

It can be used to help solve routing problems and to find hosts on a local segment that are not traceable via their IP address. MAC addresses are burnt into a chip on network cards and remain static. For this reason it is a fairly reliable source for tracking down a computer. The first 24 bits of a MAC address correspond to a hardware manufacturers assigned number, therefore it is possible to find out some information about the transmitting host by its MAC address.



RARP - Reverse ARP

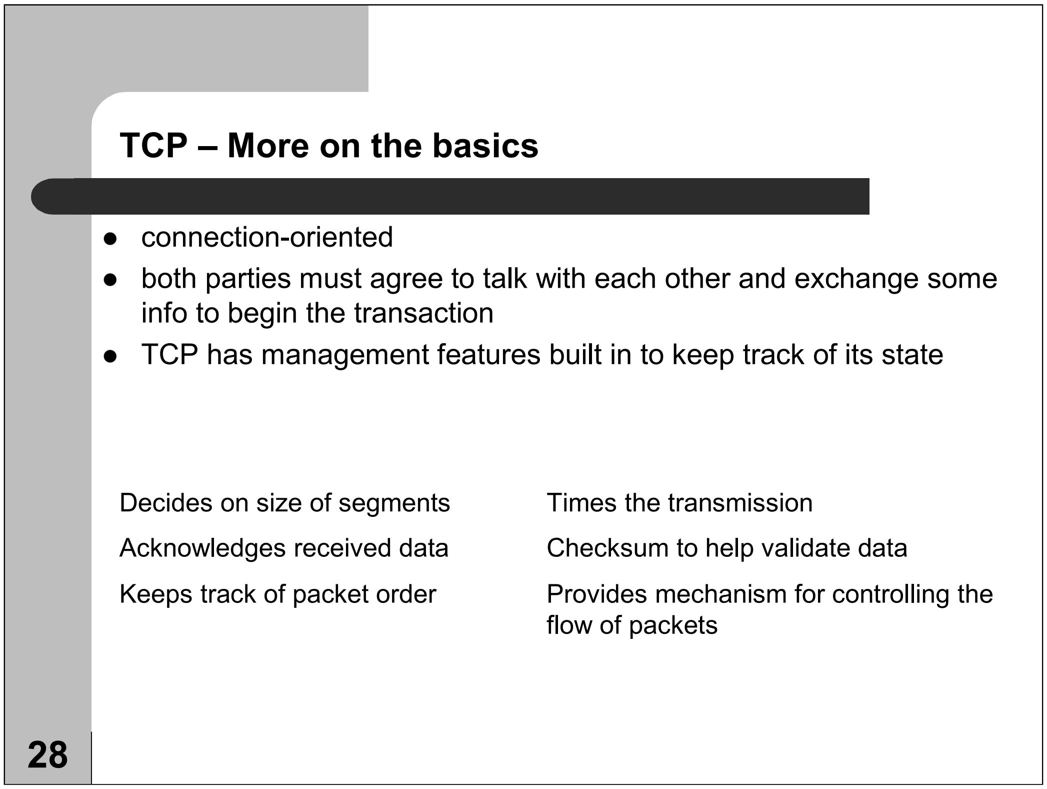
- this does the opposite of ARP
- it translates layer 2 addresses into IP addresses

At a command prompt type *arp –a*

- this should display the hosts ARP table and directly connected machines MAC addresses
- proxy ARP – forwarding ARP requests through a router to make machines appear as if they are on the same network or broadcast domain

27

This space intentionally left blank.



TCP – More on the basics

- connection-oriented
- both parties must agree to talk with each other and exchange some info to begin the transaction
- TCP has management features built in to keep track of its state

Decides on size of segments

Acknowledges received data

Keeps track of packet order

Times the transmission

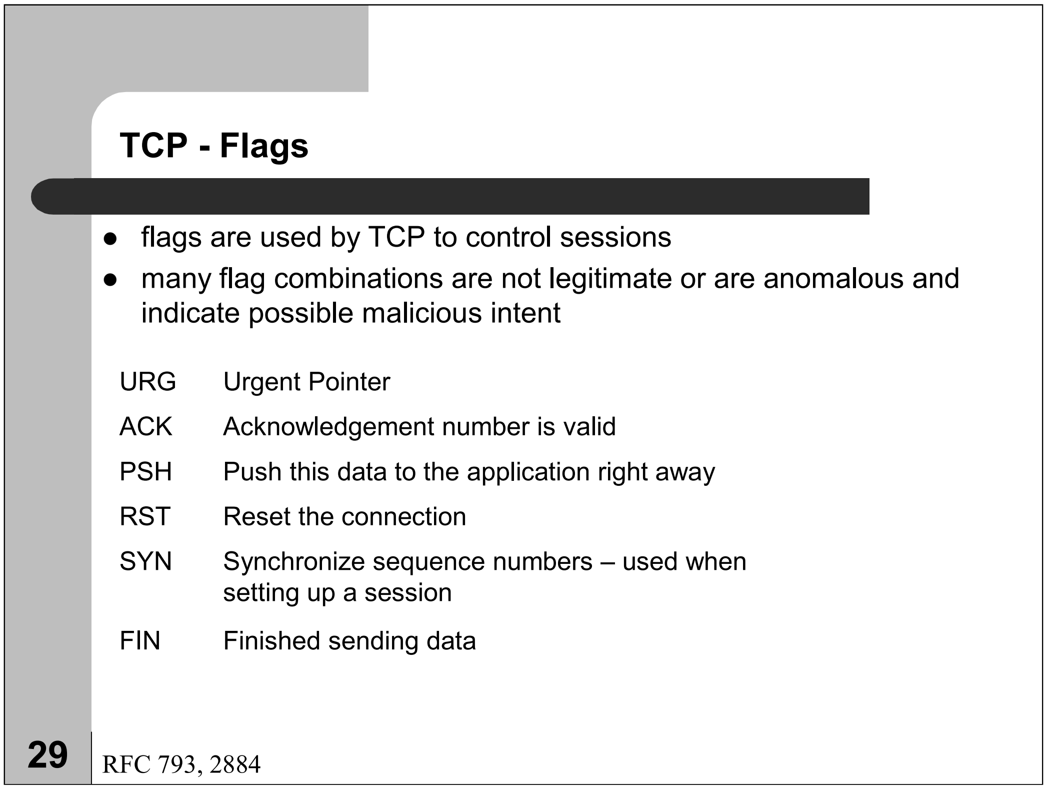
Checksum to help validate data

Provides mechanism for controlling the flow of packets

28

If something goes wrong with a transmission, TCP will know and try to correct the problem. For example, if a packet was corrupt and dropped by the destination host, after a time out period (no acknowledgement from the destination host), the packet and any subsequent packets would be retransmitted.

The size of data transferred prior to an acknowledgement is often negotiated by both communication end-points when setting up a session.



TCP - Flags

- flags are used by TCP to control sessions
- many flag combinations are not legitimate or are anomalous and indicate possible malicious intent

URG	Urgent Pointer
ACK	Acknowledgement number is valid
PSH	Push this data to the application right away
RST	Reset the connection
SYN	Synchronize sequence numbers – used when setting up a session
FIN	Finished sending data

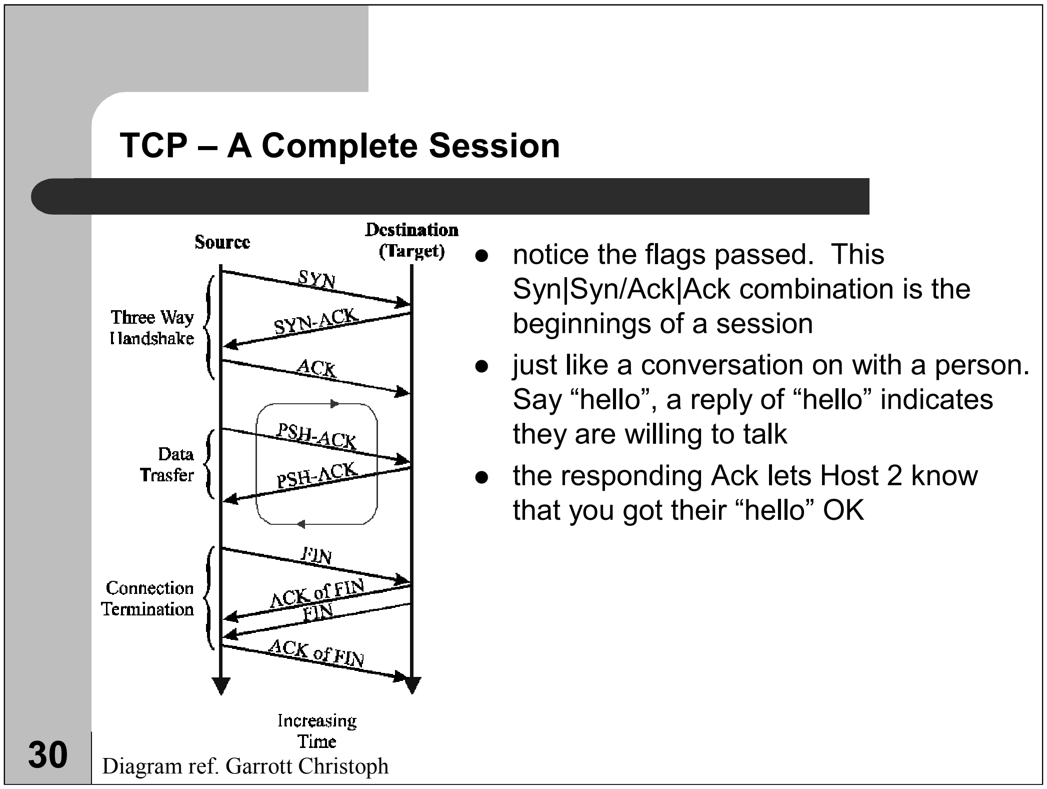
29 RFC 793, 2884

There are 12 bits within the TCP header located in bytes 12 and 13 that could be set in differing combinations to create anomalous packets. Six reserved bits and six flags. Normal communications via TCP generally only use the six valid flags presented in the slide above.

Recently the two reserved bits in the 13th byte of the TCP header began being legitimately used by Explicit Congestion Notification or ECN. The most common and standard flags used in the TCP protocol are listed above.

More information on ECN you can be found at the following reference:

http://www.aciri.org/floyd/ecn.html



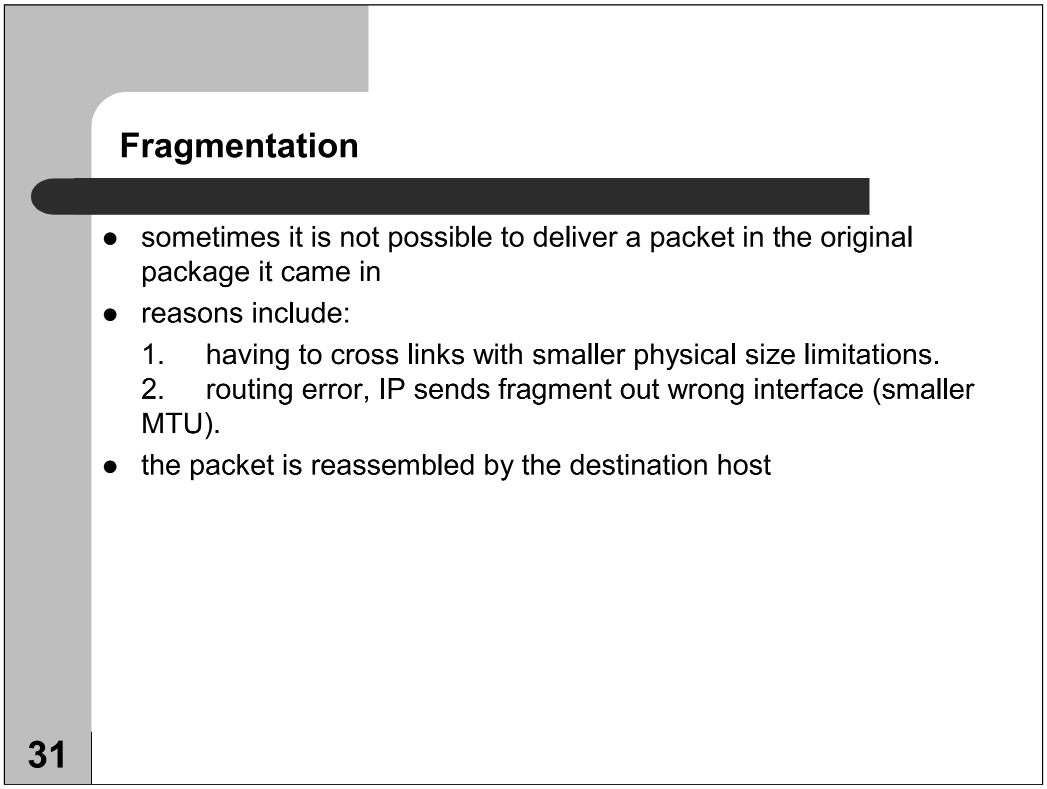
TCP – A Complete Session

- notice the flags passed. This Syn|Syn/Ack|Ack combination is the beginnings of a session
- just like a conversation on with a person. Say “hello”, a reply of “hello” indicates they are willing to talk
- the responding Ack lets Host 2 know that you got their “hello” OK

30

Diagram ref. Garrott Christoph

This diagram depicts opening, passing data in, and closing a TCP session. Get used to seeing this. Normal TCP communications adhere to these types of exchanges during normal traffic. If a different set of flags is noted, it likely is in response to abnormal stimulus. This is the model that TCP follows.



Fragmentation

- sometimes it is not possible to deliver a packet in the original package it came in
- reasons include:
  1. having to cross links with smaller physical size limitations.
  2. routing error, IP sends fragment out wrong interface (smaller MTU).
- the packet is reassembled by the destination host

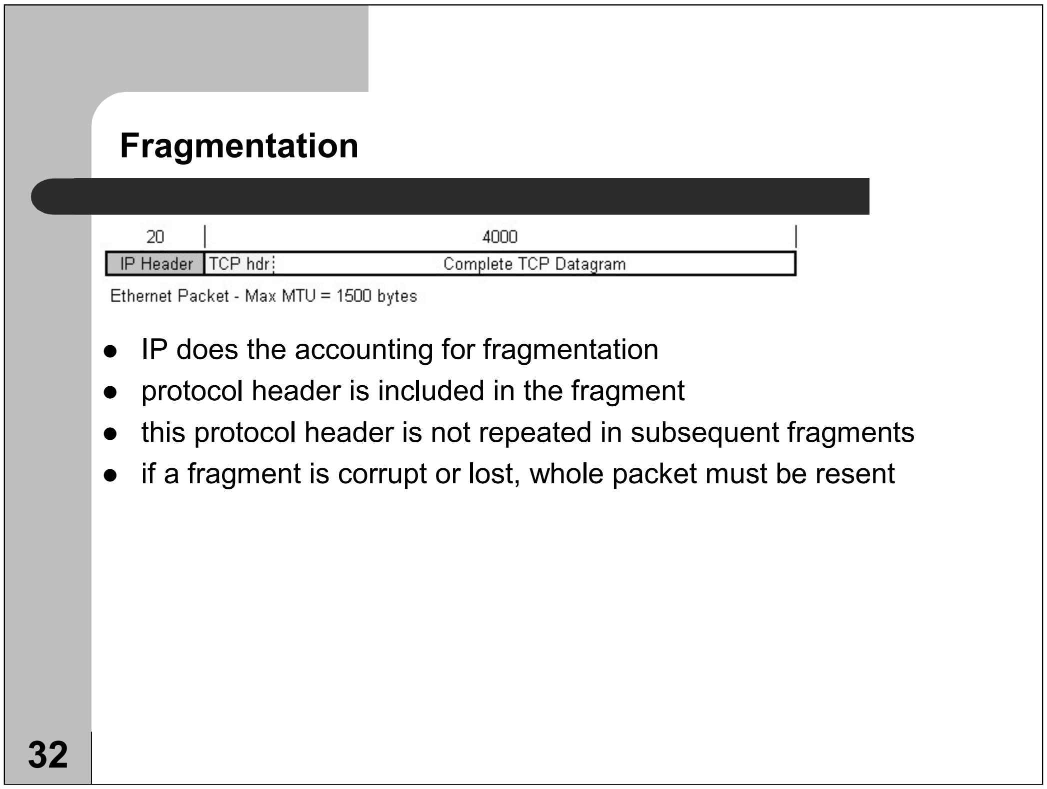
31

Maximum Transmission Unit (MTU) is most often decided upon by the physical characteristics of the medium in which the packet is transmitted on. Different types of wire, metals, and other mediums have physical properties that affect what frequency the medium can handle. Things such as amplitude and frequency affect the maximum MTU. An MTU may be reduced from the maximum physical specifications, usually through modifying variables in an operating systems TCP/IP stack settings.

Common maximum MTU's are:

Point-to-point (low delay)	296 bytes
X.25	576 bytes
IEEE 802.2/802.3	1492 bytes
Ethernet	1500 bytes
FDDI	4352 bytes
4 Mbits/sec token ring (IEEE 802.5)	4464 bytes
16 Mbits/sec token ring (IBM)	17914 bytes
Hyperchannel	65535 bytes

In IPV4 the maximum value the length field of a packet can have is 65535 bytes. Most corporate networks today are Ethernet.



Fragmentation

20 | 4000

IP Header | TCP hdr | Complete TCP Datagram

Ethernet Packet - Max MTU = 1500 bytes

- IP does the accounting for fragmentation
- protocol header is included in the fragment
- this protocol header is not repeated in subsequent fragments
- if a fragment is corrupt or lost, whole packet must be resent

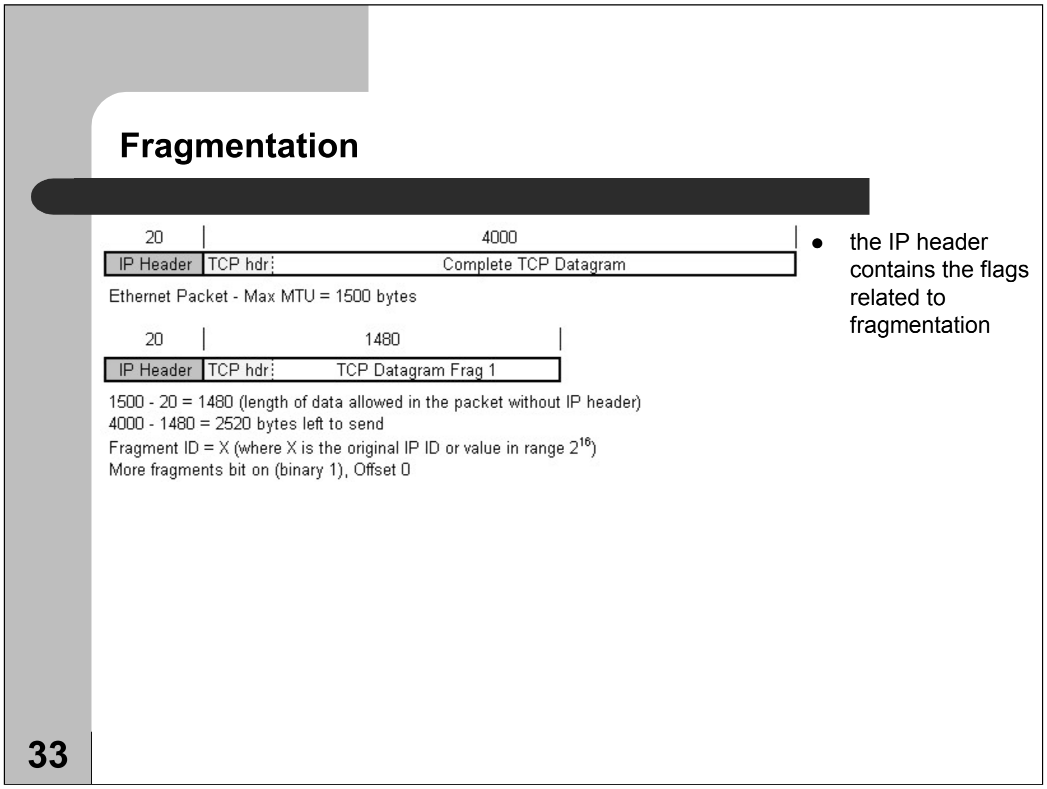
32

Fragmentation is commonly used maliciously to cause denial of service conditions on machines that must reassemble packets. This could be a stateful firewall, IDS, the destination host or any other machine that is required to inspect packets and put them back together.

Fragmentation is commonly used to try and evade IDS' by splitting up content strings between packets, for example two packets, fragment 1 with "localh" and "ost" in fragment 2 would not trigger a signature looking for "localhost" all together.

Fragments can be used to cause out of band denial of service attacks. This is accomplished by taking advantage of deficiencies in the operating systems' (OS') error handling mechanisms for fragmented packets. Forged fragments may cause memory corruption or operating system failure. Most modern operating systems are not vulnerable to these types of attacks anymore.

Yet one more commonly used exploitable feature of fragmentation is in OS fingerprinting. It is possible to determine what the OS is depending upon how it handles fragmentation and fragmented packets.



Fragmentation

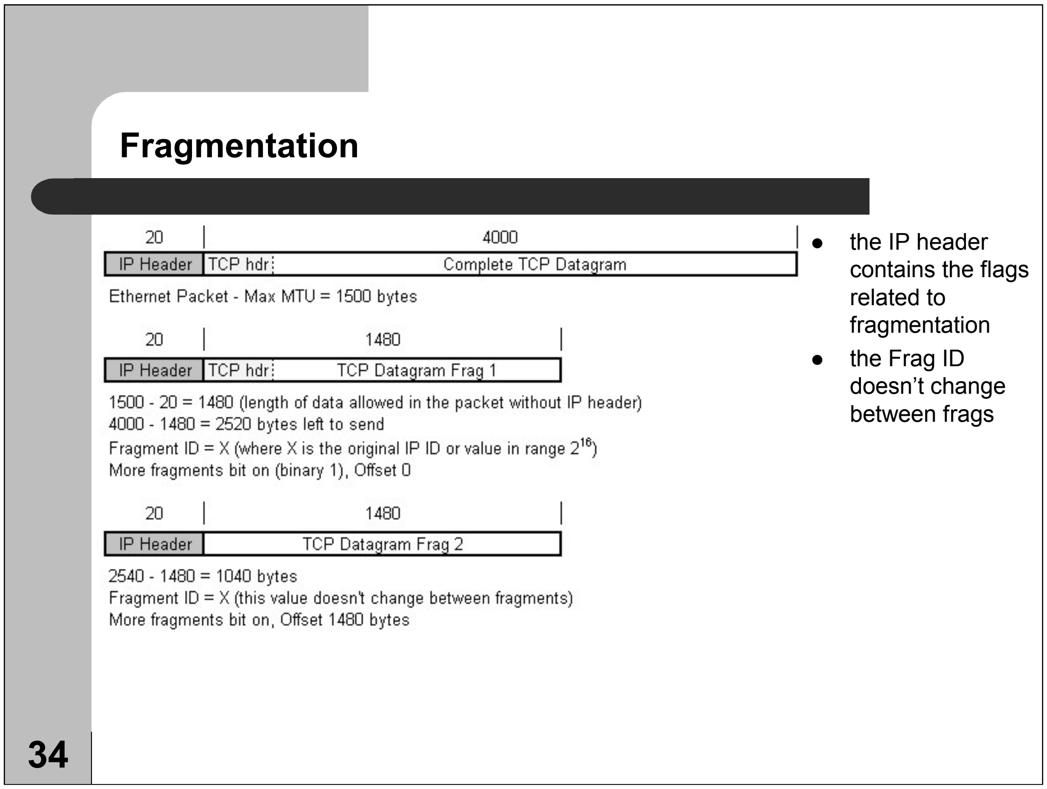
Ethernet Packet - Max MTU = 1500 bytes

1500 - 20 = 1480 (length of data allowed in the packet without IP header)
4000 - 1480 = 2520 bytes left to send
Fragment ID = X (where X is the original IP ID or value in range 2^{16})
More fragments bit on (binary 1), Offset 0

- the IP header contains the flags related to fragmentation

33

This space intentionally left blank.



Fragmentation

20 | 4000

IP Header | TCP hdr | Complete TCP Datagram

Ethernet Packet - Max MTU = 1500 bytes

20 | 1480

IP Header | TCP hdr | TCP Datagram Frag 1

1500 - 20 = 1480 (length of data allowed in the packet without IP header)
4000 - 1480 = 2520 bytes left to send
Fragment ID = X (where X is the original IP ID or value in range 2^{16})
More fragments bit on (binary 1), Offset 0

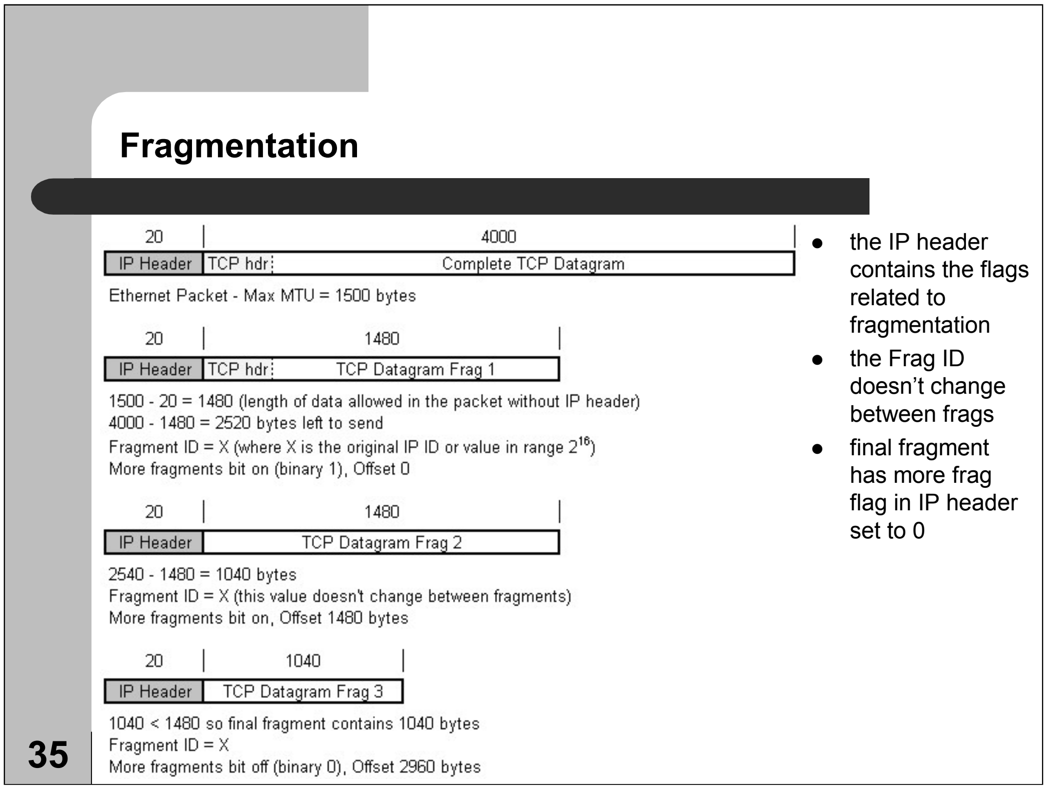
20 | 1480

IP Header | TCP Datagram Frag 2

2540 - 1480 = 1040 bytes
Fragment ID = X (this value doesn't change between fragments)
More fragments bit on, Offset 1480 bytes

- the IP header contains the flags related to fragmentation
- the Frag ID doesn't change between frags

This space intentionally left blank.



Fragmentation

20 | 4000

IP Header | TCP hdr | Complete TCP Datagram

Ethernet Packet - Max MTU = 1500 bytes

20 | 1480

IP Header | TCP hdr | TCP Datagram Frag 1

1500 - 20 = 1480 (length of data allowed in the packet without IP header)
4000 - 1480 = 2520 bytes left to send
Fragment ID = X (where X is the original IP ID or value in range 2^{16})
More fragments bit on (binary 1), Offset 0

20 | 1480

IP Header | TCP Datagram Frag 2

2540 - 1480 = 1040 bytes
Fragment ID = X (this value doesn't change between fragments)
More fragments bit on, Offset 1480 bytes

20 | 1040

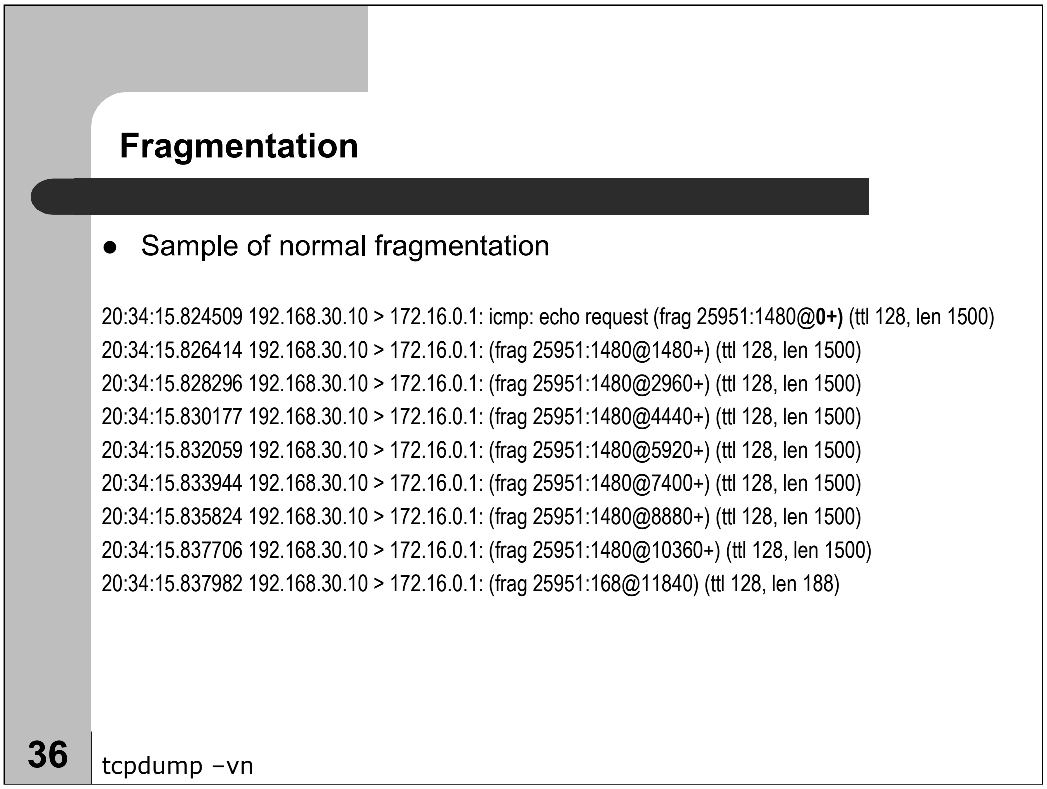
IP Header | TCP Datagram Frag 3

1040 < 1480 so final fragment contains 1040 bytes
Fragment ID = X
More fragments bit off (binary 0), Offset 2960 bytes

- the IP header contains the flags related to fragmentation
- the Frag ID doesn’t change between frags
- final fragment has more frag flag in IP header set to 0

35

This space intentionally left blank.



Fragmentation

- Sample of normal fragmentation

```
20:34:15.824509 192.168.30.10 > 172.16.0.1: icmp: echo request (frag 25951:1480@0+) (ttl 128, len 1500)
20:34:15.826414 192.168.30.10 > 172.16.0.1: (frag 25951:1480@1480+) (ttl 128, len 1500)
20:34:15.828296 192.168.30.10 > 172.16.0.1: (frag 25951:1480@2960+) (ttl 128, len 1500)
20:34:15.830177 192.168.30.10 > 172.16.0.1: (frag 25951:1480@4440+) (ttl 128, len 1500)
20:34:15.832059 192.168.30.10 > 172.16.0.1: (frag 25951:1480@5920+) (ttl 128, len 1500)
20:34:15.833944 192.168.30.10 > 172.16.0.1: (frag 25951:1480@7400+) (ttl 128, len 1500)
20:34:15.835824 192.168.30.10 > 172.16.0.1: (frag 25951:1480@8880+) (ttl 128, len 1500)
20:34:15.837706 192.168.30.10 > 172.16.0.1: (frag 25951:1480@10360+) (ttl 128, len 1500)
20:34:15.837982 192.168.30.10 > 172.16.0.1: (frag 25951:168@11840) (ttl 128, len 188)
```

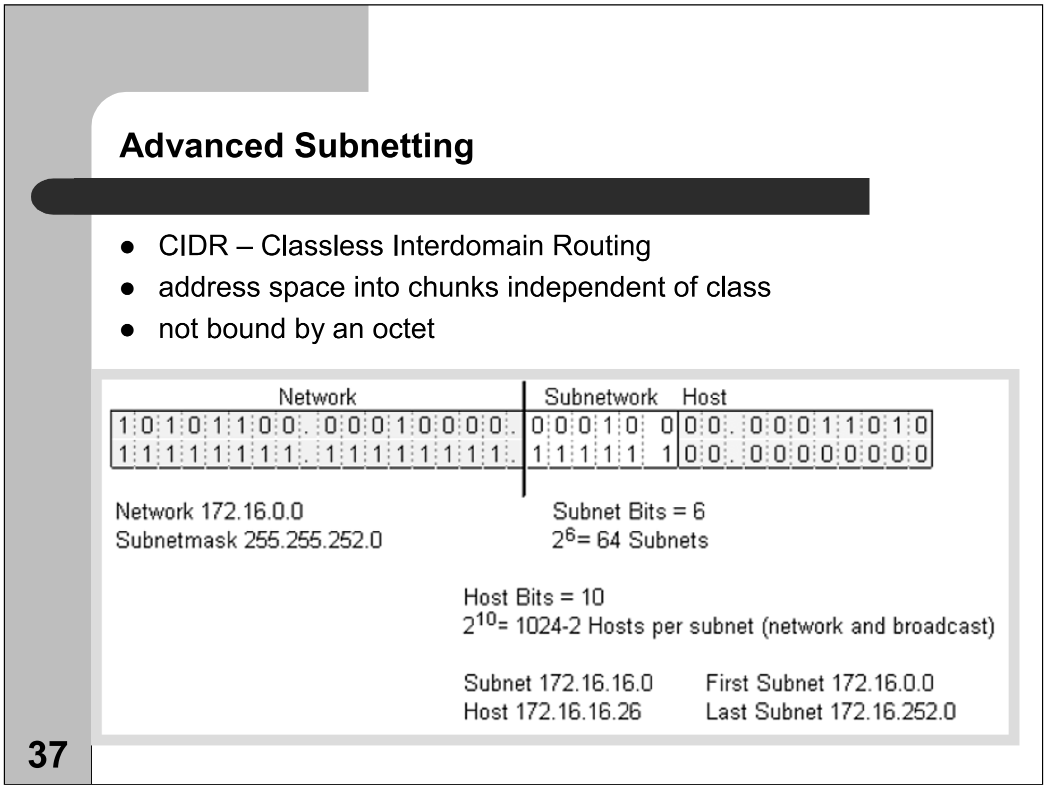
36 tcpdump –vn

Notice the IP ID is called the fragmentation ID and is propagated through each fragment belonging to the original packet. This is so the recipient can collect all the related packets for re-assembly. In the above example the frag ID is 25951.

We have seen 1480 before. This is the maximum size of a packet without a standard 20 byte IP header on an ethernet network. If the physical layer changes in size, it is highly probable that the size noted here will change too as IP adjusts the size of outgoing packets. Most students will see ethernet networks.

The initial fragment has the “0+” to indicate it is the first fragment and more will follow. The transport layer protocol header (layer 4) is noted in the first packet in normal fragmentation. It is possible to fragment this too but this is highly anomalous and should set off bells to flag the traffic for further analysis.

The last fragment will identify itself by having the no more fragments bit set. TCPDUMP displays this by removing the “+” from the fragment. Notice that the total size may be calculated by adding the fragment offset 11840 and the number of data bytes in the current fragment 168 together for a total of 12008 bytes. Therefore the original packet was 12008 bytes in length without the IP headers. Do not forget that the ICMP header was included with these fragments so if we take away the 8 byte ICMP header we are left with 12000 bytes. A rather large ICMP echo request but perfectly valid.



Advanced Subnetting

- CIDR – Classless Interdomain Routing
- address space into chunks independent of class
- not bound by an octet

```
                Network                 Subnetwork   Host
1 0 1 0 1 1 0 0 . 0 0 0 1 0 0 0 0 . | 0 0 0 1 0   0 | 0 0 . 0 0 0 1 1 0 1 0
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . | 1 1 1 1 1   1 | 0 0 . 0 0 0 0 0 0 0 0
```

Network 172.16.0.0
Subnetmask 255.255.252.0

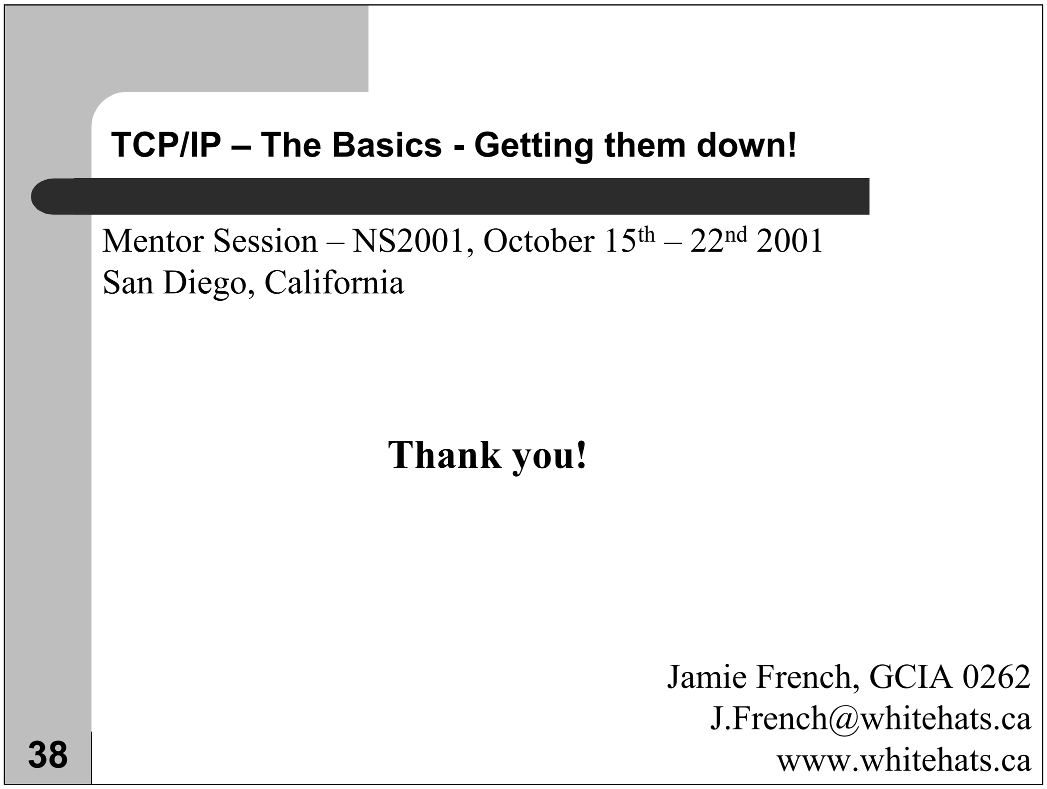
Subnet Bits = 6
2^6= 64 Subnets

Host Bits = 10
2^{10}= 1024-2 Hosts per subnet (network and broadcast)

Subnet 172.16.16.0
Host 172.16.16.26

First Subnet 172.16.0.0
Last Subnet 172.16.252.0

This is beyond the scope of this mentor session, but I thought it should be introduced.



TCP/IP – The Basics - Getting them down!

Mentor Session – NS2001, October 15th – 22nd 2001
San Diego, California

Thank you!

Jamie French, GCIA 0262
J.French@whitehats.ca
www.whitehats.ca

38

This space intentionally left blank.