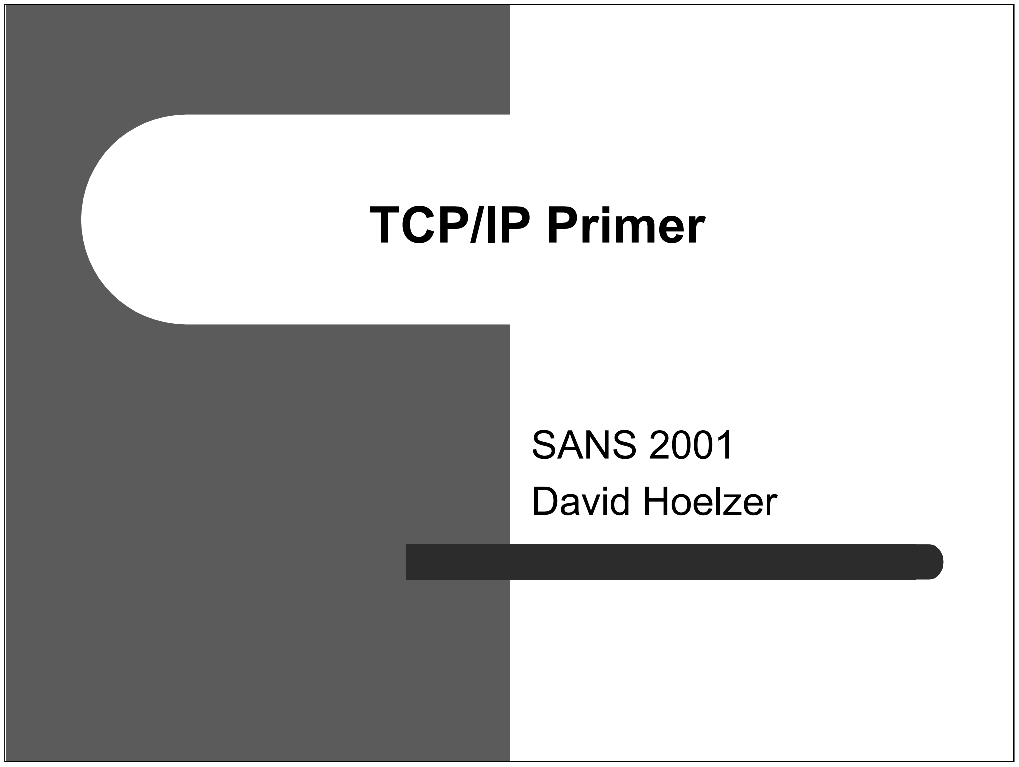
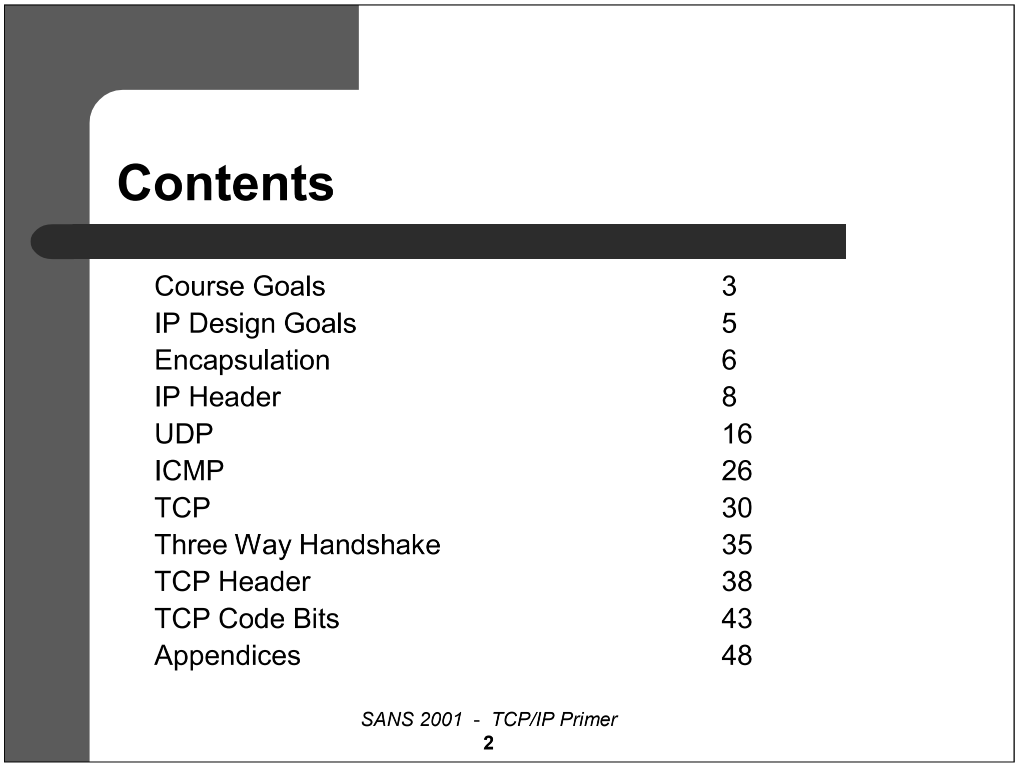


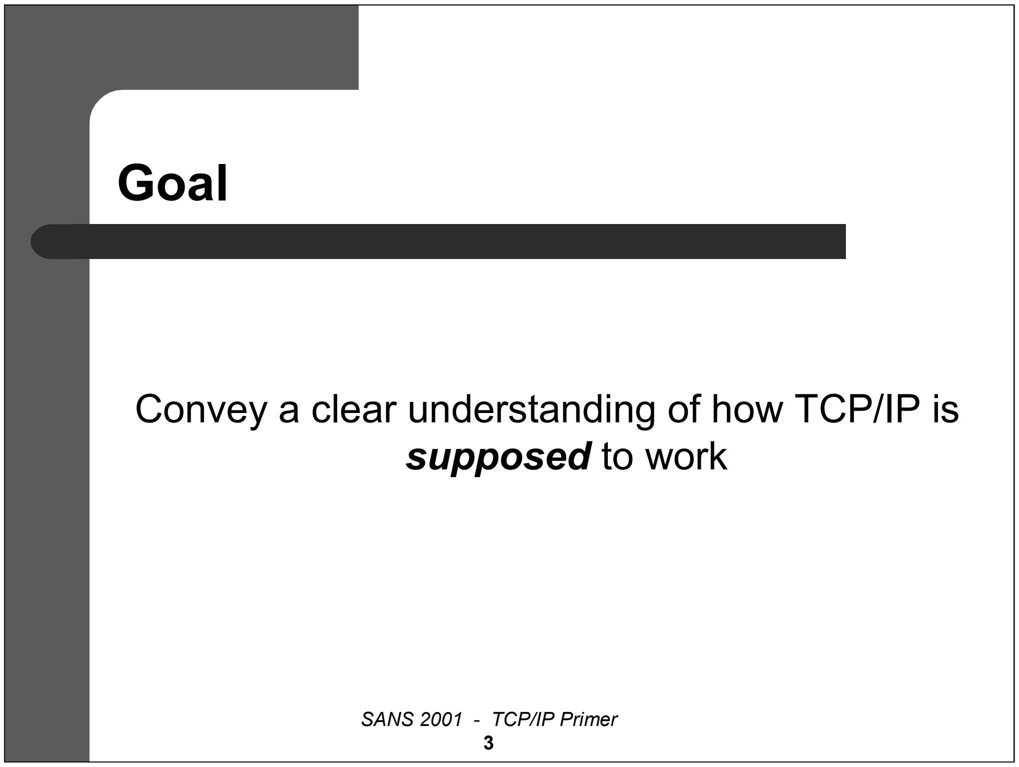
TCP/IP Primer

SANS 2001

David Hoelzer



Contents



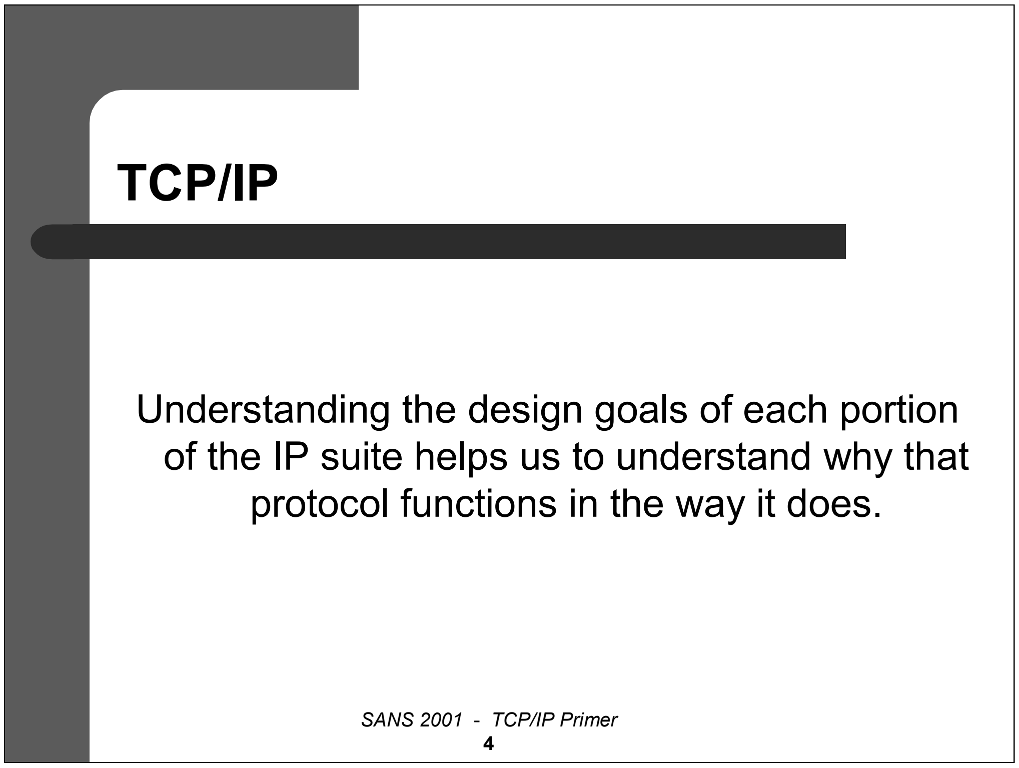
Goal

Convey a clear understanding of how TCP/IP is ***supposed*** to work

SANS 2001 - TCP/IP Primer
3

This short course is designed for individuals in the various tech tracks, particularly the ID track, who realize that they need a hands on refresher of TCP/IP fundamentals. Much of the information and methods taught in the ID track involve the detection and analysis of anomalies. In order to gain the greatest benefit from this information, it is a prerequisite that we understand how the basic protocols in the TCP/IP suite are supposed to function. If we don't understand this, how can we possibly identify anomalies?

The last few pages of this presentation are a series of short quizzes that will be covered during the course of the class. If you are wondering if this is a class that you ought to sit through, why not take a look at the quizzes now and test yourself?



TCP/IP

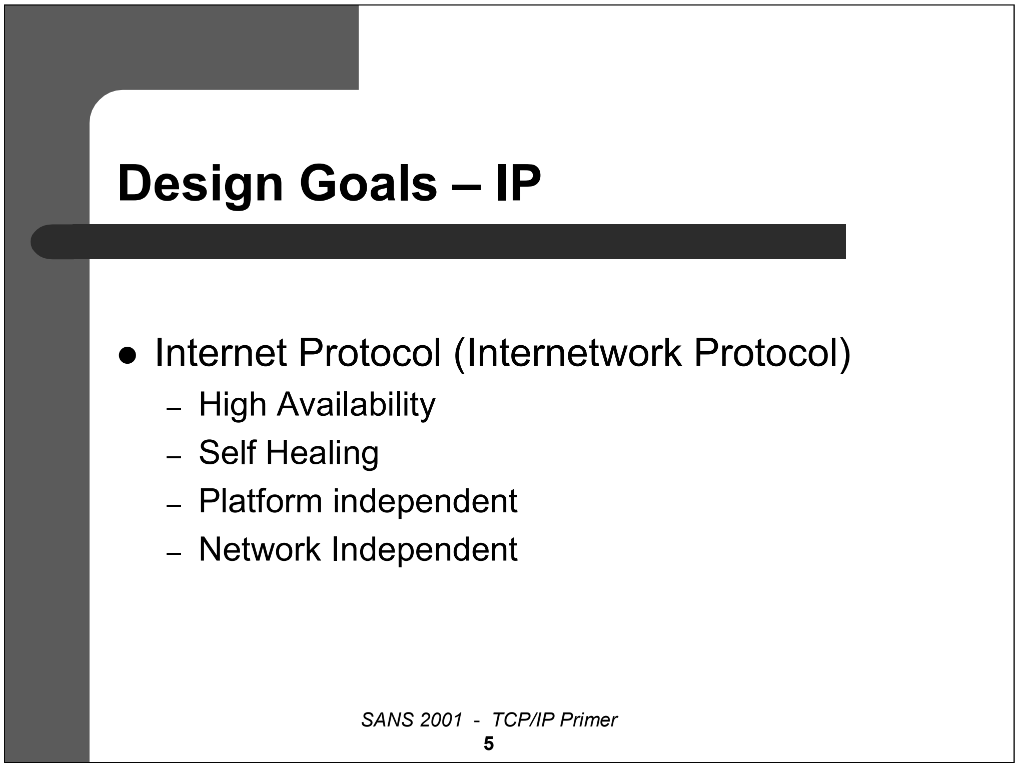
Understanding the design goals of each portion of the IP suite helps us to understand why that protocol functions in the way it does.

SANS 2001 - TCP/IP Primer
4

During my years of teaching, I have found that the most effective manner of teaching that has the highest retention rate and gives the students the greatest ability to apply the information is to help the students to build a foundation of understanding. Here's what this means:

Imagine that you go to work in an automotive garage. You've tinkered with cars before, but you've really had no formal training. The head mechanic gives you some troubleshooting "flow charts" from Chevrolet and asks you to go to work troubleshooting fuel injection systems. Following the troubleshooting charts and instructions, chances are that you can fix many many problems. But what do you do when the troubleshooter fails to identify the problem? What happens when someone brings you a Chrysler?

On the other hand, what if you were given an in-depth course regarding the mechanics of fuel injection? Would you be in a better position to troubleshoot new car models?



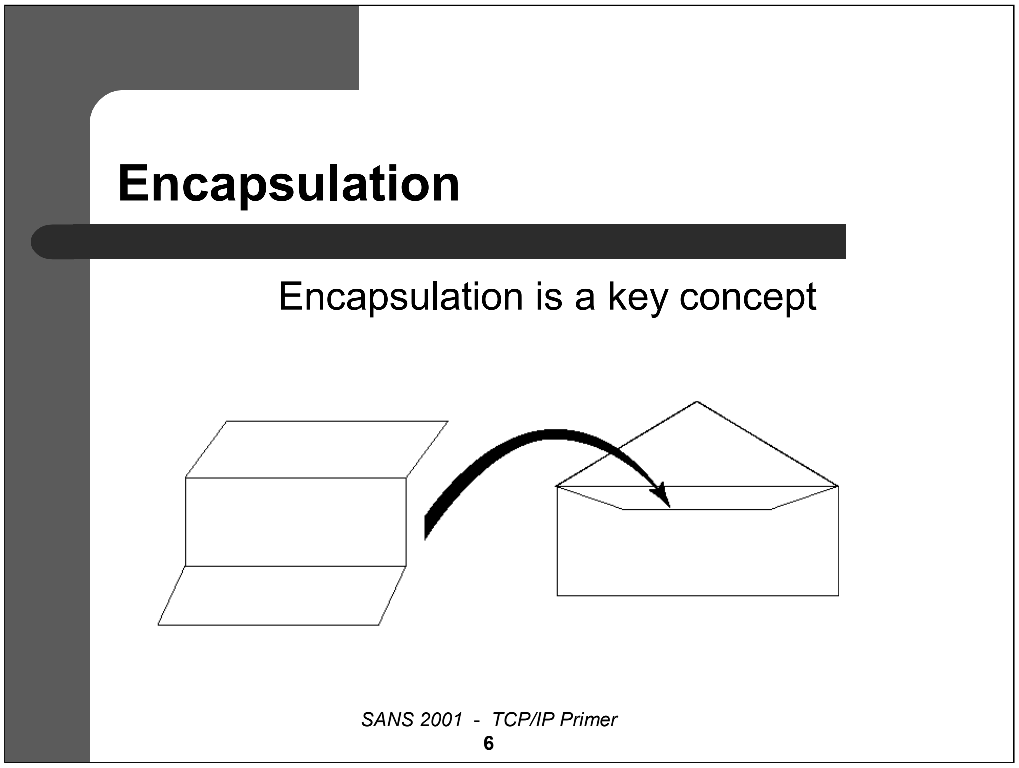
Design Goals – IP

- Internet Protocol (Internetwork Protocol)
  - High Availability
  - Self Healing
  - Platform independent
  - Network Independent

SANS 2001 - TCP/IP Primer
5

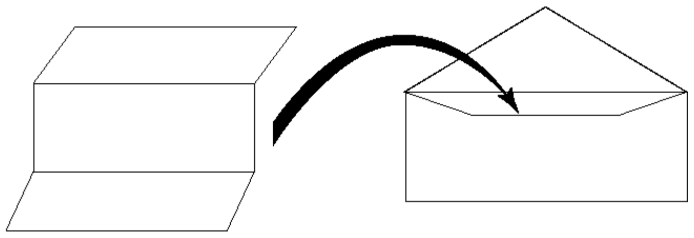
Prior to the advent of IP, the world was made up of many different disparate network architectures. The designers of IP had some very critical design goals in mind. First of all, there was a strong requirement to create a 'self healing' network. This protocol was being designed for use on DOD ARPA networks, in fact, it was being designed to unite the various networks. By making a protocol suite that would be self healing, the network would be smart enough to reroute traffic in the event of a catastrophic failure (missile strike?).

An additional important requirement is that it be platform independent. Rather than starting with particular pieces of hardware and determining how to get them to talk to one another using a particular protocol, the protocol was being designed first, later to be implemented on individual platforms. This goes hand in hand with the final point. Rather than designing a proprietary set of protocols to address individual needs, this protocol was being designed with the explicit goal of connecting various disparate networks. Thus it becomes unimportant whether the machines at opposite ends of an IP network are running over Ethernet or Token ring or FDDI or any other low level networking architecture.



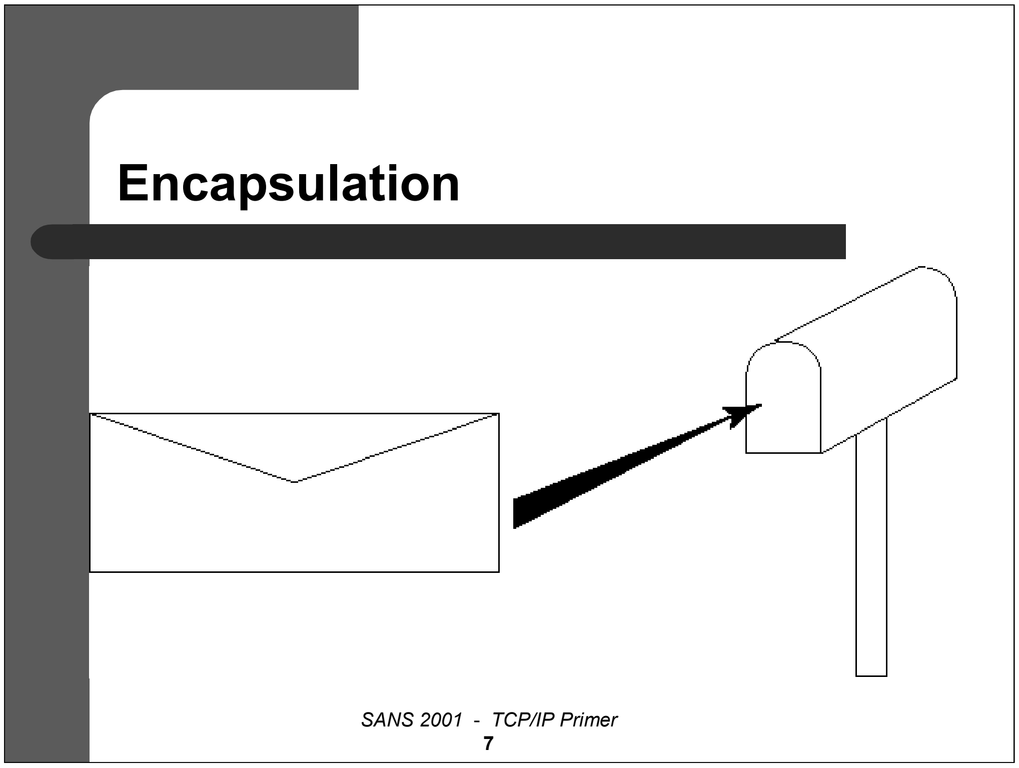
Encapsulation

Encapsulation is a key concept

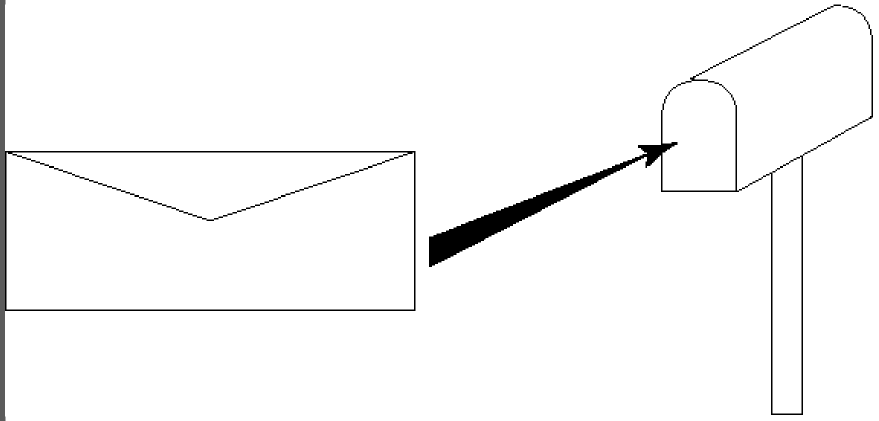


SANS 2001 - TCP/IP Primer
6

Encapsulation is what Networking is all about. Essentially, it means putting one thing inside of another. In the world of networking this is done to allow a particular piece of information to be delivered somewhere. A good analogy is the Postal system. You have a piece of data, a letter, that you want to send to an individual in California. You can't just drop the letter into your mail box because the mail man doesn't know how to handle it. Instead, you "encapsulate" it into an envelope, and on that envelope you write an address. Now, as an 'end user' you really don't care what happens in between, but you might choose to use various services that the post office offers with regard to the delivery of that message. Are you sending a post card? (Send and Pray) Are you sending a legal document that perhaps needs to be sent Certified or Registered, thus guaranteeing delivery?



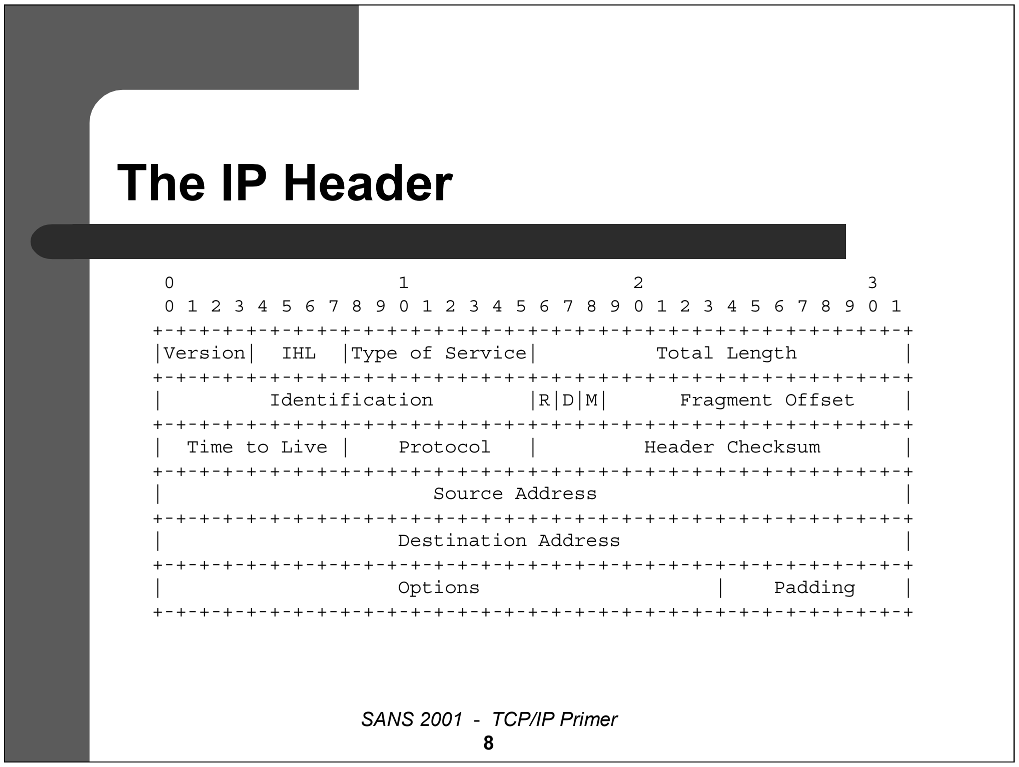
Encapsulation



SANS 2001 - TCP/IP Primer
7

Once you've encapsulate your data, your letter, you put it into your mail box. In our illustration, this is essentially the same as dropping the packet out onto the network or perhaps handing it to a router. Depending on the scope you're looking at, though, you may be interested in what happens next. In the mail system, that letter is next encapsulated into the mail man's bag, then into his mail truck, then into a post office and finally into a sorting center. If we were to pick up this process in California at the local post office, the letter would go from the truck, to the bag, to the mail box and finally to the recipient, who knows how to open the letter and can read and understand the contents.

With this in mind, we're more or less ready to dive into IP to get a first hand look at how the data is encoded and encapsulated.

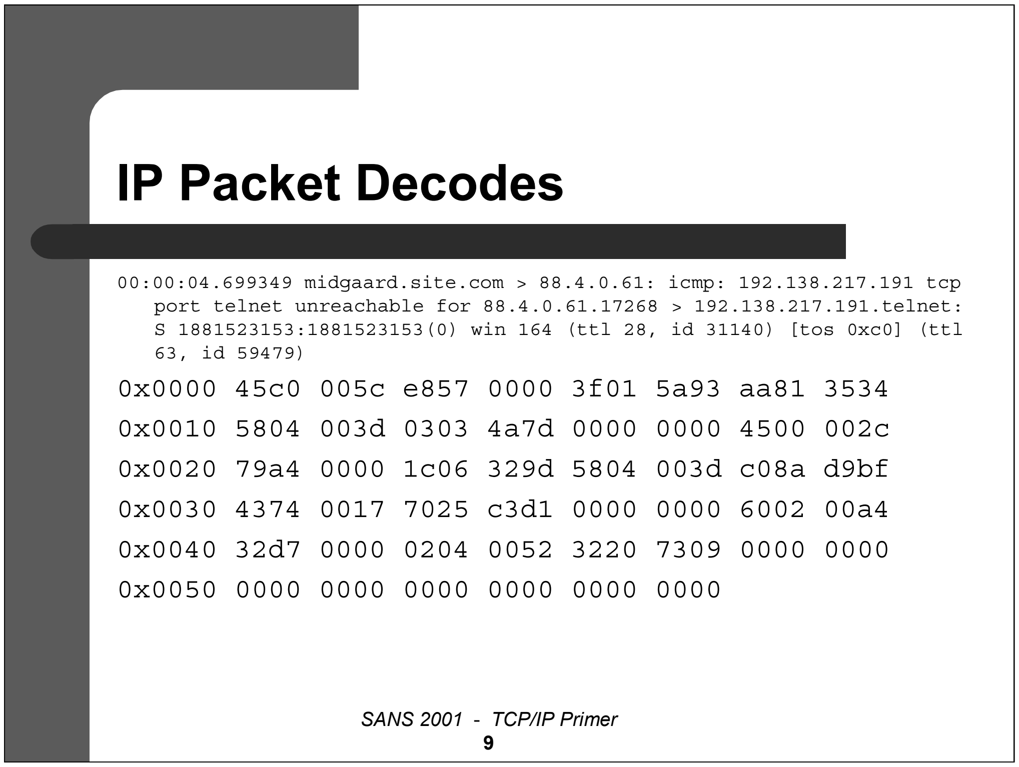


The IP Header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |R|D|M|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SANS 2001 - TCP/IP Primer
8

This is the definition of the IP header from RFC 791. Across the top I have listed Bit Positions from zero through thirty one, which gives us 32 bits in total for each line. This helps us to see how many bits or bytes are devoted to each field in the IP header. In an IP header, the only portion that is 'optional' is, appropriately, the 'options'. Other than that, the position of all pieces of information remains static. We'll look at a few packet dumps in the next few slides and discuss the meanings of some of the fields.

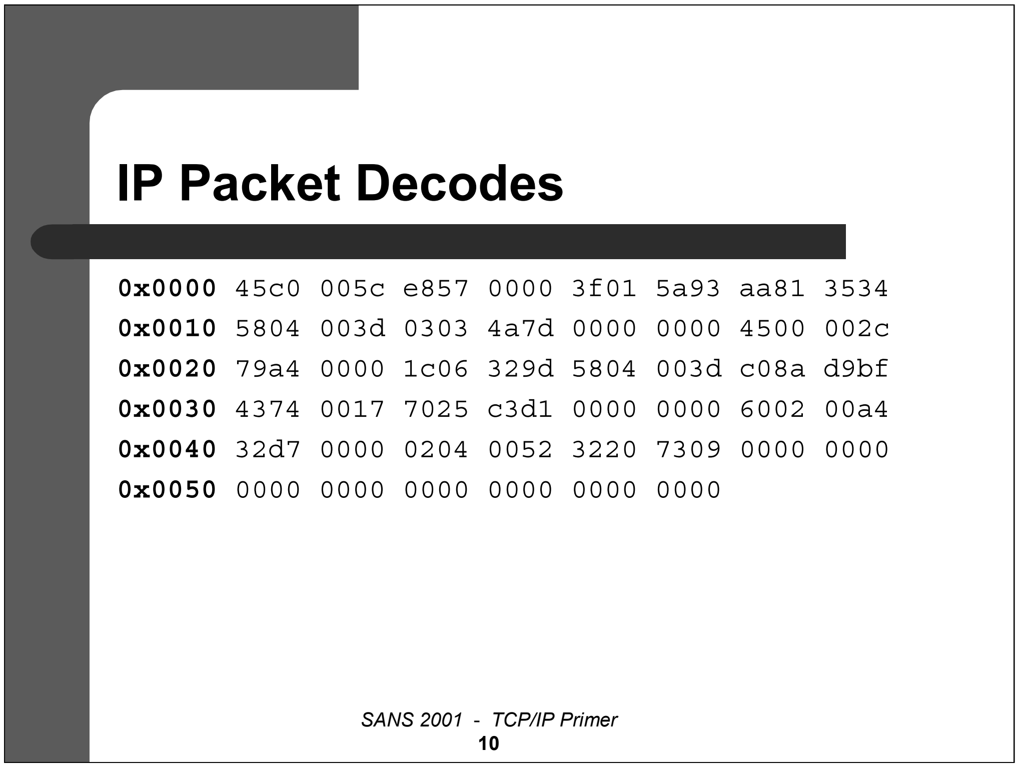


IP Packet Decodes

```
00:00:04.699349 midgaard.site.com > 88.4.0.61: icmp: 192.138.217.191 tcp
   port telnet unreachable for 88.4.0.61.17268 > 192.138.217.191.telnet:
   S 1881523153:1881523153(0) win 164 (ttl 28, id 31140) [tos 0xc0] (ttl
   63, id 59479)
0x0000 45c0 005c e857 0000 3f01 5a93 aa81 3534
0x0010 5804 003d 0303 4a7d 0000 0000 4500 002c
0x0020 79a4 0000 1c06 329d 5804 003d c08a d9bf
0x0030 4374 0017 7025 c3d1 0000 0000 6002 00a4
0x0040 32d7 0000 0204 0052 3220 7309 0000 0000
0x0050 0000 0000 0000 0000 0000 0000
```

SANS 2001 - TCP/IP Primer
9

This slide depicts a single packet dumped from TCPDump. The first 4 lines, beginning with the time stamp, show how TCPDump has decoded the information found within the hex dump that follows. We, however, are interested in being able to take that header apart ourselves. Over the next few slides we will focus on the hex portion of this dump and attempt to identify each of the various portions of the IP header

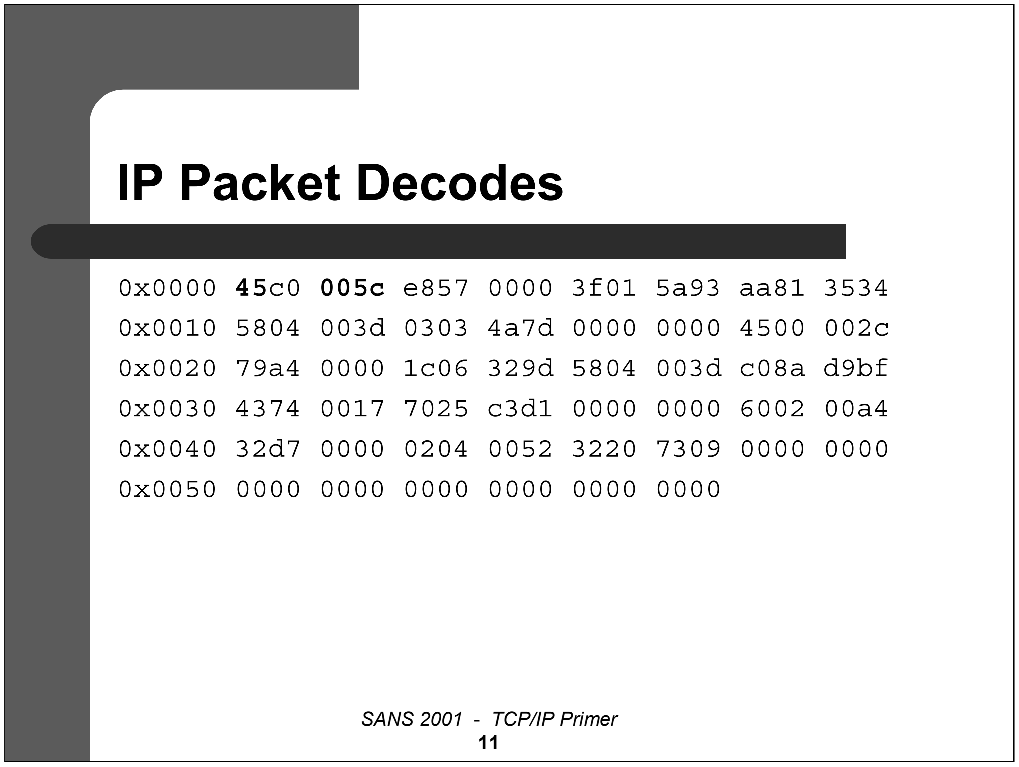


IP Packet Decodes

```
0x0000 45c0 005c e857 0000 3f01 5a93 aa81 3534
0x0010 5804 003d 0303 4a7d 0000 0000 4500 002c
0x0020 79a4 0000 1c06 329d 5804 003d c08a d9bf
0x0030 4374 0017 7025 c3d1 0000 0000 6002 00a4
0x0040 32d7 0000 0204 0052 3220 7309 0000 0000
0x0050 0000 0000 0000 0000 0000 0000
```

SANS 2001 - TCP/IP Primer
10

The bold numbers to the right (0x0000, 0x0010, etc.) show us the starting byte position of the first byte in that line. In other words, line one starts with byte zero, line two with byte 16, line 3 with byte 32, line 4 with byte 48, etc. These are really just to help you find a byte more quickly without having to count from zero every time.



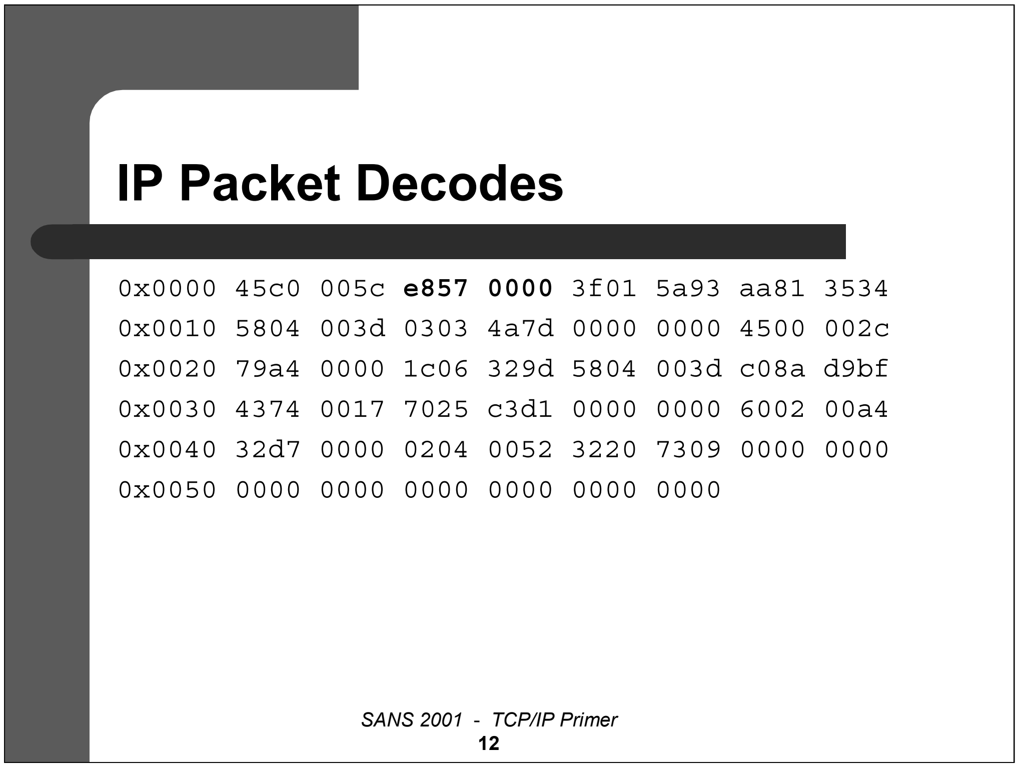
IP Packet Decodes

```
0x0000 45c0 005c e857 0000 3f01 5a93 aa81 3534
0x0010 5804 003d 0303 4a7d 0000 0000 4500 002c
0x0020 79a4 0000 1c06 329d 5804 003d c08a d9bf
0x0030 4374 0017 7025 c3d1 0000 0000 6002 00a4
0x0040 32d7 0000 0204 0052 3220 7309 0000 0000
0x0050 0000 0000 0000 0000 0000 0000
```

SANS 2001 - TCP/IP Primer
11

The first four bytes take in the first line of our header definition. We have the first half of byte zero equal to “4” which defines this as an IP Version 4 packet. The second half is set to “5” which defines our header length. The IP header was designed with a desire to conserve space. Rather than send a stream of 16 integers, the fields are chopped up using the minimum number of bits required to encode the necessary information. This means that certain fields need to be ‘interpreted’ rather than used as they are. Looking at the header definition, what is the smallest size for an IP header? 20 bytes. In fact this number, 5 in this case, defines the number of “double words” in the IP header. In simple terms, it means that the header is the value of these four bits times 4 (5 X 4 = 20). This field is used to determine whether IP Options have been set in this packet and to mark off where the data payload begins.

The second bold field, bytes two and three, are set to a value of “0x005c”. Converting this to decimal, we get 92. So the total length of this entire IP packet including the header is 92 bytes. This means that, minus the IP header, we have 72 bytes of data.



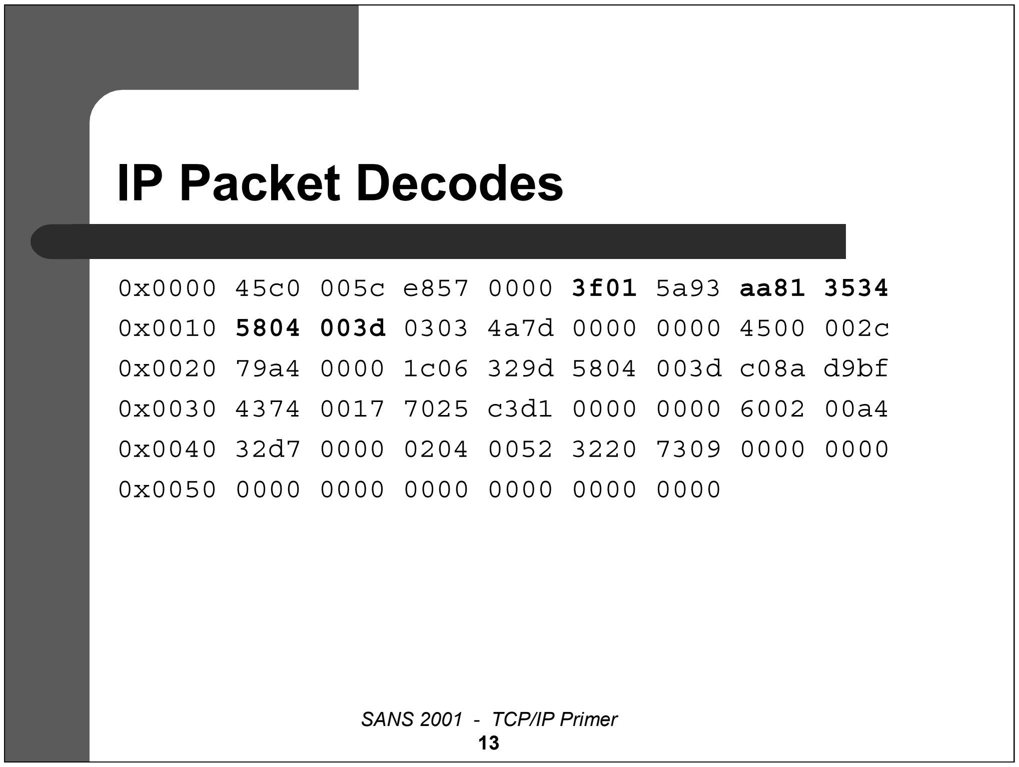
IP Packet Decodes

```
0x0000 45c0 005c e857 0000 3f01 5a93 aa81 3534
0x0010 5804 003d 0303 4a7d 0000 0000 4500 002c
0x0020 79a4 0000 1c06 329d 5804 003d c08a d9bf
0x0030 4374 0017 7025 c3d1 0000 0000 6002 00a4
0x0040 32d7 0000 0204 0052 3220 7309 0000 0000
0x0050 0000 0000 0000 0000 0000 0000
```

SANS 2001 - TCP/IP Primer
12

The next field in the header is the two byte IP Identification number. This number is not particularly important most of the time, although it can be helpful in identifying certain anomalous signatures. Effectively, this is a random number, although it is implemented in various ways in various IP stacks. Most commonly it is simply an incremented value.

The IP ID number becomes very important when used in conjunction with the next set of fields. The 4 fields that follow in the IP header take up only 2 bytes. They are used to control Fragmentation.



IP Packet Decodes

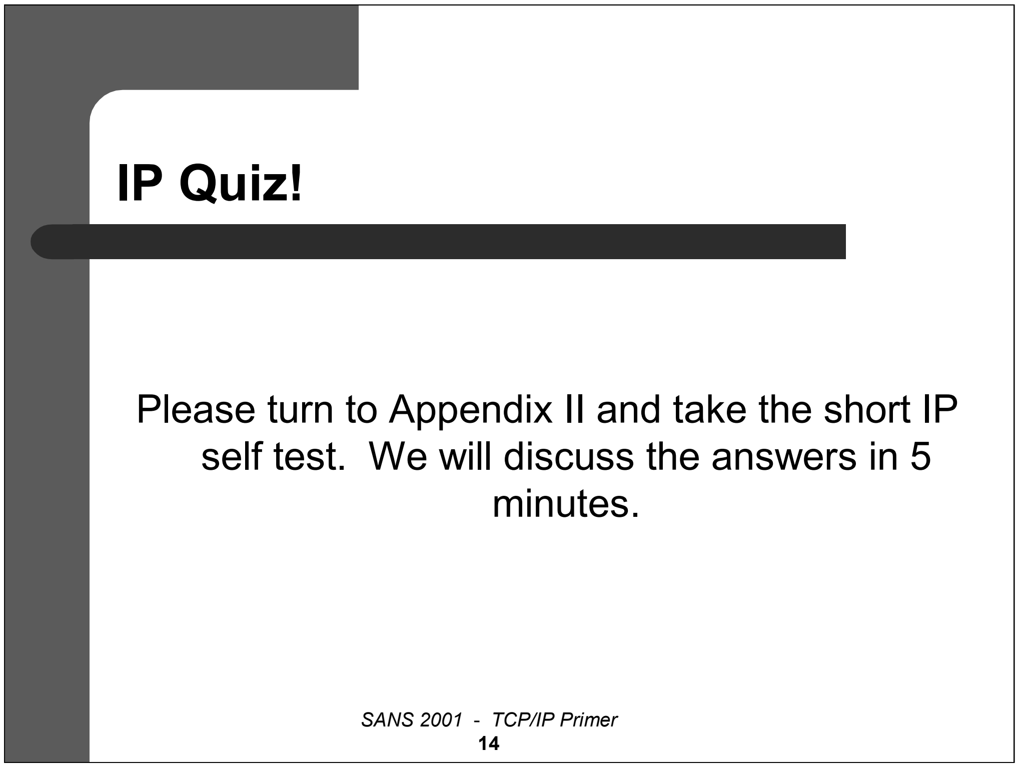
```
0x0000 45c0 005c e857 0000 3f01 5a93 aa81 3534
0x0010 5804 003d 0303 4a7d 0000 0000 4500 002c
0x0020 79a4 0000 1c06 329d 5804 003d c08a d9bf
0x0030 4374 0017 7025 c3d1 0000 0000 6002 00a4
0x0040 32d7 0000 0204 0052 3220 7309 0000 0000
0x0050 0000 0000 0000 0000 0000 0000
```

SANS 2001 - TCP/IP Primer
13

The next two fields are very important for us as well. First, we have a 1 byte "TTL" field, or time to live. Every time this packet passes through a network device, the value in this field is decremented by 1 (or a value set in the network hardware). Any network device that receives a packet with a TTL value of one will drop the packet and generate an error message directed to the originating host. This behavior is to prevent 'lost' packets from bouncing around the internet indefinitely.

The byte immediately following the TTL is the Protocol field. This field defines what sort of IP packet we are looking at. A complete list (more or less) of IP Protocol Numbers can be found in Appendix I of this presentation. There are three protocol numbers that you must know, however. 1, 6 and 17 (0x01, 0x06 and 0x11). These types are ICMP, TCP and UDP. This means that in this case, we're looking at an IP packet that encapsulates an ICMP message.

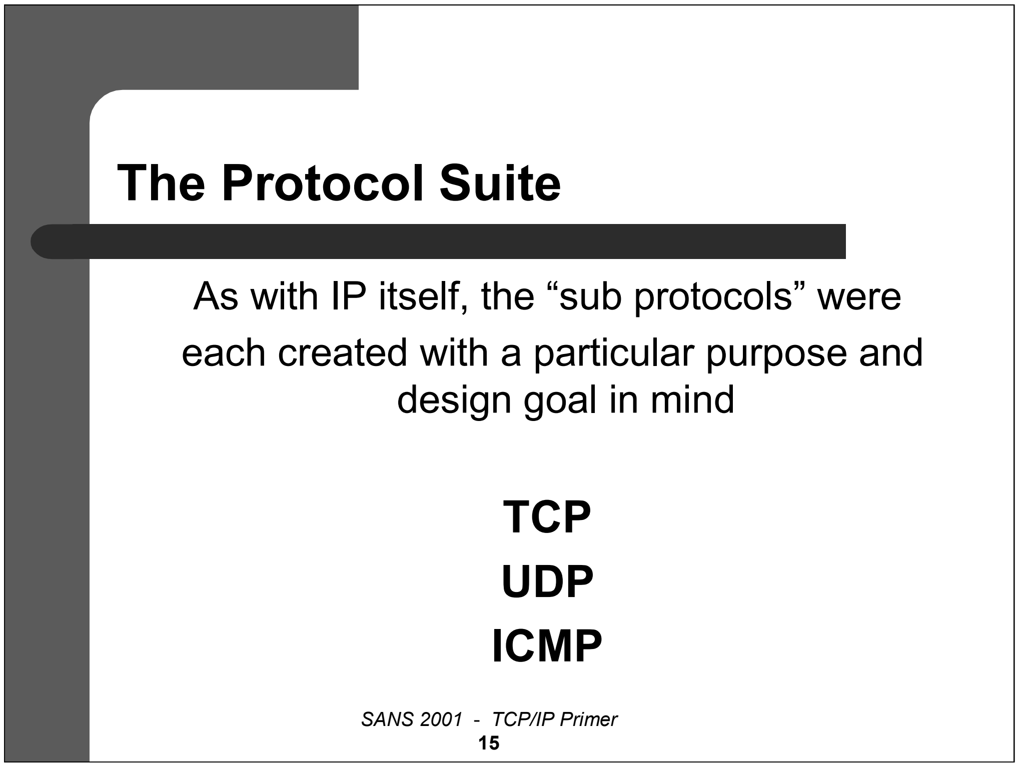
The source and destination IP addresses are also contained within the IP header. It's good to mention here that everything in a packet header will always be encoded in "Network Byte Order". Essentially this means that no matter how a machine encodes numbers internally, the data in the packet header will be encoded from left to right. Hence, the value aa813534 yields 170.129.53.52.



IP Quiz!

Please turn to Appendix II and take the short IP self test. We will discuss the answers in 5 minutes.

Please see appendix II for the quiz.



The Protocol Suite

As with IP itself, the "sub protocols" were each created with a particular purpose and design goal in mind

TCP

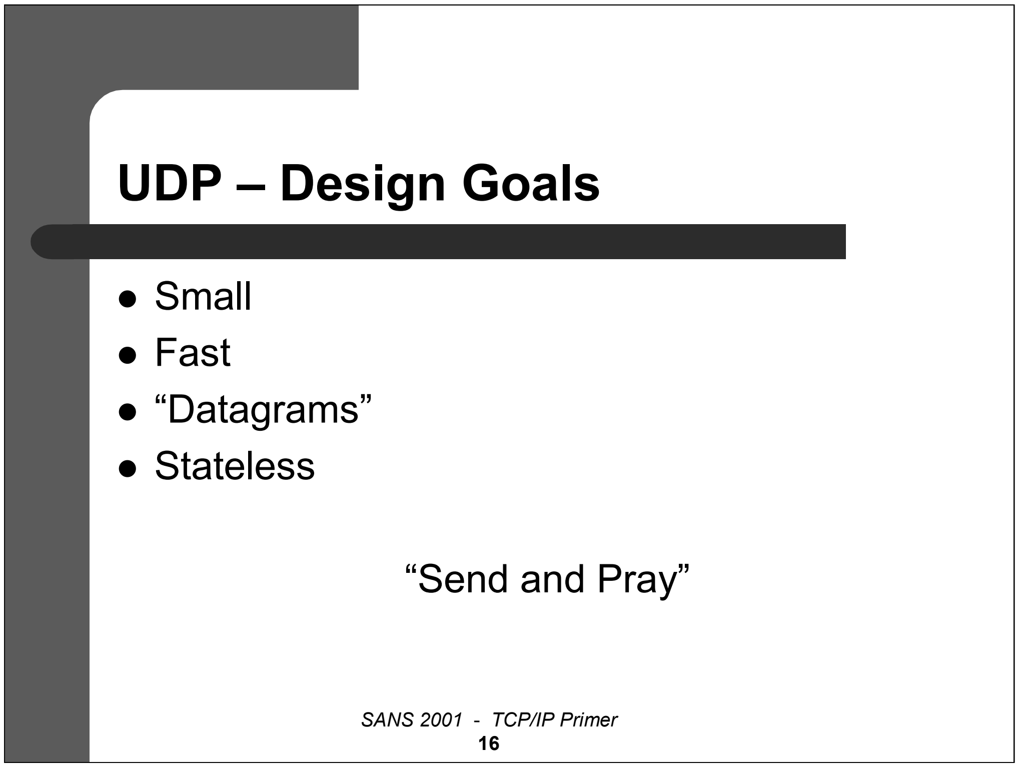
UDP

ICMP

SANS 2001 - TCP/IP Primer

15

There are actually many different parts to the IP suite of protocols. In fact, it can support up to 256 different sub protocols. However, these are the three on which we will focus our attention since they are by far the ones most commonly used in network based security and intrusion detection. We will take each in turn.



UDP – Design Goals

- Small
- Fast
- “Datagrams”
- Stateless

“Send and Pray”

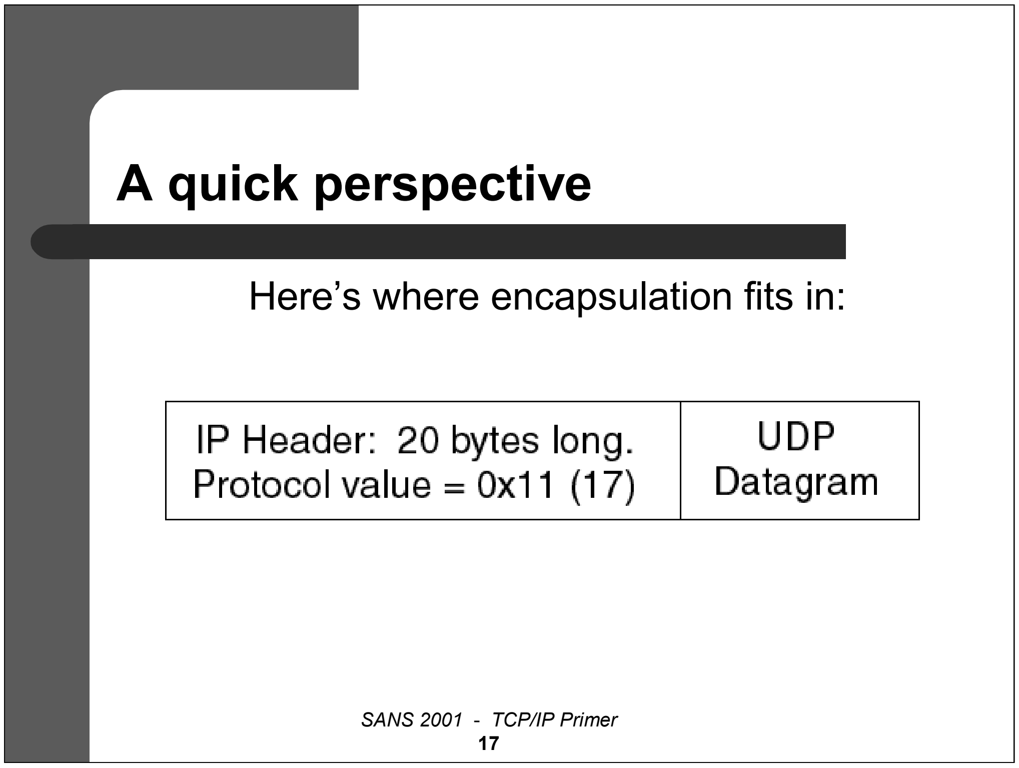
SANS 2001 - TCP/IP Primer
16

The UDP protocol (User Datagram Protocol) was included in the IP suite in order to allow for very fast and simple network communication. UDP a light weight protocol because there is no additional network “overhead” involved, which is something that we will examine with TCP. This same lack of overhead gives UDP a slight speed advantage over TCP since there is less involved in both building a connection and sending the data.

The “Datagram” portion of the name should tell you something else about this. Think of a “Telegram”. Would you expect to get a 20 page telegram? UDP Datagrams are intended for transporting small chunks of information. Sending a single answer to a single query, for instance. DNS is a good example of this.

“Stateless” means that there is no “connection” involved. One machine sends a packet to another and forgets about it, it doesn’t wait around for an answer or something to tell it that the packet arrived, hence “Send and Pray”.

It is entirely possible to implement “State” using UDP, however the state must be maintained by a user space program, for instance, NFS.



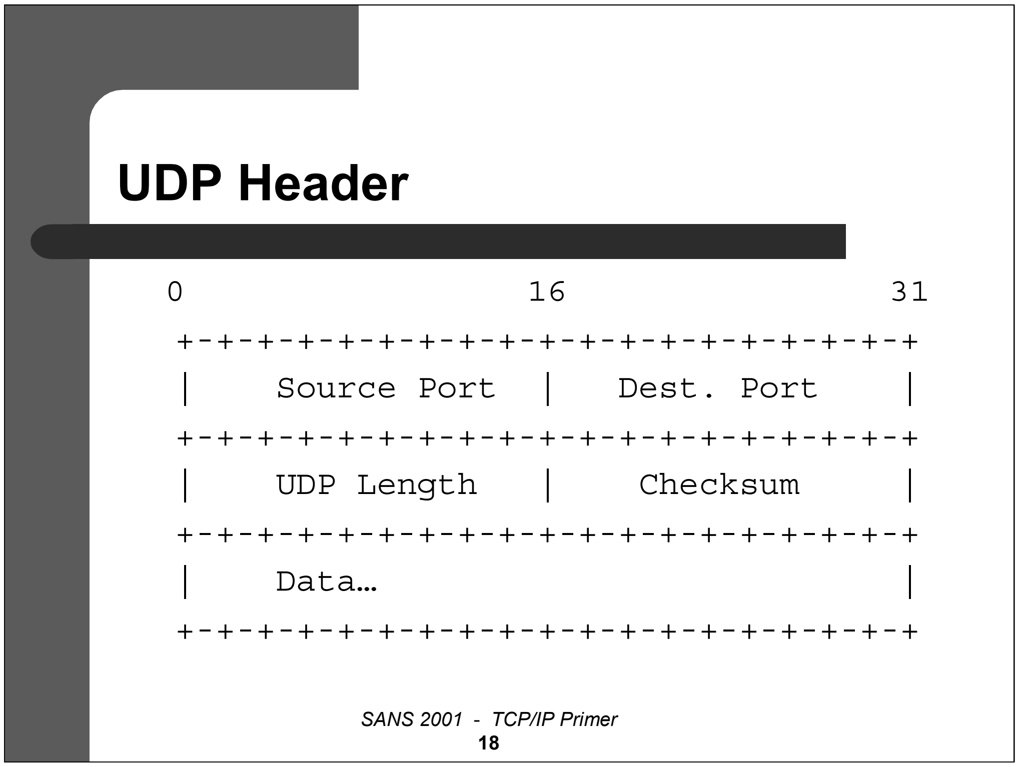
A quick perspective

Here's where encapsulation fits in:

IP Header: 20 bytes long. Protocol value = 0x11 (17)	UDP Datagram

SANS 2001 - TCP/IP Primer
17

Here's where that whole illustration about encapsulation begins to be applied. There really isn't much you can do with IP without the sub protocols that are used to actually transport data. It would be like having mail trucks and shipping lines without having anything to deliver. Once we've defined where the packet is coming from and going to, the sub protocols step up to actually encode the data that we're sending. The sub protocols are more or less like the letter inside of the envelope, while the IP header (which contains the addresses) is the envelope itself.

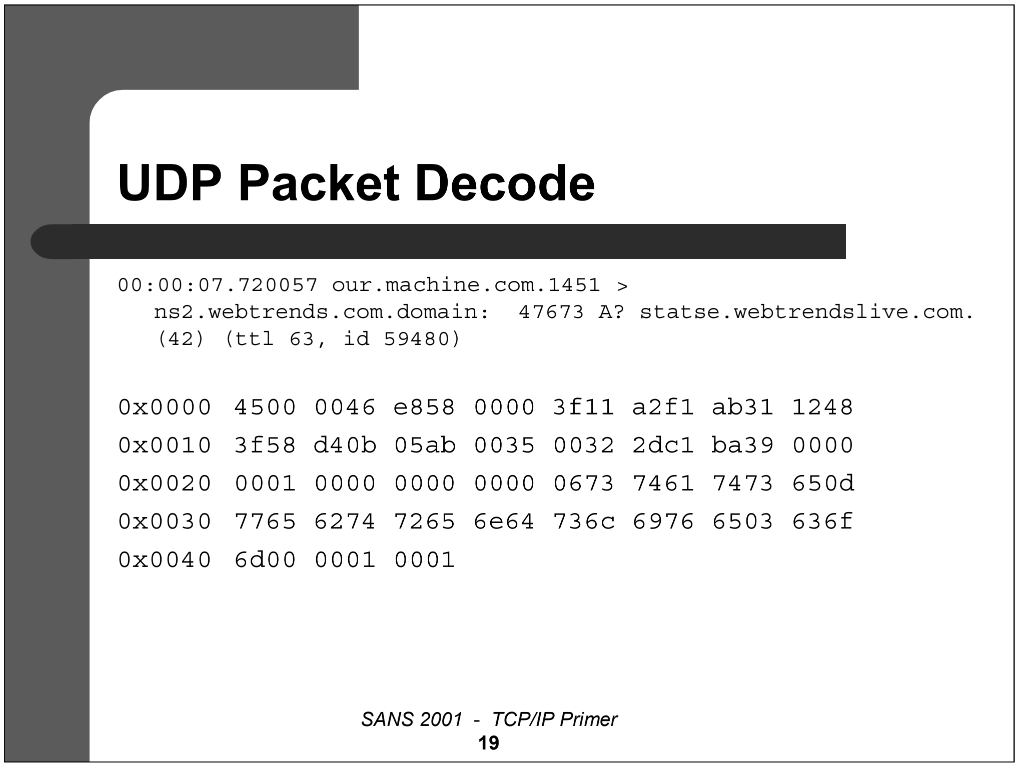


UDP Header

```
0                 16                 31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Source Port  |   Dest. Port     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    UDP Length   |    Checksum      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Data…                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SANS 2001 - TCP/IP Primer
18

We'll start with UDP since it has one of the simplest header formats. In fact, we can see that it is far simpler than the IP header. One particularly nice feature of the UDP header for an analyst is the fact that it is a fixed length. The header can never be less than 8 bytes long. Additionally, each field in the header is two bytes wide. We shall see when we compare this to TCP that it makes analysis very easy.



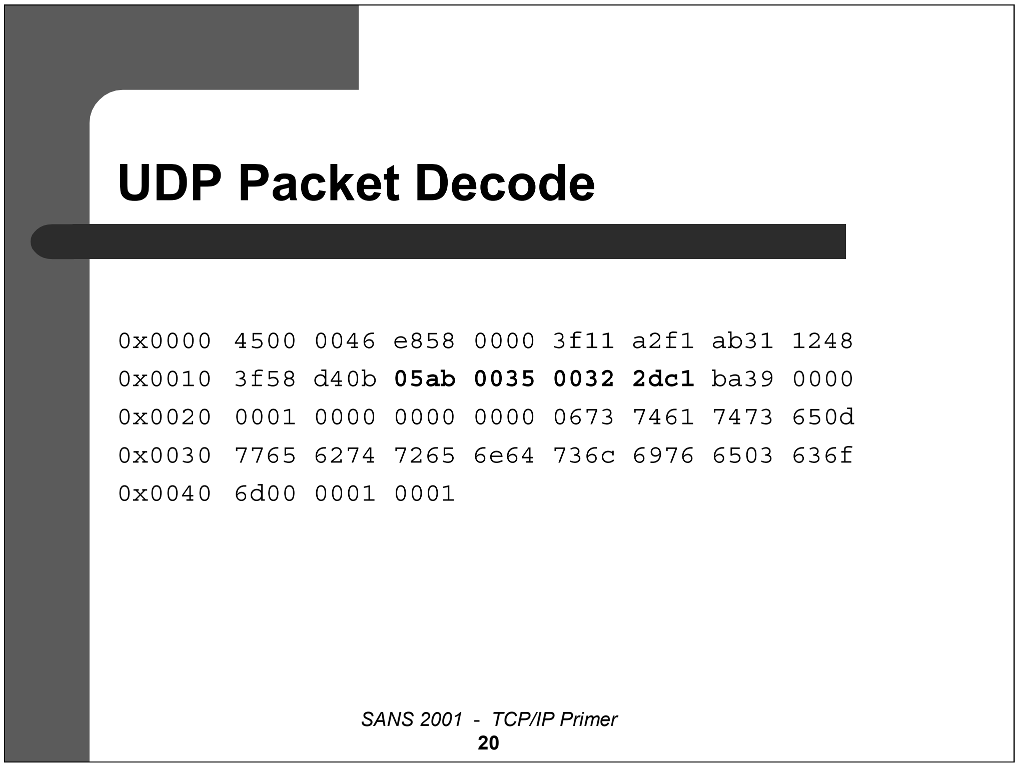
UDP Packet Decode

```
00:00:07.720057 our.machine.com.1451 >
   ns2.webtrends.com.domain:  47673 A? statse.webtrendslive.com.
   (42) (ttl 63, id 59480)

0x0000  4500 0046 e858 0000 3f11 a2f1 ab31 1248
0x0010  3f58 d40b 05ab 0035 0032 2dc1 ba39 0000
0x0020  0001 0000 0000 0000 0673 7461 7473 650d
0x0030  7765 6274 7265 6e64 736c 6976 6503 636f
0x0040  6d00 0001 0001
```

SANS 2001 - TCP/IP Primer
19

As we did with the IP header, we will now take apart the UDP header one field at a time. We'll use the packet above to accomplish this. It ought to look familiar. It was the one you just took apart the IP header for in your quiz!

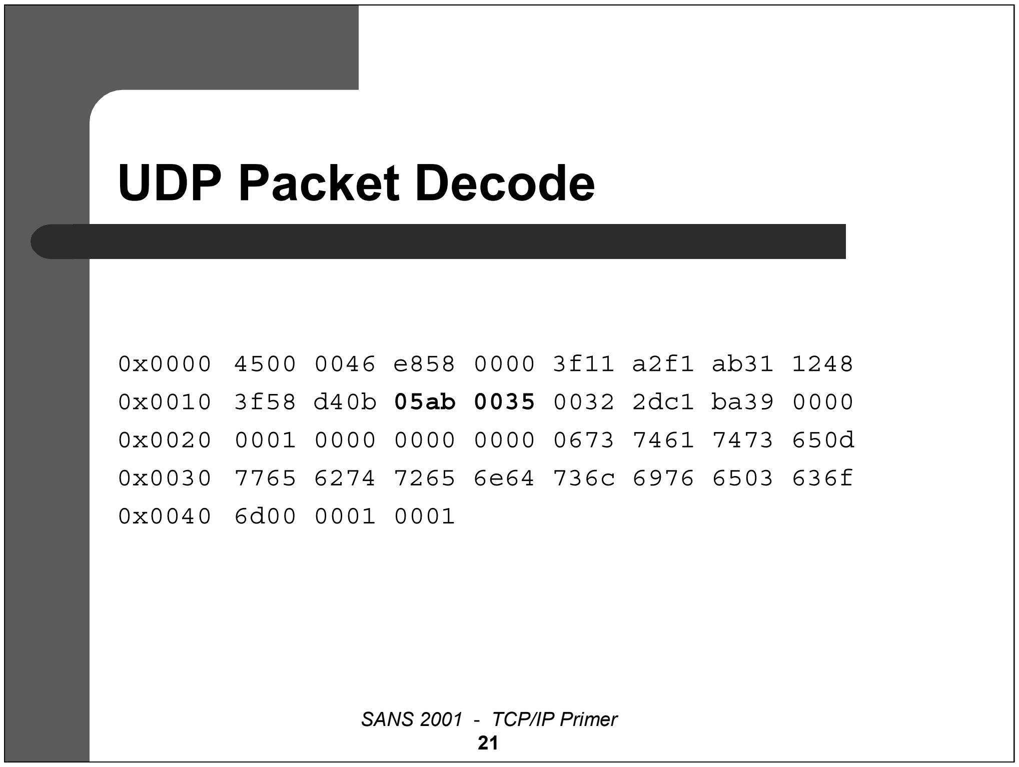


UDP Packet Decode

```
0x0000  4500 0046 e858 0000 3f11 a2f1 ab31 1248
0x0010  3f58 d40b 05ab 0035 0032 2dc1 ba39 0000
0x0020  0001 0000 0000 0000 0673 7461 7473 650d
0x0030  7765 6274 7265 6e64 736c 6976 6503 636f
0x0040  6d00 0001 0001
```

SANS 2001 - TCP/IP Primer
20

First off all, let's just highlight where the UDP header is in this packet. If you're not already, you will quickly become adept at finding the various headers quickly. Looking at the IP Version and Header size field, you see 0x45. Whenever you have an IP header length of 20, the sub protocol header will begin where this one does.



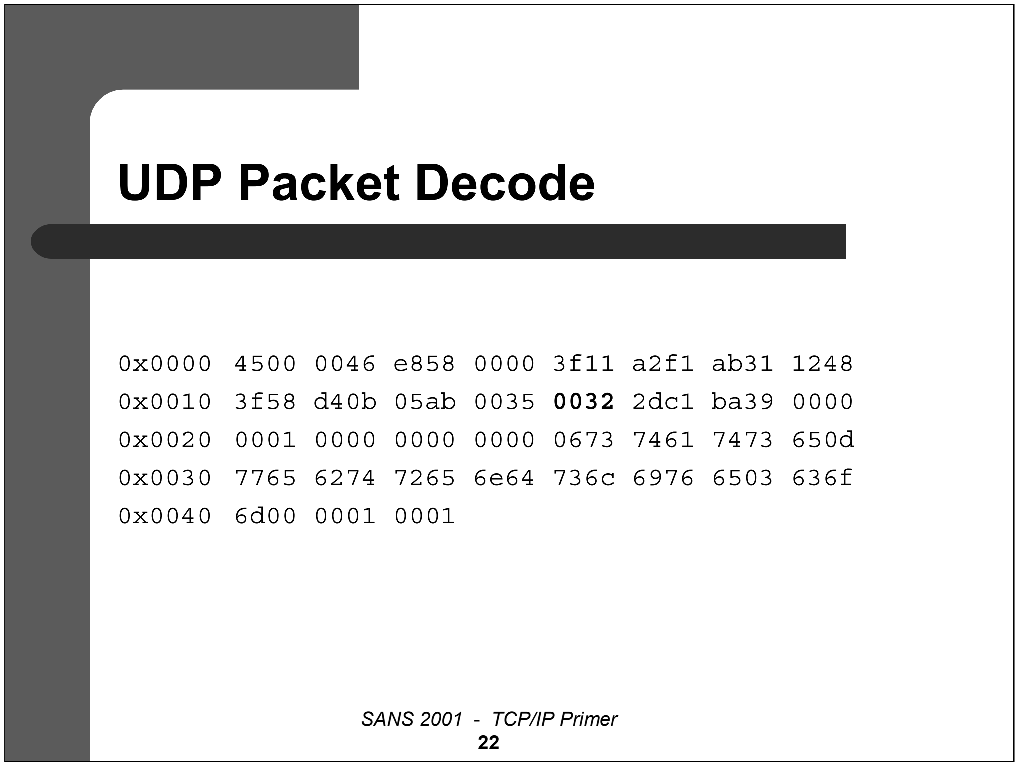
UDP Packet Decode

```
0x0000  4500 0046 e858 0000 3f11 a2f1 ab31 1248
0x0010  3f58 d40b 05ab 0035 0032 2dc1 ba39 0000
0x0020  0001 0000 0000 0000 0673 7461 7473 650d
0x0030  7765 6274 7265 6e64 736c 6976 6503 636f
0x0040  6d00 0001 0001
```

SANS 2001 - TCP/IP Primer
21

The first two fields in the header are used to identify the source and the destination port numbers. On computer systems, different services will 'listen' on various port numbers. When they receive a request to that port number, the information in the packet is routed into the program that services that port. In this case, the destination port is 53, which typically is served by the DNS server, or name server. Most often, depending on how the protocol is implemented, if a reply will be sent to the machine that originated the request, it will be sent back to the machine using the source port from the original packet. In the DNS protocol, this is in fact how it works. This means that the following happens:

```
Requestor                Server
1451----------->         53
   (The DNS server now generates a response)
1451        <----------  53
```



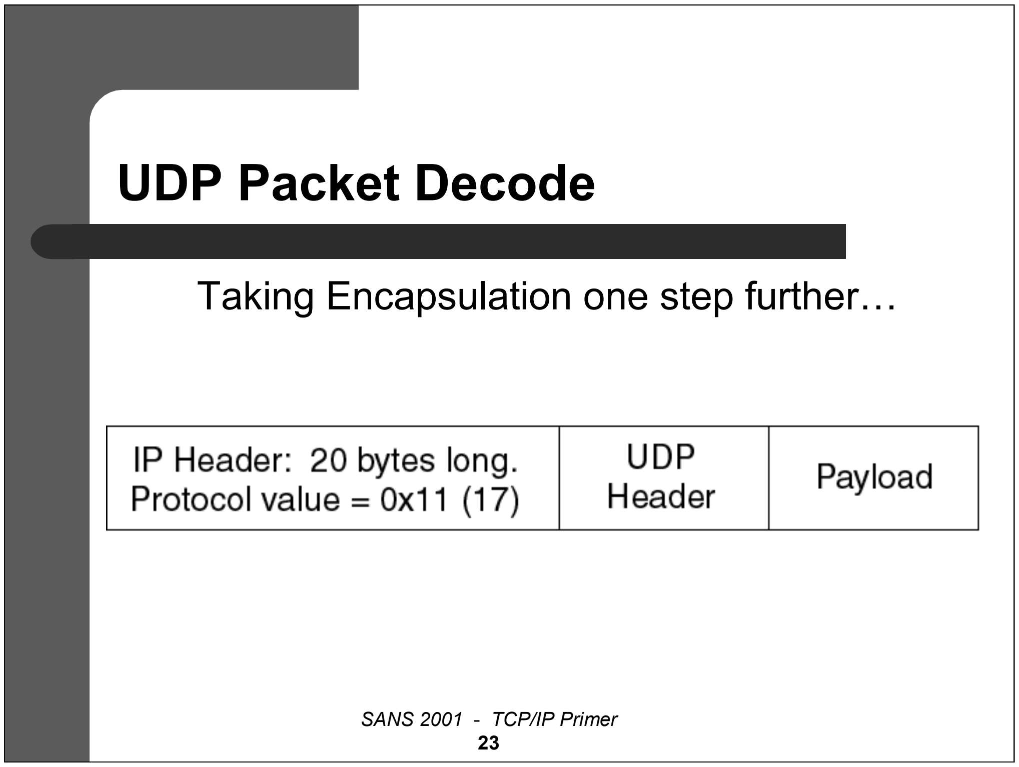
UDP Packet Decode

```
0x0000  4500 0046 e858 0000 3f11 a2f1 ab31 1248
0x0010  3f58 d40b 05ab 0035 0032 2dc1 ba39 0000
0x0020  0001 0000 0000 0000 0673 7461 7473 650d
0x0030  7765 6274 7265 6e64 736c 6976 6503 636f
0x0040  6d00 0001 0001
```

SANS 2001 - TCP/IP Primer
22

The next field in the UDP header is the UDP Message length. This field is analogous to the "datagram length" field in the IP header. It defines how many total bytes are contained in this UDP message. This means that a UDP packet can ***never*** have a length less than 8. In fact, it would be extremely rare (but legal) to have a UDP datagram length of 8.

We will skip the UDP checksum value. Essentially, this (and all other IP header checksum values) are used by the receiving machine to determine whether or not the packet header is valid or if there has been some measure of data corruption during transport.

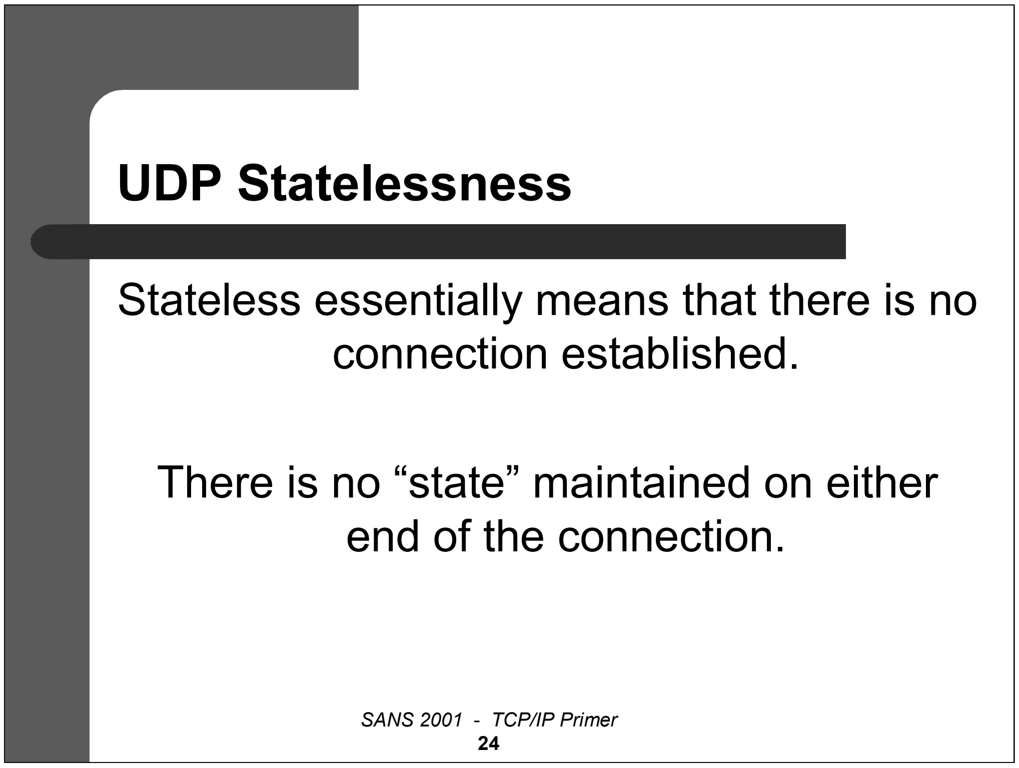


UDP Packet Decode

Taking Encapsulation one step further…

IP Header: 20 bytes long. Protocol value = 0x11 (17)	UDP Header	Payload

SANS 2001 - TCP/IP Primer
23



UDP Statelessness

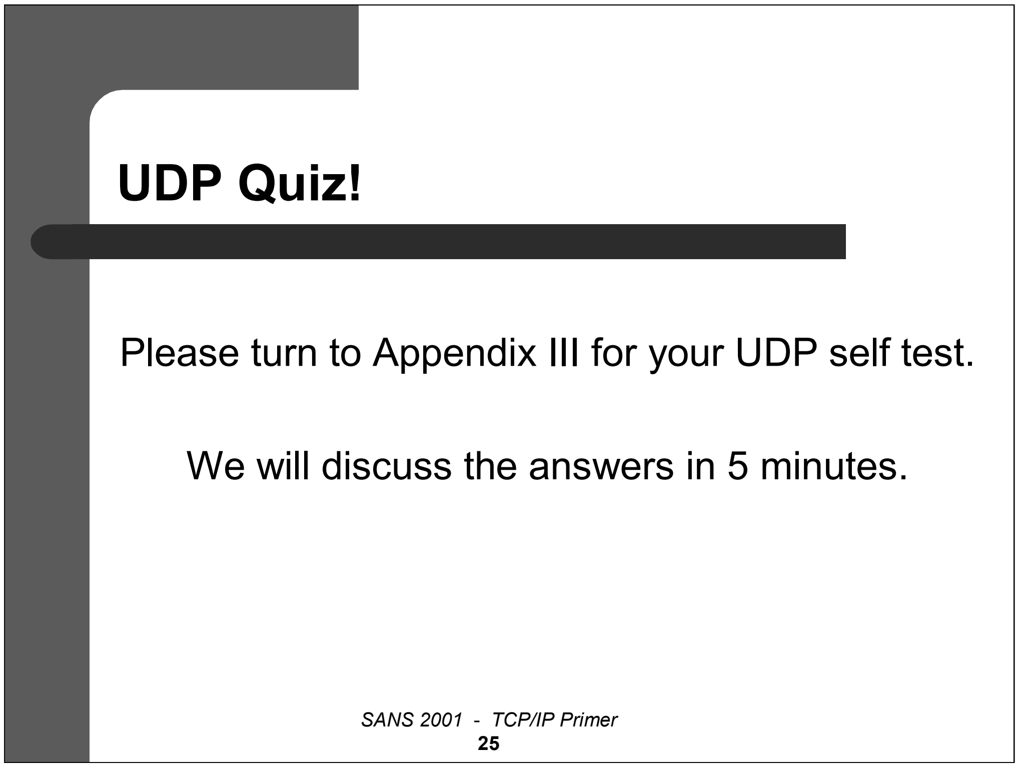
Stateless essentially means that there is no connection established.

There is no "state" maintained on either end of the connection.

SANS 2001 - TCP/IP Primer
24

The idea of state refers to the notion that the machines at either end of a data communication know where they stand in a certain data transaction. Since UDP was designed as a messaging protocol, the UDP implementation itself does not maintain this information. This doesn't mean that you can't simulate "state" using UDP. Most NFS implementations work in this way. The "state", however, is implemented and maintained via a "user space" implementation. By "user space", we are referring to everything outside of the kernel and the TCP/IP stack itself.

NFS and TFTP are excellent examples of UDP based services that can track state and/or error checking.

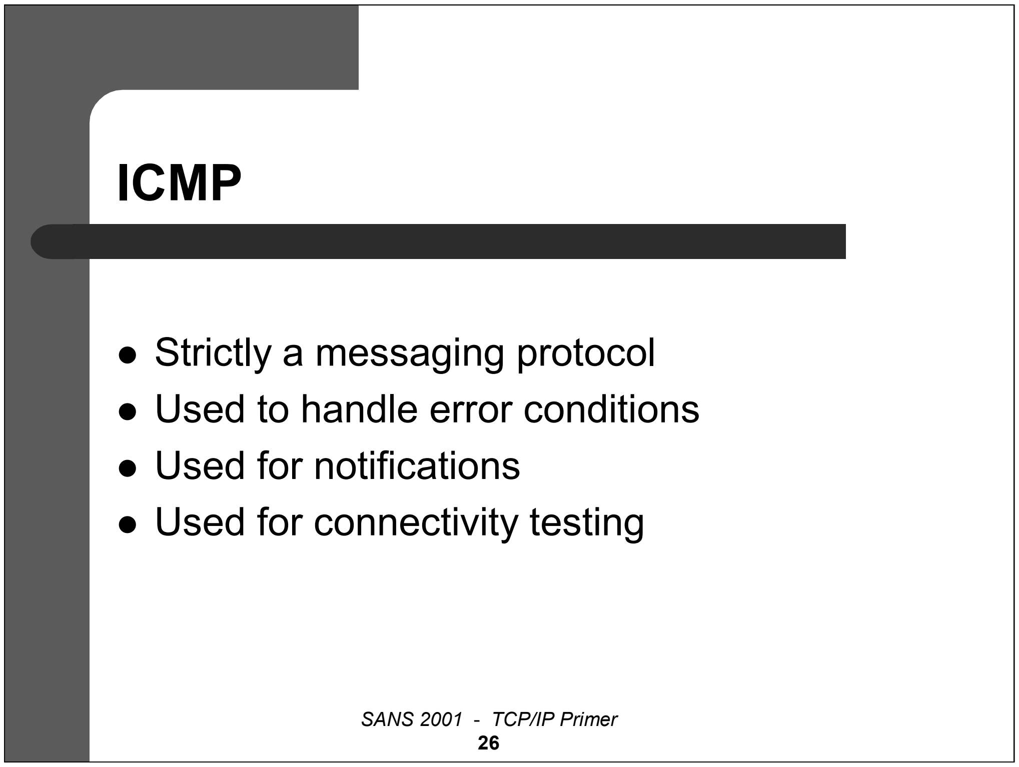


UDP Quiz!

Please turn to Appendix III for your UDP self test.

We will discuss the answers in 5 minutes.

SANS 2001 - TCP/IP Primer
25

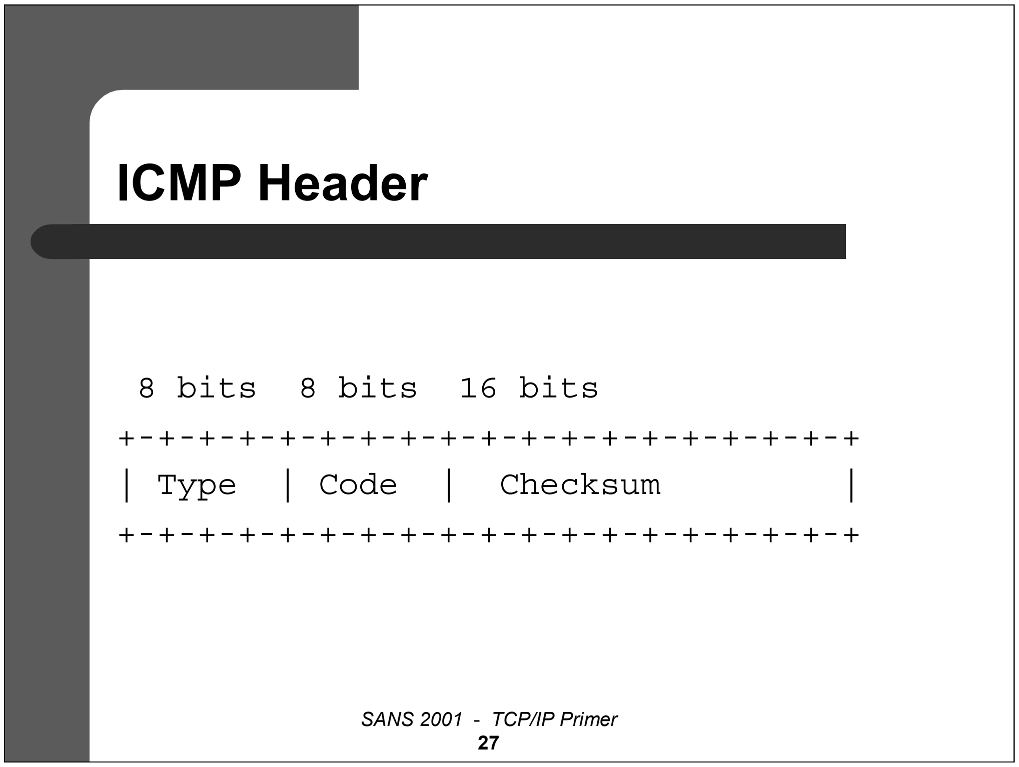


ICMP

- Strictly a messaging protocol
- Used to handle error conditions
- Used for notifications
- Used for connectivity testing

SANS 2001 - TCP/IP Primer
26

ICMP is a fairly simple protocol. While UDP too is a messaging protocol, it is actually intended to carry application data. ICMP, however, is meant to do what it name implies. It stands for "Internet Control Message Protocol". This is the protocol used when a packet is sent to a host that doesn't exist, or to notify a host that it is not permitted to send packets to a particular network. It is also used to tell hosts about cheaper routes, although routing updates is typically handled by a completely different set of protocols. We are all probably familiar with the 'Ping' program, used to send test packets to see if a host responds and perhaps see what sort of latency is involved in a round trip to that host.

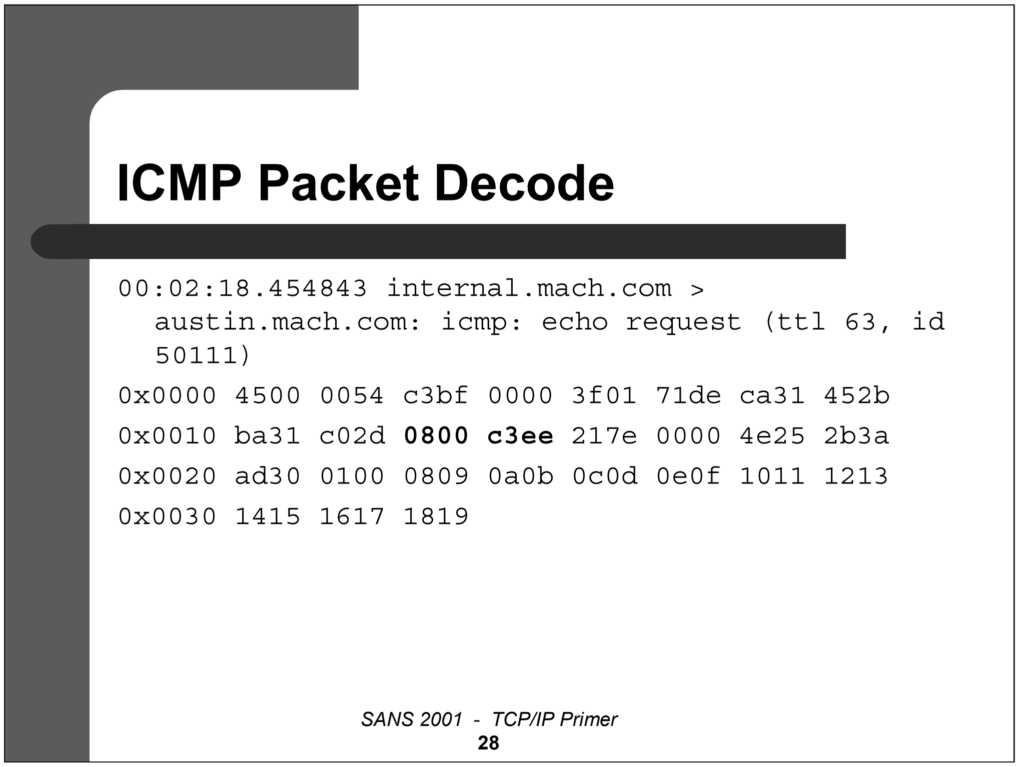


ICMP Header

```
 8 bits   8 bits   16 bits
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type   | Code   |   Checksum         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SANS 2001 - TCP/IP Primer
27

Here is the ICMP header. We will not spend nearly as much time taking apart the ICMP header as we have the IP or UDP headers. We will look at some samples that can help us to identify “predicted” behavior for this protocol.

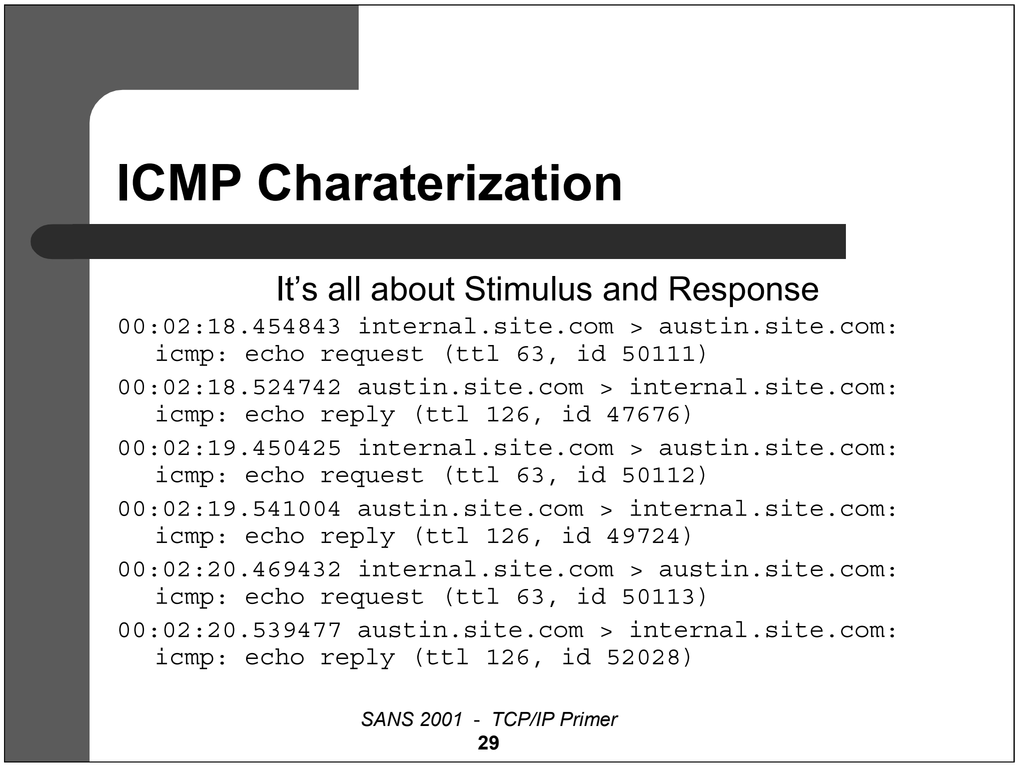

ICMP Packet Decode

```
00:02:18.454843 internal.mach.com >
  austin.mach.com: icmp: echo request (ttl 63, id
  50111)
0x0000 4500 0054 c3bf 0000 3f01 71de ca31 452b
0x0010 ba31 c02d 0800 c3ee 217e 0000 4e25 2b3a
0x0020 ad30 0100 0809 0a0b 0c0d 0e0f 1011 1213
0x0030 1415 1617 1819
```

SANS 2001 - TCP/IP Primer
28

Here, again, we have hilighted the header of interest. The first two bytes of the header are used to define what sort of ICMP packet this is, and there are many kinds. For a full list, please see Appendix IV. We also have an ICMP Checksum which we will treat in the same way as the other checksums we have encountered.

Using the table in Appendix IV, we find that this is an ICMP Echo request, or a ‘ping’ packet.



ICMP Charaterization

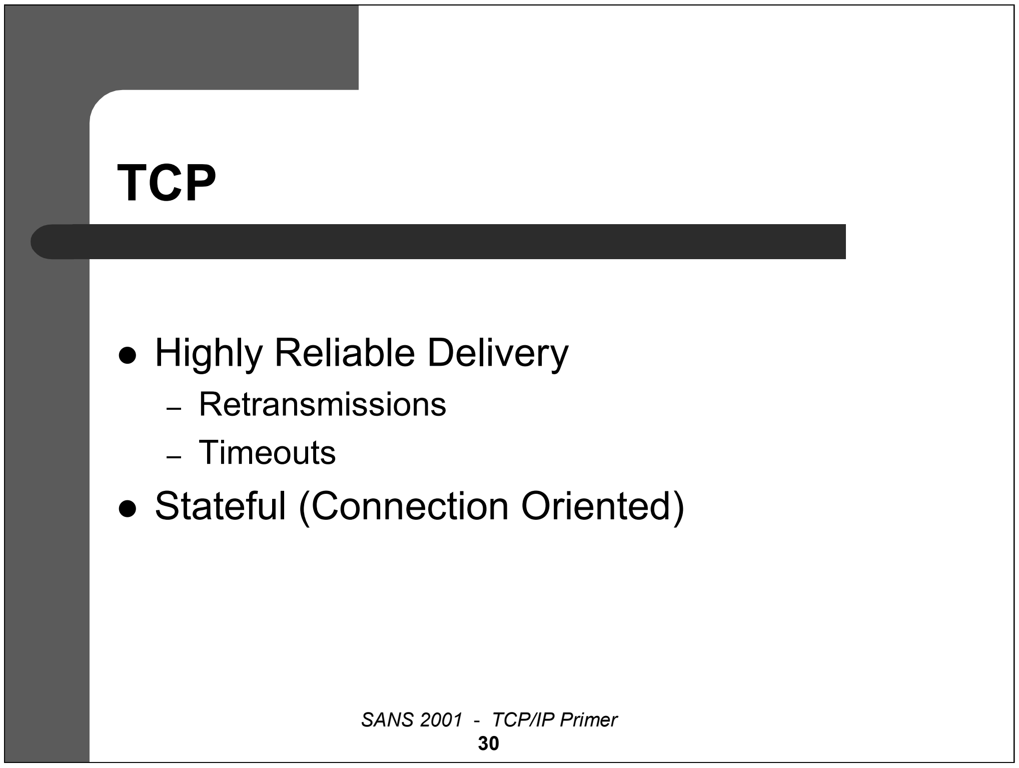
It's all about Stimulus and Response

```
00:02:18.454843 internal.site.com > austin.site.com:
   icmp: echo request (ttl 63, id 50111)
00:02:18.524742 austin.site.com > internal.site.com:
   icmp: echo reply (ttl 126, id 47676)
00:02:19.450425 internal.site.com > austin.site.com:
   icmp: echo request (ttl 63, id 50112)
00:02:19.541004 austin.site.com > internal.site.com:
   icmp: echo reply (ttl 126, id 49724)
00:02:20.469432 internal.site.com > austin.site.com:
   icmp: echo request (ttl 63, id 50113)
00:02:20.539477 austin.site.com > internal.site.com:
   icmp: echo reply (ttl 126, id 52028)
```

SANS 2001 - TCP/IP Primer
29

The various types of ICMP packets behave in very specific ways. Here we have a normal 'ping' request and reply. Internal sends a stimulus to austin and austin sends back a response.

While this may seem simple, it is important to know what the expected behavior is. Here's an easy one: How many replies should you get to a single ping? This may seem like a silly question, but when you're looking at a network someday and see multiple responses, you'll know that something's up.



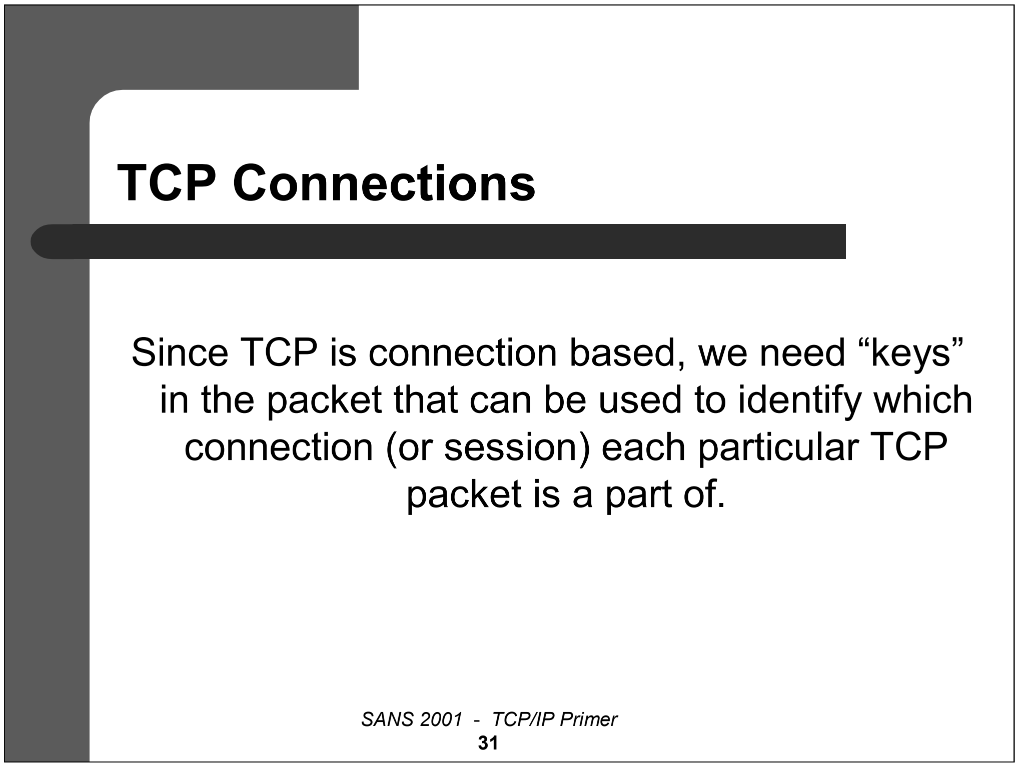
TCP

- Highly Reliable Delivery
  - Retransmissions
  - Timeouts
- Stateful (Connection Oriented)

SANS 2001 - TCP/IP Primer
30

Unlike the other sub protocols that we've looked at, TCP is highly focused on guaranteed delivery of data. While we mentioned that it is possible to implement "state" on top of other stateless protocols, in TCP the state is an inherent part of the connection. In fact, TCP is the only one of the three protocols that we will discuss today that has something called a connection.

This high reliability does introduce some added overhead network wise, however, which makes TCP undesirable for small, isolated transactions. We will see this in our packet dumps as we proceed.

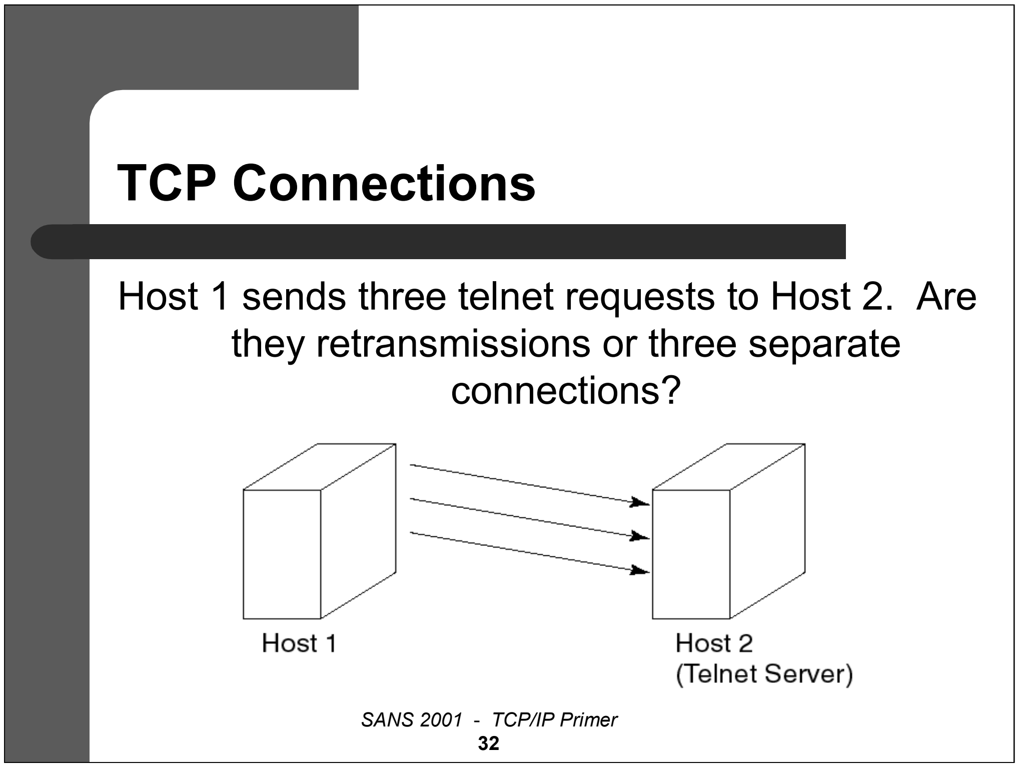


TCP Connections

Since TCP is connection based, we need “keys” in the packet that can be used to identify which connection (or session) each particular TCP packet is a part of.

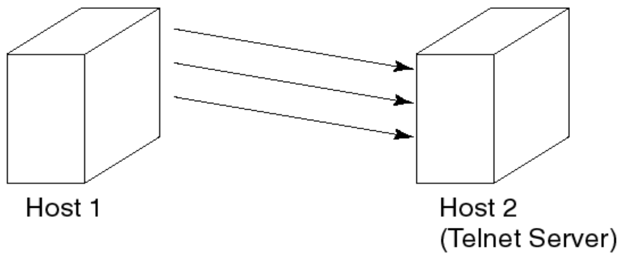
SANS 2001 - TCP/IP Primer
31

In order to implement a connection based protocol, there needs to be certain key pieces of data within the packet that can be used to identify which session each packet is a part of.



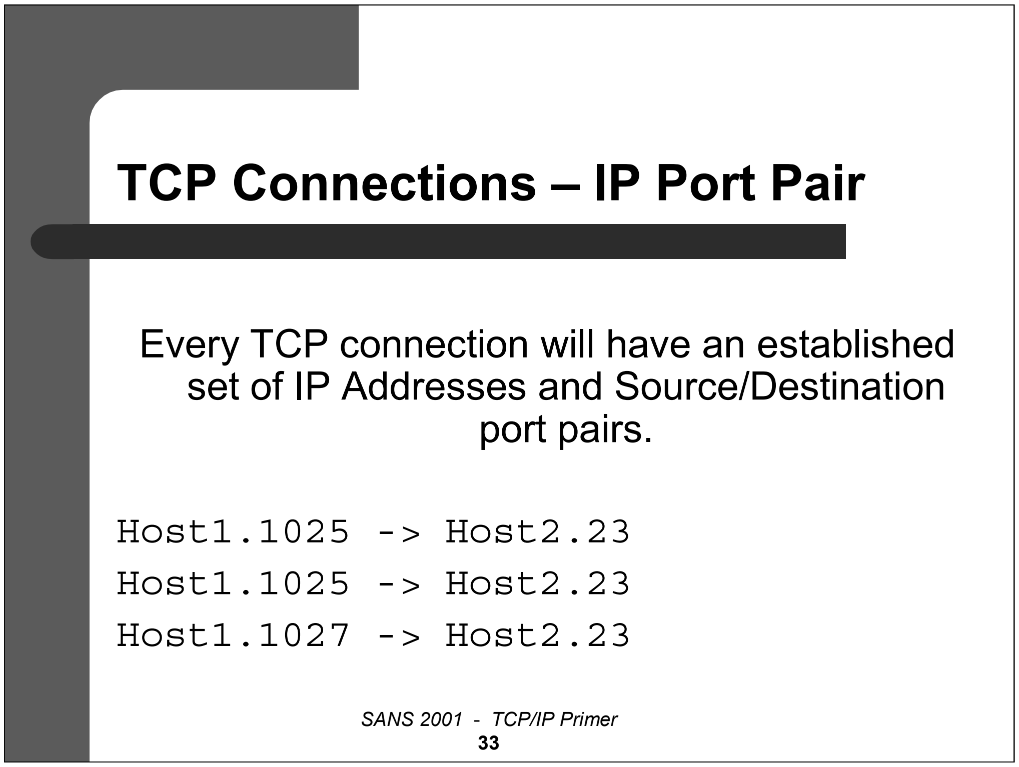
TCP Connections

Host 1 sends three telnet requests to Host 2. Are they retransmissions or three separate connections?



SANS 2001 - TCP/IP Primer
32

Looking at this slide we can begin to envision one of the problems with implementing state. How do we know if the three packets are part of the same session, part of three different sessions or some combination thereof? Without a system of identifying TCP Sessions, we can't tell.



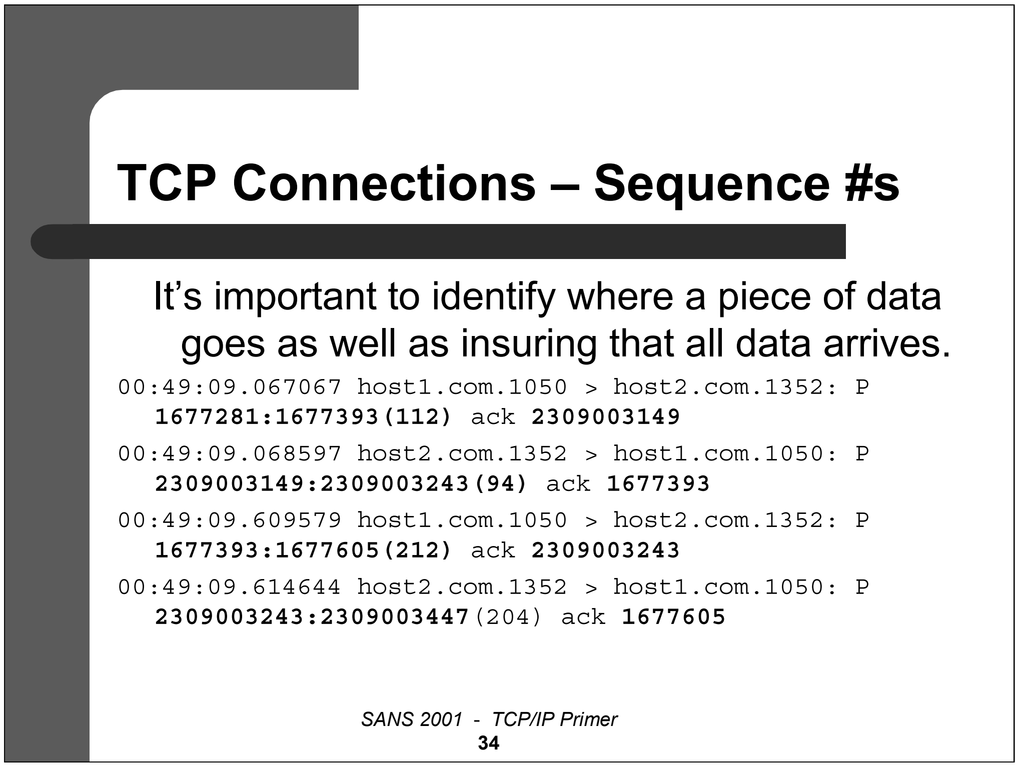
TCP Connections – IP Port Pair

Every TCP connection will have an established set of IP Addresses and Source/Destination port pairs.

```
Host1.1025 -> Host2.23
Host1.1025 -> Host2.23
Host1.1027 -> Host2.23
```

SANS 2001 - TCP/IP Primer
33

The IP Port Pair concept is the key way to identify a session. The TCP stack identifies a session by matching the source IP and source port with the destination IP and destination port. In this way the stack is able to determine if a particular packet is part of an established session or not, and if it is, which instance of the server code to send the data to.



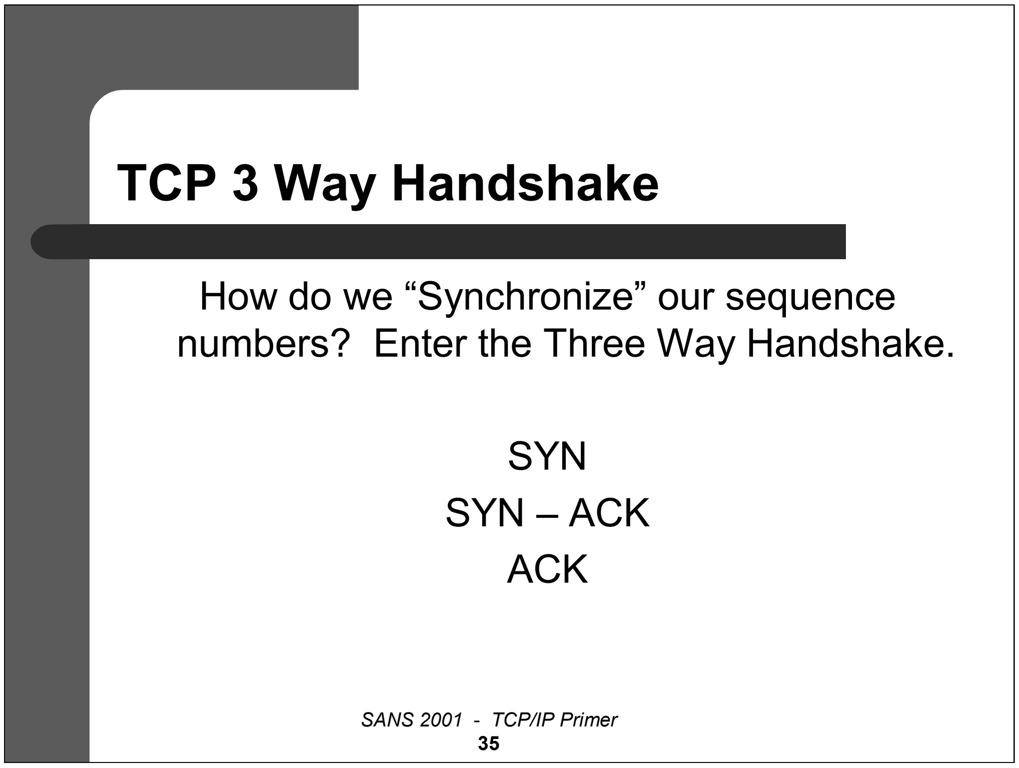
TCP Connections – Sequence #s

It's important to identify where a piece of data goes as well as insuring that all data arrives.

```
00:49:09.067067 host1.com.1050 > host2.com.1352: P
  1677281:1677393(112) ack 2309003149
00:49:09.068597 host2.com.1352 > host1.com.1050: P
  2309003149:2309003243(94) ack 1677393
00:49:09.609579 host1.com.1050 > host2.com.1352: P
  1677393:1677605(212) ack 2309003243
00:49:09.614644 host2.com.1352 > host1.com.1050: P
  2309003243:2309003447(204) ack 1677605
```

SANS 2001 - TCP/IP Primer
34

In order to implement "State" into the connection or session, we need to know where in the session a particular packet goes. It's sort of like a big jig saw puzzle of packets. The stack needs to be able to identify where a particular piece goes or the data may be garbled. You may wonder at first why this is necessary since the client ought to send the data out in order. However, there is no guarantee that any specific packet will be delivered. This state implementation allows the stack to recognize when a packet has been lost. There is also no guarantee that all packets will be routed in the same way. This means that while the packets may leave in order, they may not arrive in order. How can we put them back together? Sequence numbers is the answer. The Sequence numbers are used to track how many bytes have been sent and acknowledged by each side of the connection.



TCP 3 Way Handshake

How do we "Synchronize" our sequence numbers? Enter the Three Way Handshake.

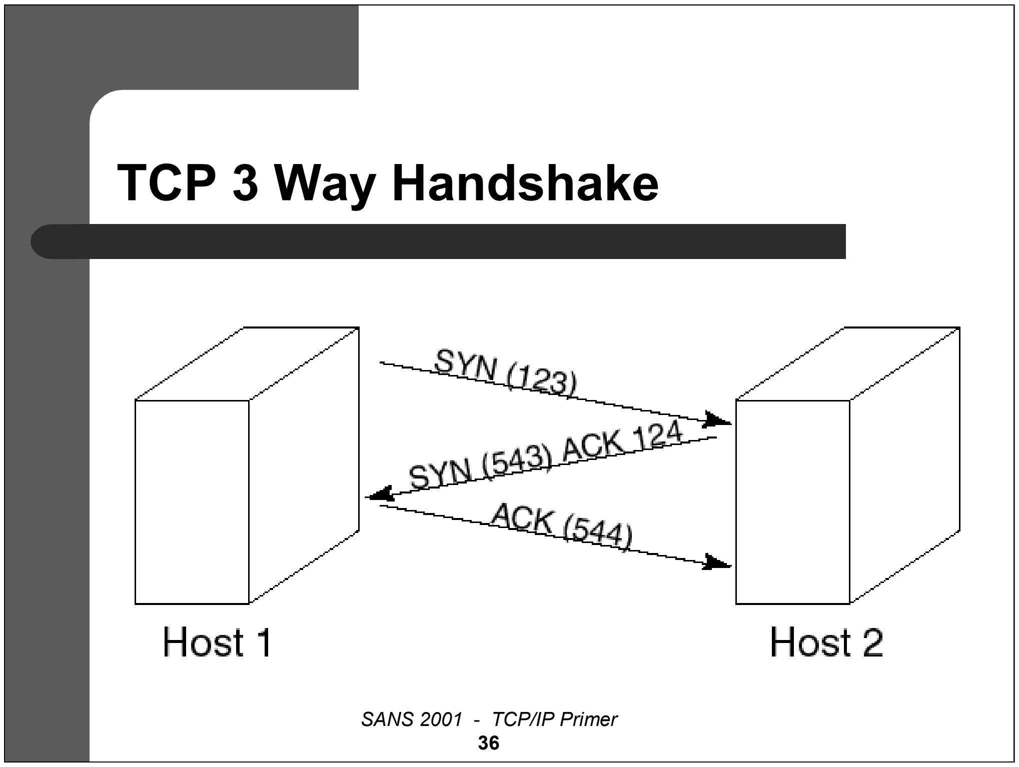
SYN

SYN – ACK

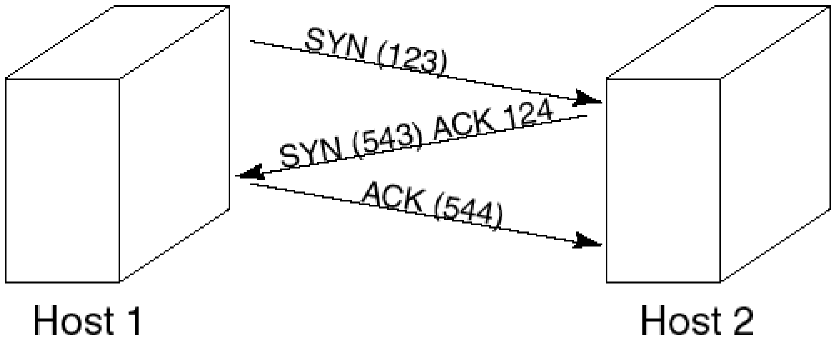
ACK

SANS 2001 - TCP/IP Primer
35

Now that we can see the usefulness of sequence numbers in addition to the IP Port pairs, we need to have a way to "synchronize" our sequence numbers. Why? The simplest reason is this: Imagine you and I are both logged into the same unix server, and we both decide to telnet to another host. Aside from the port pairs, how can the data streams be identified? An even better reason is this. Imagine that you're logged into a server and I'm a malicious person somewhere else in the world. What would stop me from injecting packets into your stream of data? I could certainly forge the source port and the source IP address, but if the sequence numbers are random to start with, how am I going to get my hands on that number without sniffing your connection?



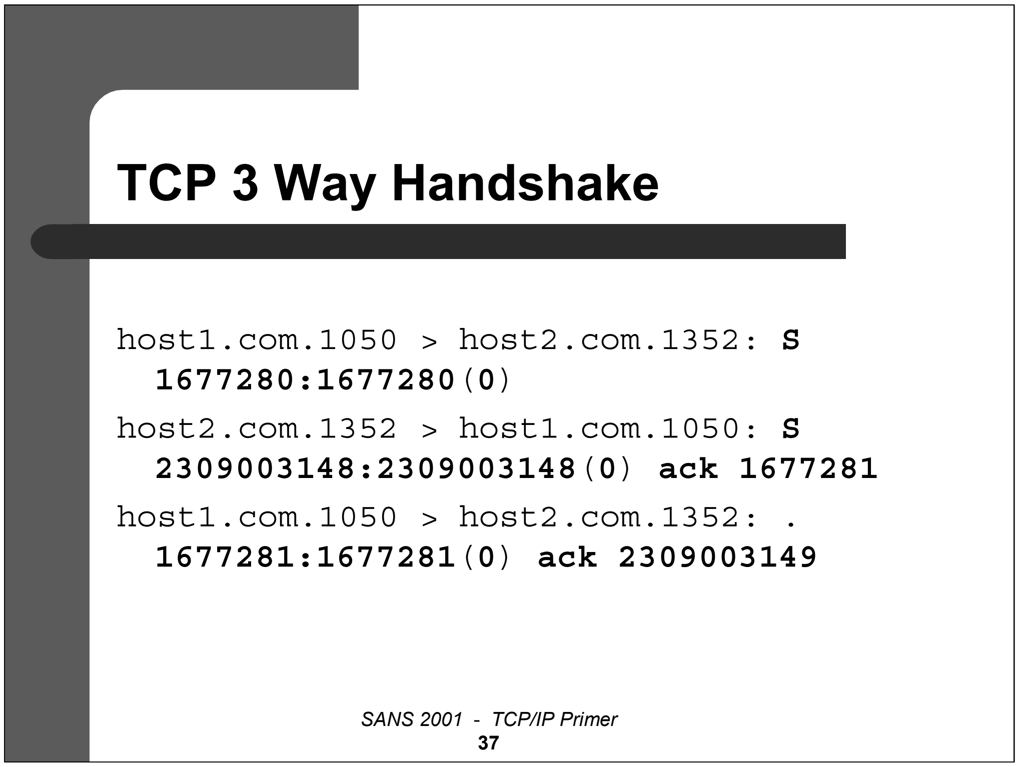
TCP 3 Way Handshake



Here we have the three way handshake. Host 1 starts off with an initial SYN packet. SYN stands for “Synchronize”, as in Synchronize Sequence numbers. Host 1 includes the starting sequence number that it has chosen, in this case “123”.

Next host 2 Acknowledges (ACKs) that sequence number +1, to indicate that it is expecting the next byte. Additionally, it sends a SYN of its own with an initial sequence number of “543”.

Finally, Host 1 Acknowledges the SYN from Host 2, sending an ACK #544, indicating that it is ready to receive the next byte. We now have an established TCP connection.

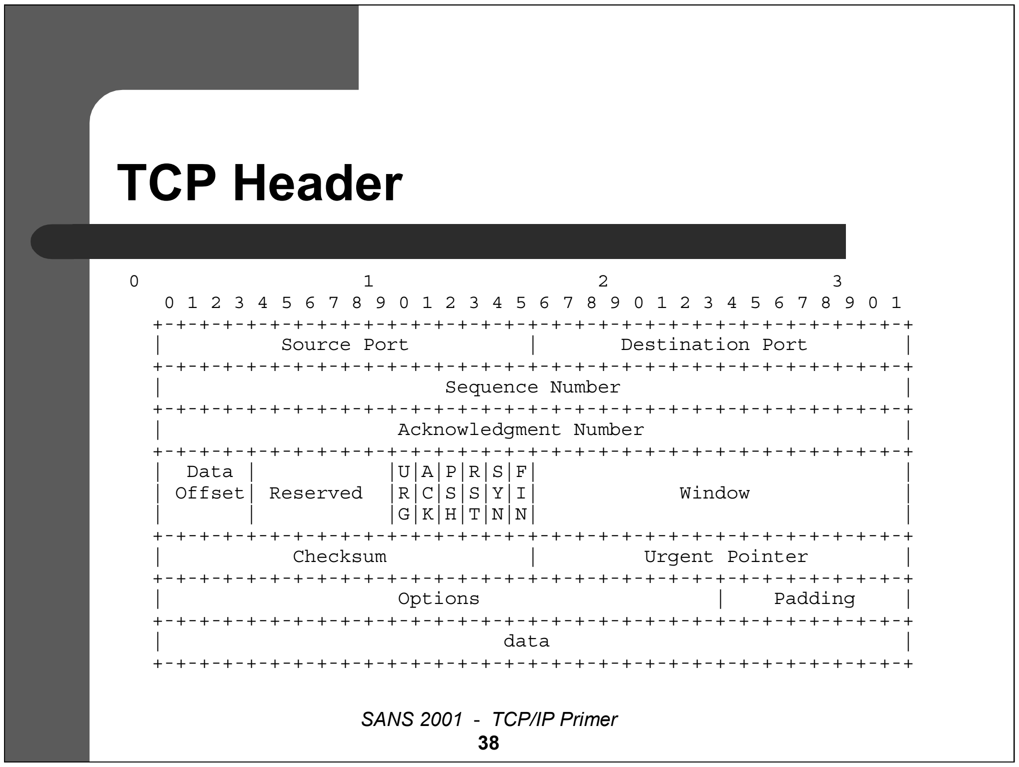


TCP 3 Way Handshake

```
host1.com.1050 > host2.com.1352: S
  1677280:1677280(0)
host2.com.1352 > host1.com.1050: S
  2309003148:2309003148(0) ack 1677281
host1.com.1050 > host2.com.1352: .
  1677281:1677281(0) ack 2309003149
```

SANS 2001 - TCP/IP Primer
37

Here's a 3 way handshake "from the wild". This is what they really look like. Notice the incrementing sequence numbers and the SYN, SYN ACK, ACK. It is also noteworthy that the sequence number in the last packet is still 1677281. The sequence number is included even if no data is actually sent. This is important because if the number were not included, it would become more difficult to place packets in the data stream.

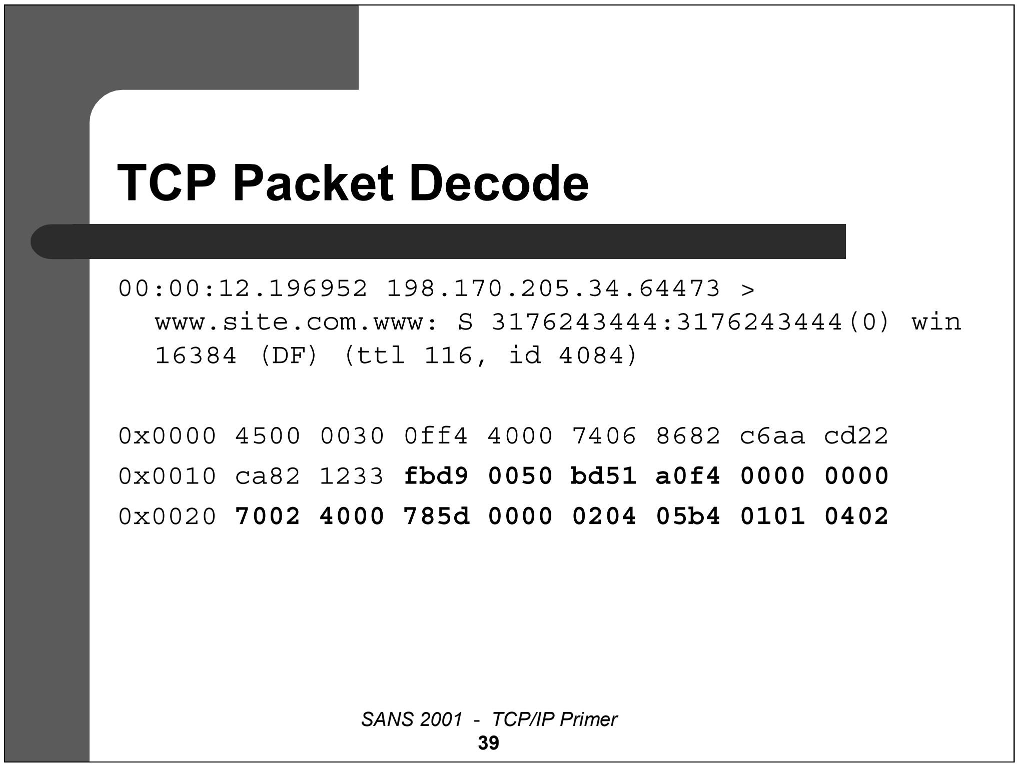


TCP Header

```
0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SANS 2001 - TCP/IP Primer
38

There it is, the TCP header, and it's a monster. Remember the basics that we've already learned. This header would immediately follow the IP header if the protocol byte in the IP header is set to 6. Once again, we'll go through this packet piece by piece and then look at some actual traffic to see if we can identify connections and what's going on in connections.



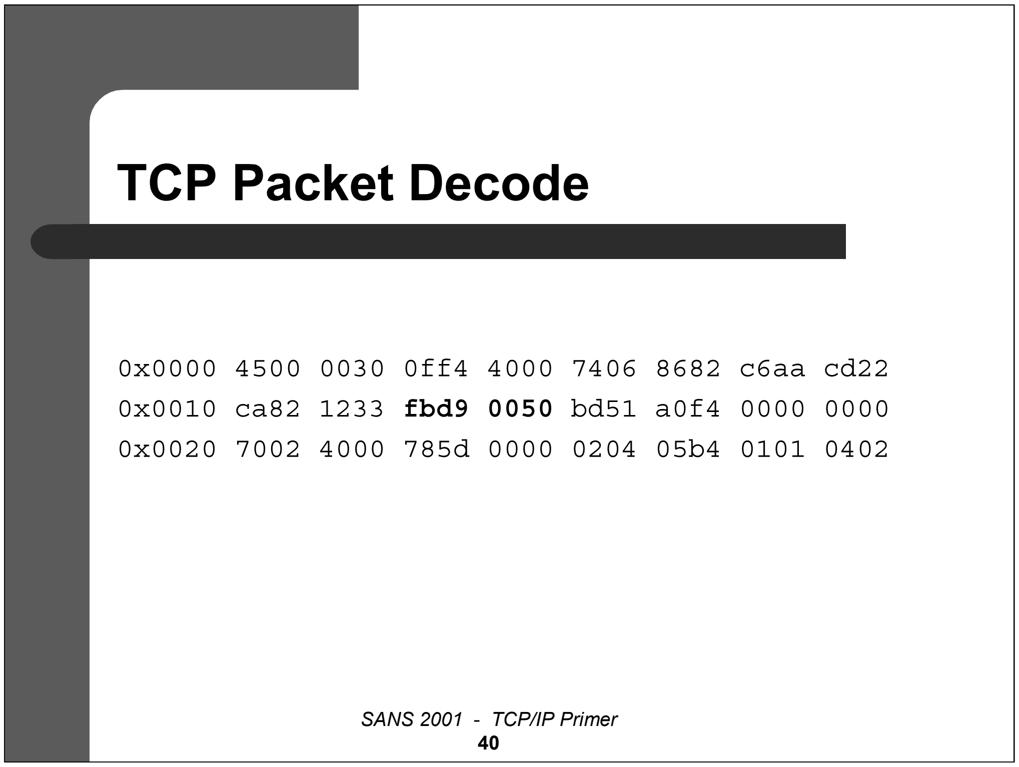
TCP Packet Decode

```
00:00:12.196952 198.170.205.34.64473 >
  www.site.com.www: S 3176243444:3176243444(0) win
  16384 (DF) (ttl 116, id 4084)

0x0000 4500 0030 0ff4 4000 7406 8682 c6aa cd22
0x0010 ca82 1233 fbd9 0050 bd51 a0f4 0000 0000
0x0020 7002 4000 785d 0000 0204 05b4 0101 0402
```

SANS 2001 - TCP/IP Primer
39

Here's the packet that we will examine as we decode the TCP header. We see the familiar 20 byte IP header followed immediately by the beginning of the TCP header.

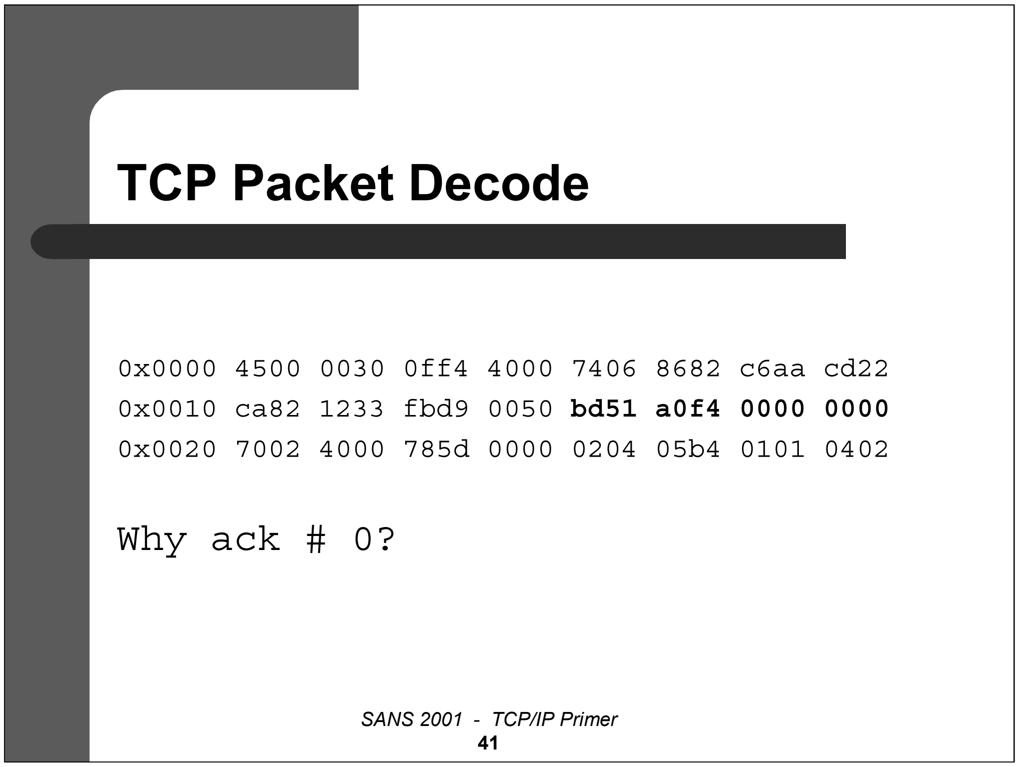


TCP Packet Decode

```
0x0000 4500 0030 0ff4 4000 7406 8682 c6aa cd22
0x0010 ca82 1233 fbd9 0050 bd51 a0f4 0000 0000
0x0020 7002 4000 785d 0000 0204 05b4 0101 0402
```

SANS 2001 - TCP/IP Primer
40

First off we have the source and destination port. This initial portion of the TCP header is identical to the UDP header. A lot of thought went into the placement of the individual pieces of data and their placement within the header. It seems naturally reasonable that the next most important piece of information for any intervening IP stack or for the endpoint stacks after the source and destination addresses would be the source and destination ports. In this example we have a source port of 64473 and a destination port of 80.



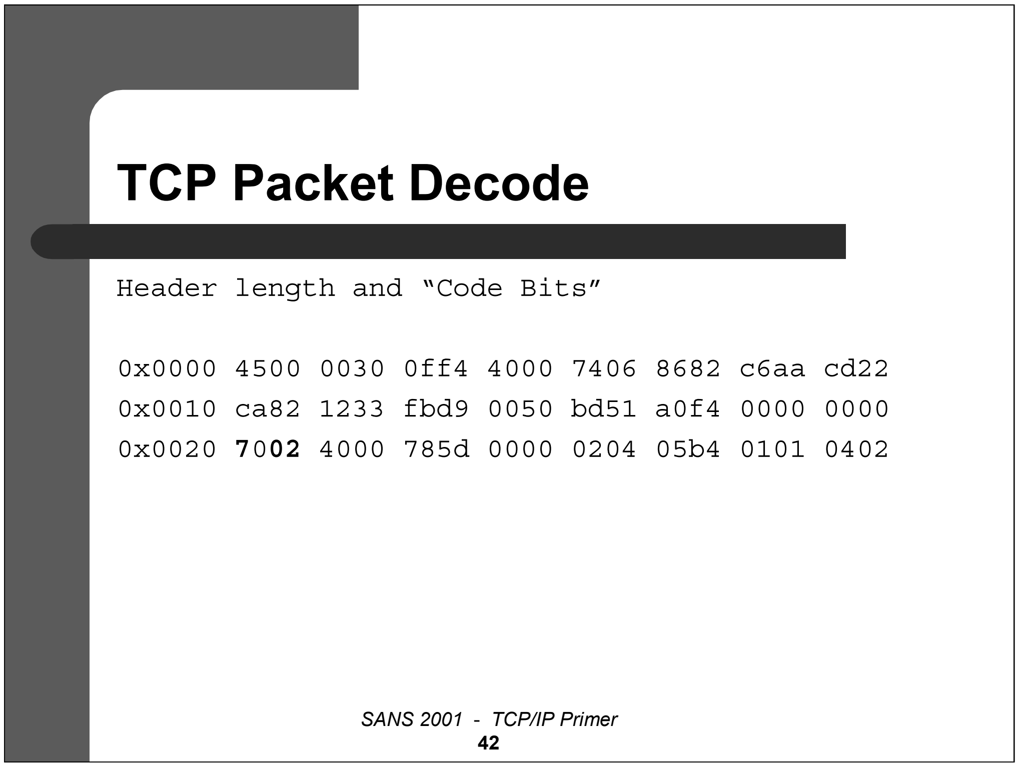
TCP Packet Decode

```
0x0000 4500 0030 0ff4 4000 7406 8682 c6aa cd22
0x0010 ca82 1233 fbd9 0050 bd51 a0f4 0000 0000
0x0020 7002 4000 785d 0000 0204 05b4 0101 0402
```

Why ack # 0?

SANS 2001 - TCP/IP Primer
41

The next two fields are each 32 bit values. These are the sequence numbers. Actually, the first value is the sequence number of *this* packet and the second value is the sequence number that is being acknowledged or that was last acknowledged by this end of the connection (plus one, of course). Decoding these two numbers we come up with a sequence number of "3176243444" and an acknowledgement number of "0". Why is it zero? Can you tell from your knowledge of TCP so far?



TCP Packet Decode

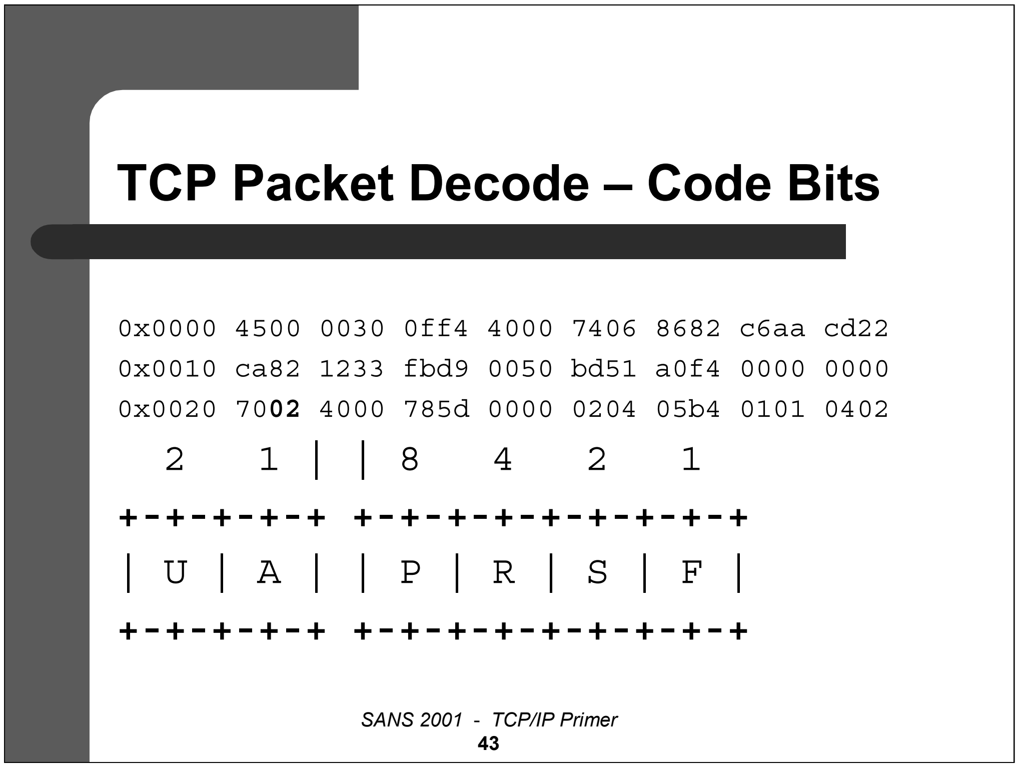
Header length and "Code Bits"

```
0x0000 4500 0030 0ff4 4000 7406 8682 c6aa cd22
0x0010 ca82 1233 fbd9 0050 bd51 a0f4 0000 0000
0x0020 7002 4000 785d 0000 0204 05b4 0101 0402
```

SANS 2001 - TCP/IP Primer
42

Here we have hilighted both the header length and the code bits. TCP is somewhat different from UDP in that you can not tell from the TCP header how much data will follow. The only thing the TCP header tells you is where the header ends and the data begins (if there is data!). In order to determine how much data is attached, we would need to refer back to the IP Datagram size and subtract off the size of the IP header and the size of the TCP header. This would give us our total data payload size. Much like its IP cousin (or perhaps parent), the TCP header size field marks off double words. That means that in this case, the TCP header is 7 X 4 bytes long, or 28. The shortest possible size for a TCP header is 20 bytes (a value of 5 in the TCP header length field). The header becomes larger if there are TCP options attached.

The Code Bits is where all of the fun happens. These individual bits are used as switches to define what the packet is supposed to be doing. We'll discuss them further on the next slide.



TCP Packet Decode – Code Bits

```
0x0000 4500 0030 0ff4 4000 7406 8682 c6aa cd22
0x0010 ca82 1233 fbd9 0050 bd51 a0f4 0000 0000
0x0020 7002 4000 785d 0000 0204 05b4 0101 0402
  2   1 | | 8   4   2   1
+-+-+-+-+ +-+-+-+-+-+-+-+-+
| U | A | | P | R | S | F |
+-+-+-+-+ +-+-+-+-+-+-+-+-+
```

SANS 2001 - TCP/IP Primer
43

The code bits (or TCP flags as they are sometimes called) are:

Urgent: Rarely used. Intended to elevate the priority of a packet.

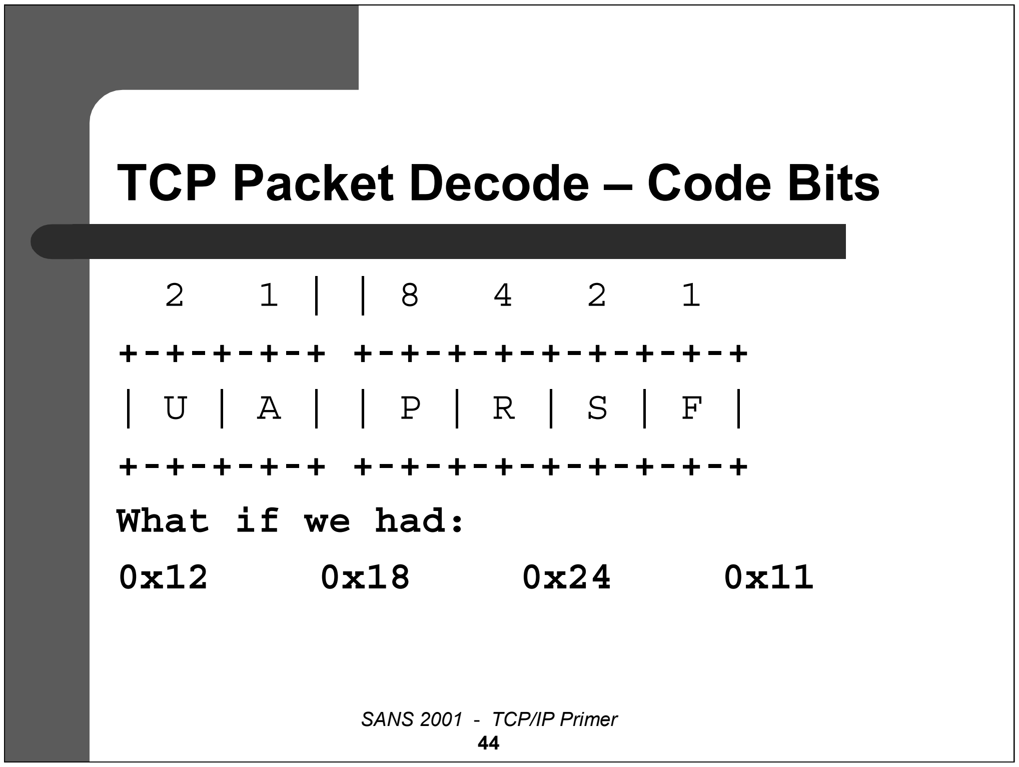
Ack: Acknowledge a particular sequence number (data received, 3 way handshake)

Push: Indicate to the TCP stack that the data ought to be delivered to the application now.

Reset: Reset a connection. Used for error conditions and timeouts.

Syn: Synchronize Sequence numbers (initial connection)

Fin: Signal for the close of the connection



TCP Packet Decode – Code Bits

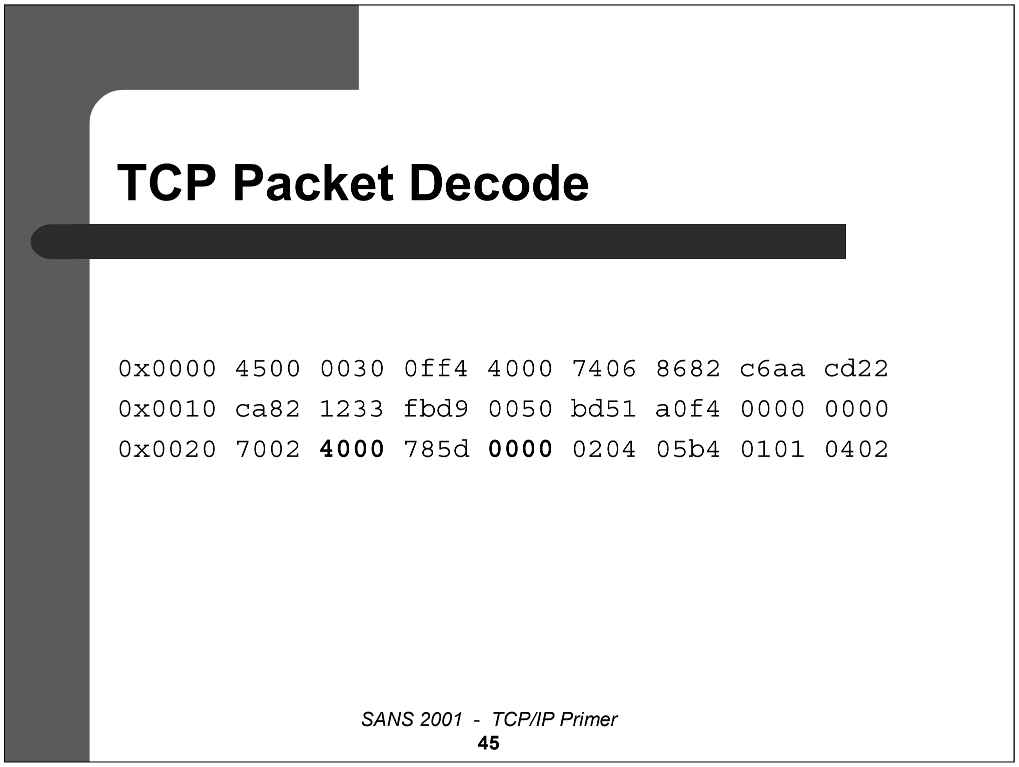
```
  2   1 | | 8   4   2   1
+-+-+-+-+ +-+-+-+-+-+-+-+-+
| U | A | | P | R | S | F |
+-+-+-+-+ +-+-+-+-+-+-+-+-+
```

What if we had:

0x12   0x18   0x24   0x11

SANS 2001 - TCP/IP Primer
44

Please figure out which bits would be set in the above examples.



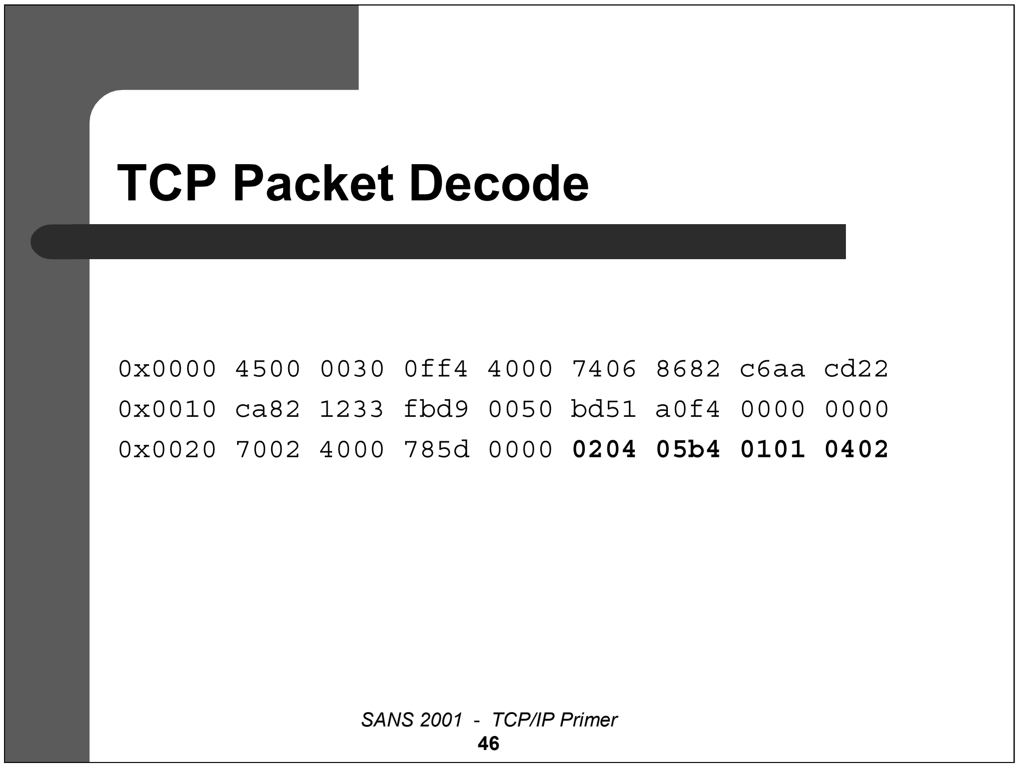
TCP Packet Decode

```
0x0000 4500 0030 0ff4 4000 7406 8682 c6aa cd22
0x0010 ca82 1233 fbd9 0050 bd51 a0f4 0000 0000
0x0020 7002 4000 785d 0000 0204 05b4 0101 0402
```

SANS 2001 - TCP/IP Primer
45

The Window Size in this packet is set to 0x4000 or 16384. The window size is used to implement flow control in the connection. Flow control is used to throttle how much data is sent through at a time. A window size of 16384 means that the other end of the connection can send up to 16384 bytes through. In fact, if there is that much or more data to send, the other side of the connection will proceed to send that much data without waiting for an acknowledgement, however once 16384 bytes of data has been sent, the sender will stop and wait for an acknowledgement (and perhaps an adjustment in the window size. It can be different in every packet).

The Urgent pointer is only useful if the URG bit is set. The URG bit indicates that the value in this field is valid. This field indicates where into the data payload the urgent data resides.

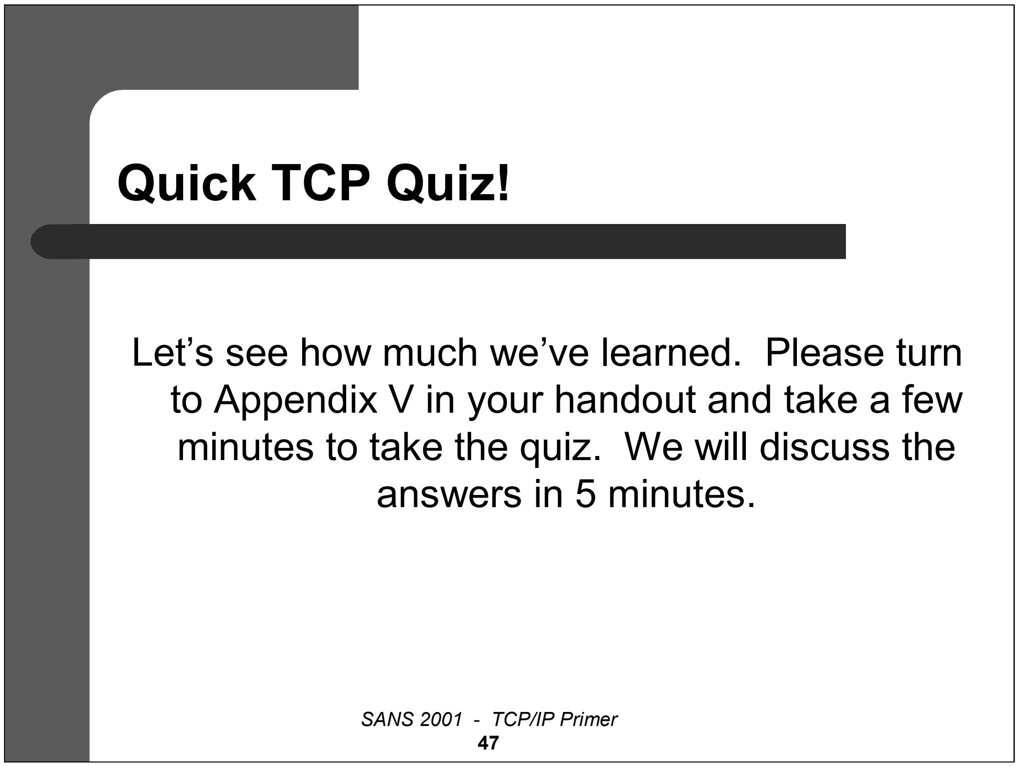


TCP Packet Decode

```
0x0000 4500 0030 0ff4 4000 7406 8682 c6aa cd22
0x0010 ca82 1233 fbd9 0050 bd51 a0f4 0000 0000
0x0020 7002 4000 785d 0000 0204 05b4 0101 0402
```

SANS 2001 - TCP/IP Primer
46

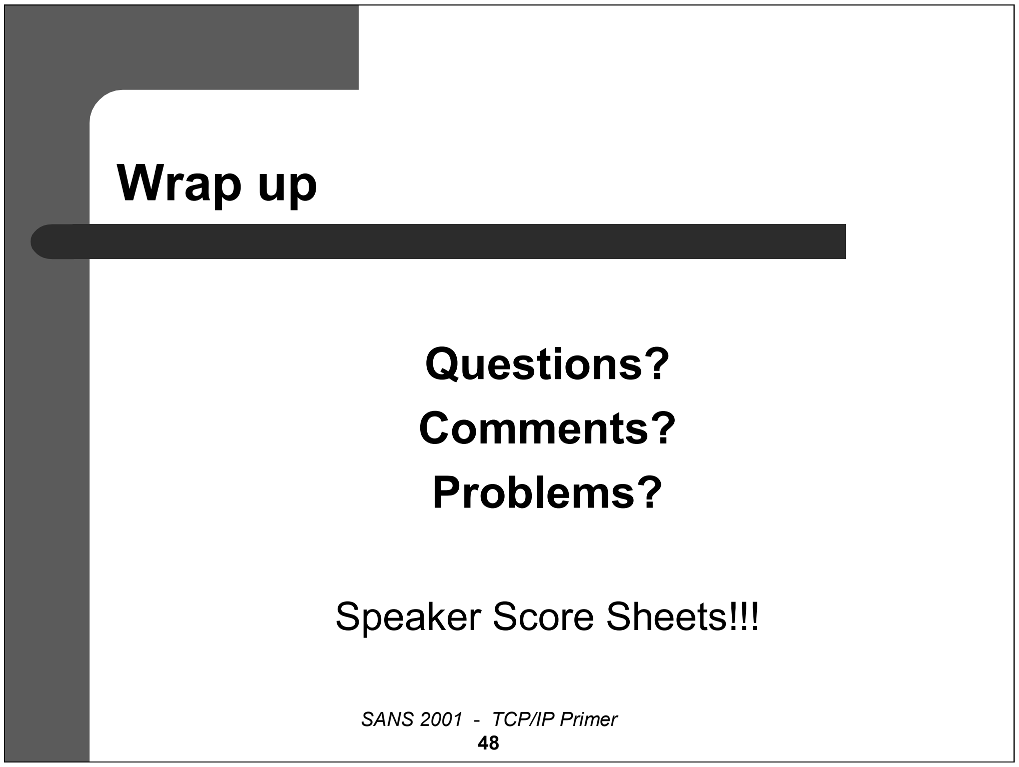
Finally we have any TCP Options that may be included in the packet. These are not required. They will only appear when the TCP Header size is greater than 5. There must always be some multiple of 4 bytes of options if there are any at all due to the nature of the definition of the TCP header field.



Quick TCP Quiz!

Let's see how much we've learned. Please turn to Appendix V in your handout and take a few minutes to take the quiz. We will discuss the answers in 5 minutes.

SANS 2001 - TCP/IP Primer
47



Wrap up

Questions?

Comments?

Problems?

Speaker Score Sheets!!!

SANS 2001 - TCP/IP Primer

48

Appendix I

PROTOCOL NUMBERS

In the Internet Protocol version 4 (IPv4) [RFC791] there is a field, called "Protocol", to identify the next level protocol. This is an 8 bit field. In Internet Protocol version 6 (IPv6) [RFC1883] this field is called the "Next Header" field.

Assigned Internet Protocol Numbers

Decimal	Keyword	Protocol	References
0	HOPOPT	IPv6 Hop-by-Hop Option	[RFC1883]
1	ICMP	Internet Control Message	[RFC792]
2	IGMP	Internet Group Management	[RFC1112]
3	GGP	Gateway-to-Gateway	[RFC823]
4	IP	IP in IP (encapsulation)	[RFC2003]
5	ST	Stream	[RFC1190,IEN119]
6	TCP	Transmission Control	[RFC793]
7	CBT	CBT	[Ballardie]
8	EGP	Exterior Gateway Protocol	[RFC888,DLM1]
9	IGP	any private interior gateway (used by Cisco for their IGRP)	[IANA]
10	BBN-RCC-MON	BBN RCC Monitoring	[SGC]
11	NVP-II	Network Voice Protocol	[RFC741,SC3]
12	PUP	PUP	[PUP,XEROX]
13	ARGUS	ARGUS	[RWS4]
14	EMCON	EMCON	[BN7]
15	XNET	Cross Net Debugger	[IEN158,JFH2]
16	CHAOS	Chaos	[NC3]
17	UDP	User Datagram	[RFC768,JBP]
18	MUX	Multiplexing	[IEN90,JBP]
19	DCN-MEAS	DCN Measurement Subsystems	[DLM1]
20	HMP	Host Monitoring	[RFC869,RH6]
21	PRM	Packet Radio Measurement	[ZSU]
22	XNS-IDP	XEROX NS IDP	[ETHERNET,XEROX]
23	TRUNK-1	Trunk-1	[BWB6]
24	TRUNK-2	Trunk-2	[BWB6]
25	LEAF-1	Leaf-1	[BWB6]
26	LEAF-2	Leaf-2	[BWB6]
27	RDP	Reliable Data Protocol	[RFC908,RH6]
28	IRTP	Internet Reliable Transaction	[RFC938,TXM]
29	ISO-TP4	ISO Transport Protocol Class 4	[RFC905,RC77]
30	NETBLT	Bulk Data Transfer Protocol	[RFC969,DDC1]
31	MFE-NSP	MFE Network Services Protocol	[MFENET,BCH2]
32	MERIT-INP	MERIT Internodal Protocol	[HWB]
33	SEP	Sequential Exchange Protocol	[JC120]
34	3PC	Third Party Connect Protocol	[SAF3]
35	IDPR	Inter-Domain Policy Routing Protocol	[MXS1]

36	XTP	XTP	[GXC]
37	DDP	Datagram Delivery Protocol	[WXC]
38	IDPR-CMTP	IDPR Control Message Transport Proto	[MXS1]
39	TP++	TP++ Transport Protocol	[DXF]
40	IL	IL Transport Protocol	[Presotto]
41	IPv6	Ipv6	[Deering]
42	SDRP	Source Demand Routing Protocol	[DXE1]
43	IPv6-Route	Routing Header for IPv6	[Deering]
44	IPv6-Frag	Fragment Header for IPv6	[Deering]
45	IDRP	Inter-Domain Routing Protocol	[Sue Hares]
46	RSVP	Reservation Protocol	[Bob Braden]
47	GRE	General Routing Encapsulation	[Tony Li]
48	MHRP	Mobile Host Routing Protocol	[David Johnson]
49	BNA	BNA	[Gary Salamon]
50	ESP	Encap Security Payload for IPv6	[RFC1827]
51	AH	Authentication Header for IPv6	[RFC1826]
52	I-NLSP	Integrated Net Layer Security TUBA	[GLENN]
53	SWIPE	IP with Encryption	[JI6]
54	NARP	NBMA Address Resolution Protocol	[RFC1735]
55	MOBILE	IP Mobility	[Perkins]
56	TLSP	Transport Layer Security Protocol using Kryptonet key management	[Oberg]
57	SKIP	SKIP	[Markson]
58	IPv6-ICMP	ICMP for IPv6	[RFC1883]
59	IPv6-NoNxt	No Next Header for IPv6	[RFC1883]
60	IPv6-Opts	Destination Options for IPv6	[RFC1883]
61		any host internal protocol	[IANA]
62	CFTP	CFTP	[CFTP,HCF2]
63		any local network	[IANA]
64	SAT-EXPAK	SATNET and Backroom EXPAK	[SHB]
65	KRYPTOLAN	Kryptolan	[PXL1]
66	RVD	MIT Remote Virtual Disk Protocol	[MBG]
67	IPPC	Internet Pluribus Packet Core	[SHB]
68		any distributed file system	[IANA]
69	SAT-MON	SATNET Monitoring	[SHB]
70	VISA	VISA Protocol	[GXT1]
71	IPCV	Internet Packet Core Utility	[SHB]
72	CPNX	Computer Protocol Network Executive	[DXM2]
73	CPHB	Computer Protocol Heart Beat	[DXM2]
74	WSN	Wang Span Network	[VXD]
75	PVP	Packet Video Protocol	[SC3]
76	BR-SAT-MON	Backroom SATNET Monitoring	[SHB]
77	SUN-ND	SUN ND PROTOCOL-Temporary	[WM3]
78	WB-MON	WIDEBAND Monitoring	[SHB]
79	WB-EXPAK	WIDEBAND EXPAK	[SHB]
80	ISO-IP	ISO Internet Protocol	[MTR]
81	VMTP	VMTP	[DRC3]
82	SECURE-VMTP	SECURE-VMTP	[DRC3]
83	VINES	VINES	[BXH]
84	TTP	TTP	[JXS]
85	NSFNET-IGP	NSFNET-IGP	[HWB]
86	DGP	Dissimilar Gateway Protocol	[DGP,ML109]
87	TCF	TCF	[GAL5]
88	EIGRP	EIGRP	[CISCO,GXS]
89	OSPFIGP	OSPFIGP	[RFC1583,JTM4]
90	Sprite-RPC	Sprite RPC Protocol	[SPRITE,BXW]
91	LARP	Locus Address Resolution Protocol	[BXH]

92	MTP	Multicast Transport Protocol	[SXA]
93	AX.25	AX.25 Frames	[BK29]
94	IPIP	IP-within-IP Encapsulation Protocol	[JI6]
95	MICP	Mobile Internetworking Control Pro.	[JI6]
96	SCC-SP	Semaphore Communications Sec. Pro.	[HXH]
97	ETHERIP	Ethernet-within-IP Encapsulation	[RXH1]
98	ENCAP	Encapsulation Header	[RFC1241,RXB3]
99		any private encryption scheme	[IANA]
100	GMTP	GMTP	[RXB5]
101	IFMP	Ipsilon Flow Management Protocol	[Hinden]
102	PNNI	PNNI over IP	[Callon]
103	PIM	Protocol Independent Multicast	[Farinacci]
104	ARIS	ARIS	[Feldman]
105	SCPS	SCPS	[Durst]
106	QNX	QNX	[Hunter]
107	A/N	Active Networks	[Braden]
108	IPComp	IP Payload Compression Protocol	[RFC2393]
109	SNP	Sitara Networks Protocol	[Sridhar]
110	Compaq-Peer	Compaq Peer Protocol	[Volpe]
111	IPX-in-IP	IPX in IP	[Lee]
112	VRRP	Virtual Router Redundancy Protocol	[Hinden]
113	PGM	PGM Reliable Transport Protocol	[Speakman]
114		any 0-hop protocol	[IANA]
115	L2TP	Layer Two Tunneling Protocol	[Aboba]
116	DDX	D-II Data Exchange (DDX)	[Worley]
117	IATP	Interactive Agent Transfer Protocol	[Murphy]
118	STP	Schedule Transfer Protocol	[JMP]
119	SRP	SpectraLink Radio Protocol	[Hamilton]
120	UTI	UTI	[Lothberg]
121	SMP	Simple Message Protocol	[Ekblad]
122	SM	SM	[Crowcroft]
123	PTP	Performance Transparency Protocol	[Welzl]
124	ISIS over IPv4		[Przygienda]
125	FIRE		[Partridge]
126	CRTP	Combat Radio Transport Protocol	[Sautter]
127	CRUDP	Combat Radio User Datagram	[Sautter]
128	SSCOPMCE		[Waber]
129	IPLT		[Hollbach]
130	SPS	Secure Packet Shield	[McIntosh]
131	PIPE	Private IP Encapsulation within IP	[Petri]
132	SCTP	Stream Control Transmission Protocol	[Stewart]
133	FC	Fibre Channel	[Rajagopal]
134-254		Unassigned	[IANA]
255		Reserved	[IANA]

Appendix II

IP Quiz

True/False

1. IP is a connectionless protocol. ______
2. IP Packets can not be fragmented. ______
3. IP is very platform specific and as a result can only be used on a few selected platforms. ______
4. The IP Identification number is used to track all packets. ______
5. The most widely used version of IP is version 9. ______

Use the following packet dump to answer the questions that follow:

```
00:00:07.720057 our.machine.com.1451 > ns2.webtrends.com.domain:  47673
A? statse.webtrendslive.com. (42) (ttl 63, id 59480)
0x0000   4500 0046 e858 0000 3f11 a2f1 ab31 1248
0x0010   3f58 d40b 05ab 0035 0032 2dc1 ba39 0000
0x0020   0001 0000 0000 0000 0673 7461 7473 650d
0x0030   7765 6274 7265 6e64 736c 6976 6503 636f
0x0040   6d00 0001 0001
```

6. This packet is IP version ________.
7. The header in this packet is _______ bytes long.
8. The total packet length is ____________.
9. The IP Identification number of this packet is _______________ in hexadecimal.
10. I know that fragmentation ___________(is or is not) permitted in this packet because byte # ___________ contains a value of _________.
11. The source address for this packet is ________________________ in hexadecimal.
12. This is a(n) _______________ packet, based upon the protocol value of ____________ in byte # ____________.

Appendix III

(True/False)

1. Most implementations of UDP include code to maintain State. _____
2. The UDP protocol guarantees delivery of a packet. _____
3. A UDP length of 6 is valid. _____
4. The Source Port defines which port the requesting machine wants the server to answer from. _____
5. UDP is IP Protocol number 6 _____

Using the following packet decode, please answer the remaining questions:

```
0x0000   4500 0054 27a5 4000 fd11 0ba0 ba21 301c
0x0010   1231 2544 e74f 006f 0040 01fb 3a2b 83d4
0x0020   0000 0000 0000 0002 0001 86a0 0000 0002
0x0030   0000 0003 0000
```

6. The Source port in this packet is ____________________ hexadecimal.
7. The Destination port in this packet is __________________ hexadecimal.
8. The UDP length of this packet is ____________ and the total length of the packet is ________________.
9. The UDP Header in this packet is __________ bytes long.

Appendix IV

Type	*Code*	*Description*
0	0	Echo Reply
3	?	Destination Unreachable messages
3	0	Network Unreachable
3	1	Host Unreachable
3	2	Protocol Unreachable
3	3	Port Unreachable
3	4	Fragmentation Required, DF set
3	5	Source Route Failed
3	6	Destination Network Unknown
3	7	Destination Host Unknown
3	8	Source Host Isolated (Obsolete)
3	9	Destination network Administratively prohibited
3	10	Destination Host Administratively prohibited
3	11	Network Unreachable for TOS
3	12	Host Unreachable for TOS
3	13	Communication Administratively prohibited by filtering
3	14	Host Precedence Violation
3	15	Precedence cutoff in effect
4	0	Source Quench
5	?	Redirects
5	0	Redirect for network
5	1	Redirect for Host
5	2	Redirect for TOS and network
5	3	Redirect for TOS and host
8	0	Echo Request
9	0	Router Advertisement
10	0	Router Solicitation
11	?	Time Exceeded Messages
11	0	TTL equals 0 during transit (TIMEX)
11	1	TTL equals 0 during reassembly
12	?	Parameter Problems
12	0	Bad IP Header
12	1	Required option missing
13	0	Timestamp Request

14	0	Timestamp Reply
15	0	Information Request (Obsolete)
16	0	Information Reply (obsolete)
17	0	Address Mask Request
18	0	Address Mask Reply

Appendix V

(True/False)

1. TCP is a connection based protocol. _____
2. The sequence numbers are used to count how many packets have passed in a given session. _____
3. The window size is used to measure the size of the data payload in a particular packet. _____
4. The Source and Destination port fields are only valid when we have a UDP header, not a TCP header. _____
5. The Urgent pointer is used to mark where the important data is inside of the data payload. _____
6. The minimum size of a TCP header is 20 bytes. _____
7. The minimum size of an IP Packet with a TCP payload is 40 bytes. _____

Please find all of the parts of a three way handshake in the following traffic:

```
00:00:04.327216 209.132.99.156.80 > firewall.62374: . 1167810001:1167810001(0) ack 400064884 win 31744
00:00:04.329386 209.132.99.156.80 > firewall.62374: P 1167810001:1167810231(230) ack 400064884 win 32120
00:00:04.329420 209.132.99.156.80 > firewall.62374: F 1167810231:1167810231(0) ack 400064884 win 32120
00:00:04.332046 firewall.62374 > 209.132.99.156.80: . 400064884:400064884(0) ack 1167810232 win 8530
00:00:04.333807 firewall.62374 > 209.132.99.156.80: F 400064884:400064884(0) ack 1167810232 win 8530
00:00:04.459872 209.132.99.156.80 > firewall.62374: . 1167810232:1167810232(0) ack 400064885 win 32120
00:00:05.221791 firewall.62367 > 209.66.74.83.80: P 1764518752:1764519084(332) ack 2459166228 win 8008
00:00:05.284853 firewall.62357 > 209.66.74.83.80: F 1764512930:1764512930(0) ack 2454368173 win 8469
00:00:05.285292 firewall.62375 > 209.225.26.101.80: S 1764532604:1764532604(0) win 8192
00:00:05.301307 firewall.62373 > 209.66.74.83.80: P 1764527602:1764527932(330) ack 2461767203 win 8093
00:00:05.325924 209.225.26.101.80 > firewall.62375: S 2321976518:2321976518(0) ack 1764532605 win 64860
00:00:05.341830 166.49.74.123.554 > firewall.64830: P 3296019943:3296020566(623) ack 513520145 win 32736
00:00:05.417434 firewall.62375 > 209.225.26.101.80: . 1764532605:1764532605(0) ack 2321976519 win 8280
00:00:05.456758 firewall.62375 > 209.225.26.101.80: P 1764532605:1764533063(458) ack 2321976519 win 8280
00:00:05.456793 firewall.64830 > 166.49.74.123.554: . 513520145:513520145(0) ack 3296020566 win 8760
00:00:05.500435 209.225.26.101.80 > firewall.62375: . 2321976519:2321976519(0) ack 1764533063 win 64860
00:00:05.544696 166.49.74.123.554 > firewall.64830: P 3296020566:3296021180(614) ack 513520145 win 32736
00:00:05.656729 firewall.64830 > 166.49.74.123.554: . 513520145:513520145(0) ack 3296021180 win 8146
00:00:05.663384 209.66.74.83.80 > firewall.62367: P 2459166228:2459166951(723) ack 1764519084 win 16534
00:00:05.683287 216.223.50.133.80 > firewall.62493: P 1446792753:1446792754(1) ack 496627 win 8424
```

Please use the following hex dump to answer the next set of questions:

```
0x0000   4500 05dc 0cc7 0000 f606 a7e1 cca2 60d7
0x0010   da21 4218 0050 f428 bfd6 f97d 011d 15cf
0x0020   5018 faf0 bd64 0000 4854 5450 2f31 2e30
0x0030   2032 3030 204f 4b0d 0a4d 494d 452d 5665
0x0040   7273 696f 6e3a 2031 2e30 0d0a 4461 7465
0x0050   3a20 4d6f 6e2c 2030 3420 4465 6320 3230
0x0060   3030 2030 373a 3531 3a31 3120 474d 540d
0x0070   0a43
```

8. Based on the value, ________, in byte # ________ of the IP Header, this is a _________ *(TCP/UDP/ICMP)* packet.
9. The IP Header in this example is ___________ bytes long.
10. The TCP Header in this example is ___________ bytes long.

11. The window size in this particular packet is __________________ as indicated by the hexadecimal value _______________ in byte position ________________ of the ____________ header.
12. The Code Bits in this packet indicate that the following bits are turned on:_______________________
13. The total amount of data in this packet is _____________________ bytes.
14. I determined the amount of data in this packet by ____ ____ ____
15. The sequence number in this packet is ______________________ hexadecimal.
16. The acknowledgement number in this packet is _______________________ hexadecimal.
17. Is the acknowledgement number in this packet valid?
18. The source port in this packet is ____________ and the destination port is ___________.
19. Are there options in this packet?___________ How do you know?____