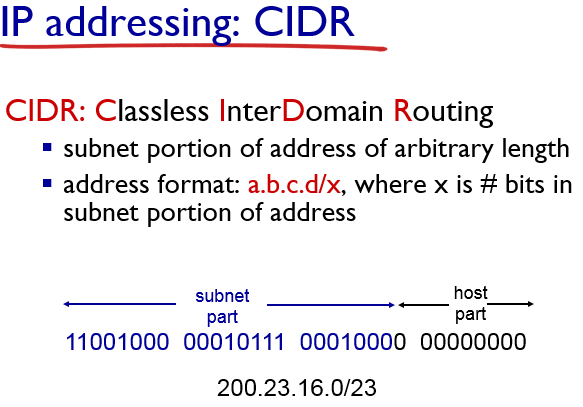
**NETWORK LAYER:**

IPv4:

IPv4 is a connectionless protocol for use on [packet-switched](http://en.wikipedia.org/wiki/Packet-switched) networks. It operates on a [best effort delivery](http://en.wikipedia.org/wiki/Best_effort_delivery) model, in that it does not guarantee delivery, nor does it assure proper sequencing or avoidance of duplicate delivery. These aspects, including data integrity, are addressed by an [upper layer](http://en.wikipedia.org/wiki/Upper_layer_protocol) transport protocol, such as the [Transmission Control Protocol](http://en.wikipedia.org/wiki/Transmission_Control_Protocol) (TCP).

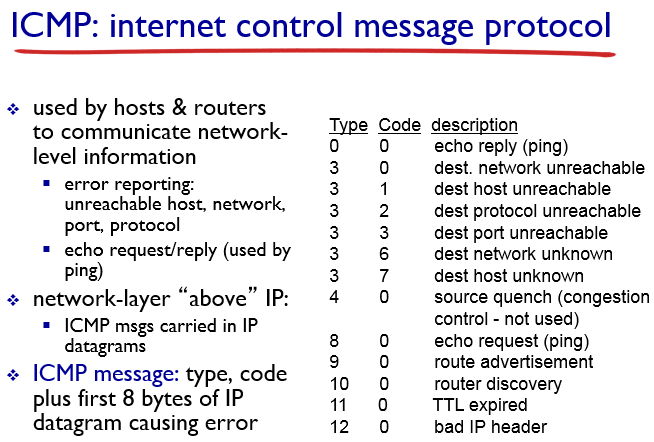
|  |  |  |
| --- | --- | --- |
| **Table 56: Internet Protocol Version 4 (IPv4) Datagram Format** | | |
| **Field Name** | **Size (bytes)** | **Description** |
| ***Version*** | 1/2 (4 bits) | ***Version:*** Identifies the version of IP used to generate the datagram. For IPv4, this is of course the number 4. The purpose of this field is to ensure compatibility between devices that may be running different versions of IP. In general, a device running an older version of IP will reject datagrams created by newer implementations, under the assumption that the older version may not be able to interpret the newer datagram correctly. |
| ***IHL*** | 1/2 (4 bits) | ***Internet Header Length (IHL):*** Specifies the length of the IP header, in 32-bit words. This includes the length of any options fields and padding. The normal value of this field when no options are used is 5 (5 32-bit words = 5\*4 = 20 bytes). Contrast to the longer *Total Length* field below. |
| ***TOS*** | 1 | ***Type Of Service (TOS):*** A field designed to carry information to provide quality of service features, such as prioritized delivery, for IP datagrams. It was never widely used as originally defined, and its meaning has been subsequently redefined for use by a technique called *Differentiated Services (DS)*. See below for more information. |
| ***TL*** | 2 | ***Total Length (TL):*** Specifies the total length of the IP datagram, in bytes. Since this field is 16 bits wide, the maximum length of an IP datagram is 65,535 bytes, though most are much smaller. |
| ***Identification*** | 2 | ***Identification:*** This field contains a 16-bit value that is common to each of the fragments belonging to a particular message; for datagrams originally sent unfragmented it is still filled in, so it can be used if the datagram must be fragmented by a router during delivery. This field is used by the recipient to reassemble messages without accidentally mixing fragments from different messages. This is needed because fragments may arrive from multiple messages mixed together, since IP datagrams can be received out of order from any device. [See the discussion of IP message fragmentation.](http://www.tcpipguide.com/free/t_IPMessageFragmentationProcess.htm) |
| ***Flags*** | 3/8 (3 bits) | http://www.tcpipguide.com/free/aa1f0152.png |
| ***Fragment Offset*** | 1 5/8 (13 bits) | ***Fragment Offset:*** When fragmentation of a message occurs, this field specifies the offset, or position, in the overall message where the data in this fragment goes. It is specified in units of 8 bytes (64 bits). The first fragment has an offset of 0. Again, [see the discussion of fragmentation](http://www.tcpipguide.com/free/t_IPMessageFragmentationProcess.htm) for a description of how the field is used. |
| ***TTL*** | 1 | ***Time To Live (TTL):*** Short version: Specifies how long the datagram is allowed to “live” on the network, in terms of router hops. Each router decrements the value of the TTL field (reduces it by one) prior to transmitting it. If the TTL field drops to zero, the datagram is assumed to have taken too long a route and is discarded.  See below for the longer explanation of *TTL*. |
| ***Protocol*** | 1 | http://www.tcpipguide.com/free/aa1f02ce.png |
| ***Header Checksum*** | 2 | ***Header Checksum:*** A checksum computed over the header to provide basic protection against corruption in transmission. This is not the more complex CRC code typically used by data link layer technologies such as Ethernet; it's just a 16-bit checksum. It is calculated by dividing the header bytes into words (a word is two bytes) and then adding them together. The data is not checksummed, only the header. At each hop the device receiving the datagram does the same checksum calculation and on a mismatch, discards the datagram as damaged. |
| ***Source Address*** | 4 | ***Source Address:*** The 32-bit IP address of the originator of the datagram. Note that even though intermediate devices such as routers may handle the datagram, they do not normally put their address into this field—it is always the device that originally sent the datagram. |
| ***Destination Address*** | 4 | ***Destination Address:*** The 32-bit IP address of the intended recipient of the datagram. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination. |
| ***Options*** | Variable | ***Options:*** One or more of several types of options may be included after the standard headers in certain IP datagrams. I discuss them in [the topic that follows this one](http://www.tcpipguide.com/free/t_IPDatagramOptionsandOptionFormat.htm). |
| ***Padding*** | Variable | ***Padding:*** If one or more options are included, and the number of bits used for them is not a multiple of 32, enough zero bits are added to “pad out” the header to a multiple of 32 bits (4 bytes). |
| ***Data*** | Variable | ***Data:*** The data to be transmitted in the datagram, either an entire higher-layer message or a fragment of one. |

|  |
| --- |
| http://www.tcpipguide.com/free/diagrams/ipformat.png |
| **Figure 86: Internet Protocol Version 4 (IPv4) Datagram Format**  This diagram shows graphically the all-important IPv4 datagram format. The first 20 bytes are the fixed IP header, followed by an optional *Options* section, and a variable-length *Data* area. Note that the *Type Of Service* field is shown as originally defined in the IPv4 standard.  IPv6:   |  |  |  | | --- | --- | --- | | **Table 68: IPv6 Main Header Format** | | | | **Field Name** | **Size (bytes)** | **Description** | | ***Version*** | 1/2 (4 bits) | ***Version:*** Identifies the version of IP used to generate the datagram. This field is used the same way as in IPv4, except of course that it carries the value 6 (0110 [binary](http://www.tcpipguide.com/free/t_IPv6DatagramMainHeaderFormat.htm)). | | ***Traffic Class*** | 1 | ***Traffic Class:*** This field replaces the *Type Of Service (TOS)* field in the IPv4 header. It is used not in the original way that the TOS field was defined (with Precedence, D, T and R bits) but using the new *Differentiated Services (DS)* method defined in RFC 2474. That RFC actually specifies quality of service (QOS) techniques for both IPv4 and IPv6; [see the IPv4 format description for a bit more information](http://www.tcpipguide.com/free/t_IPDatagramGeneralFormat.htm). | | ***Flow Label*** | 2 1/2 (20 bits) | ***Flow Label:*** This large field was created to provide additional support for real-time datagram delivery and quality of service features. The concept of a *flow* is defined in RFC 2460 as a sequence of datagrams sent from a source device to one or more destination devices. A unique flow label is used to identify all the datagrams in a particular flow, so that routers between the source and destination all handle them the same way, to help [ensure](http://www.tcpipguide.com/free/t_IPv6DatagramMainHeaderFormat.htm) uniformity in how the datagrams in the flow are delivered. For example, if a video stream is being sent across an IP internetwork, the datagrams containing the stream could be identified with a flow label to ensure that they are delivered with minimal latency.  Not all devices and routers may support flow label handling, and use of the field by a source device is entirely optional. Also, the field is still somewhat experimental and may be refined over time. | | ***Payload Length*** | 2 | ***Payload Length:*** This field replaces the *Total Length* field from the IPv4 header, but it is used differently. Rather than measuring the length of the whole datagram, it only contains the number of bytes of the payload. However, if extension headers are included, their length is counted here as well.  In simpler terms, this field measures the length of the datagram less the 40 bytes of the main header itself. | | ***Next Header*** | 1 | ***Next Header:*** This field replaces the *Protocol* field and has two uses. When a datagram has extension headers, this field specifies the identity of the first extension header, which is the next header in the datagram. When a datagram has just this “main” header and no extension headers, it serves the same purpose as the old IPv4 *Protocol* field and has the same values, though new numbers are used for IPv6 versions of common protocols. In this case the “next header” is the header of the upper layer message the IPv6 datagram is carrying. See below for more details. | | ***Hop Limit*** | 1 | ***Hop Limit:*** This replaces the *Time To Live (TTL)* field in the IPv4 header; its name better reflects the way that *TTL* is used in modern networks (since *TTL* is really used to count hops, not time.) | | ***Source Address*** | 16 | ***Source Address:*** The 128-bit [IP address](http://www.tcpipguide.com/free/t_IPv6DatagramMainHeaderFormat.htm) of the originator of the datagram. As with IPv4, this is always the device that originally sent the datagram. | | ***Destination Address*** | 16 | ***Destination Address:*** The 128-bit IP address of the intended recipient of the datagram; unicast, anycast or multicast. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination. |      |  | | --- | | http://www.tcpipguide.com/free/diagrams/ipv6format.png | | **Figure 105: IPv6 Main Header Format** |   The IP service model is a **best-effort delivery service**. This means that IP makes its "best effort" to deliver segments between communicating hosts, but **it makes no guarantees**. In particular, it does not guarantee segment delivery, it does not guarantee orderly delivery of segments, and it does not guarantee the integrity of the data in the segments. For these reasons, IP is said to be an **unreliable service**. We also mention here that every host has an IP address. We need to keep in mind that each host has a *unique* IP address.  Comparison with IPv4:  Larger address space.  Multicasting: [Multicasting](http://en.wikipedia.org/wiki/Multicast), the transmission of a packet to multiple destinations in a single send operation, is part of the base specification in IPv6.  Stateless address autoconfiguration (SLAAC) |
|  |
|  |

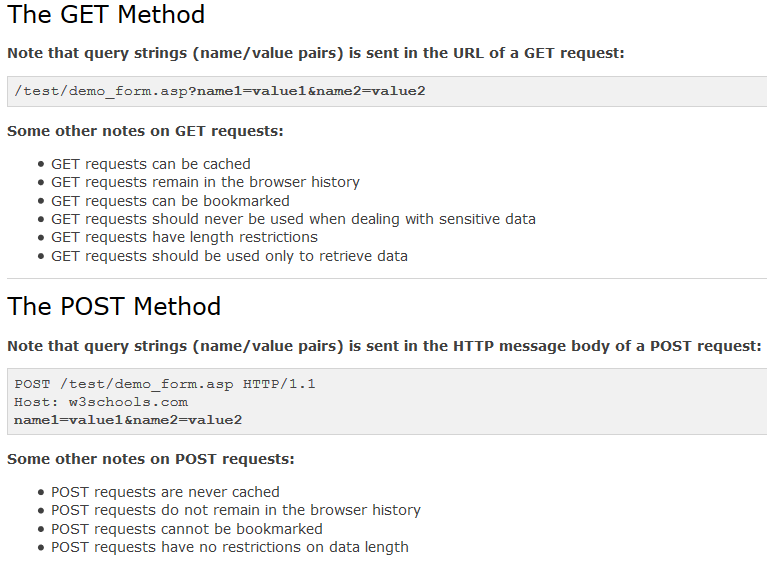


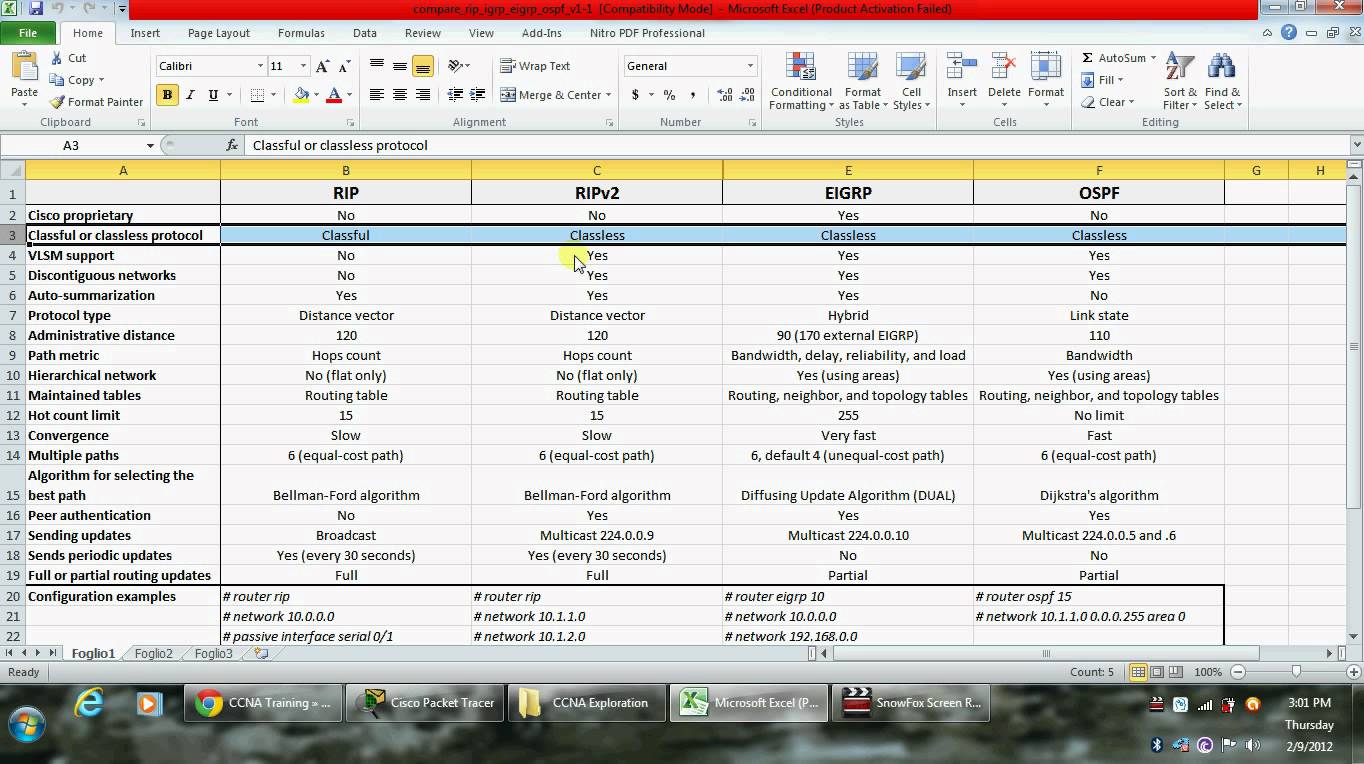
* *what’s a subnet ?*
  + device interfaces with same subnet part of IP address
  + can physically reach each other *without intervening router*

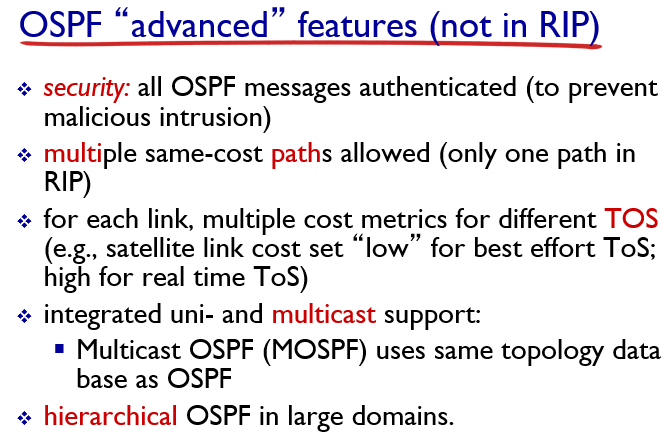
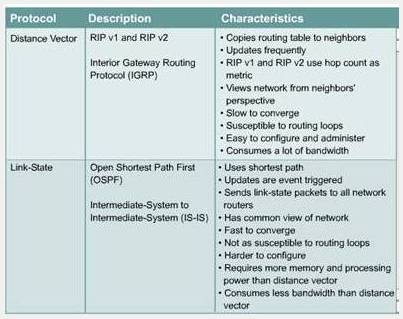
The **Internet Control Message Protocol** *(****ICMP****)* is one of the main protocols of the [Internet Protocol Suite](http://en.wikipedia.org/wiki/Internet_Protocol_Suite). It is used by network devices, like routers, to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached. ICMP can also be used to relay query messages.[[1]](http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#cite_note-Forouzan-1) It is assigned protocol number 1.[[2]](http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#cite_note-2) ICMP[[3]](http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#cite_note-rfc792-3) differs from transport protocols such as [TCP](http://en.wikipedia.org/wiki/Transmission_Control_Protocol) and [UDP](http://en.wikipedia.org/wiki/User_Datagram_Protocol) in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the exception of some diagnostic tools like [ping](http://en.wikipedia.org/wiki/Ping_(networking_utility)) and [traceroute](http://en.wikipedia.org/wiki/Traceroute)).



The **Dynamic Host Configuration Protocol** (**DHCP**) is a [standardized](http://en.wikipedia.org/wiki/Standardized) networking protocol used on [Internet Protocol](http://en.wikipedia.org/wiki/Internet_Protocol) (IP) networks for dynamically distributing network configuration parameters, such as [IP addresses](http://en.wikipedia.org/wiki/IP_address) for interfaces and services. With DHCP, computers request IP addresses and networking parameters automatically from a [DHCP server](http://en.wikipedia.org/wiki/DHCP_server), reducing the need for a [network administrator](http://en.wikipedia.org/wiki/Network_administrator) or a user to configure these settings manually.

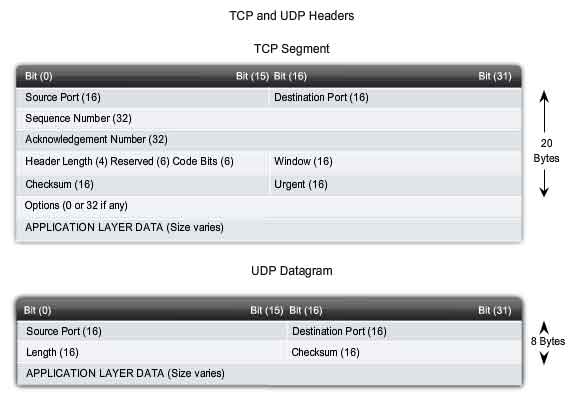


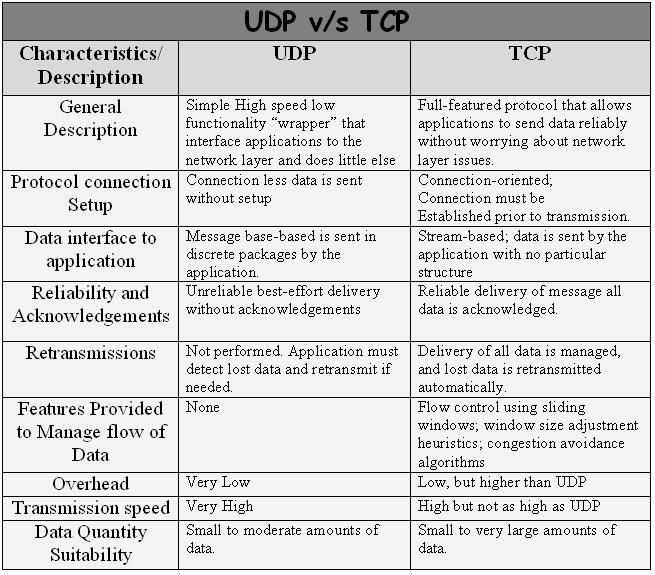


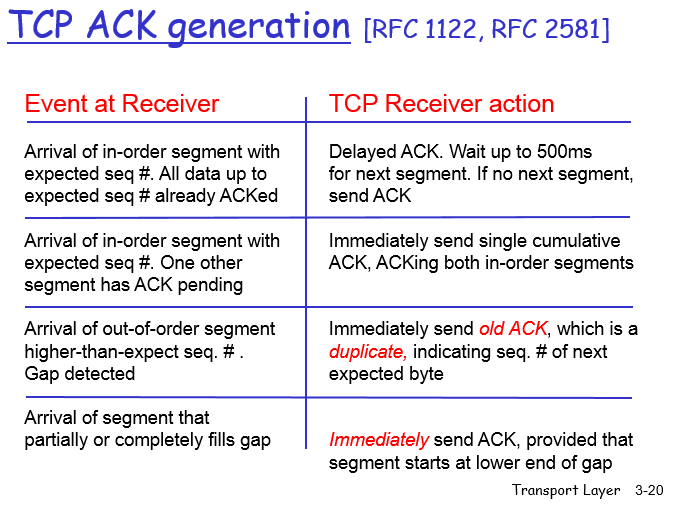


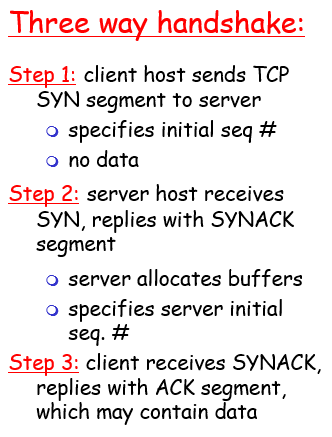
**Comparison between TCP and UDP**

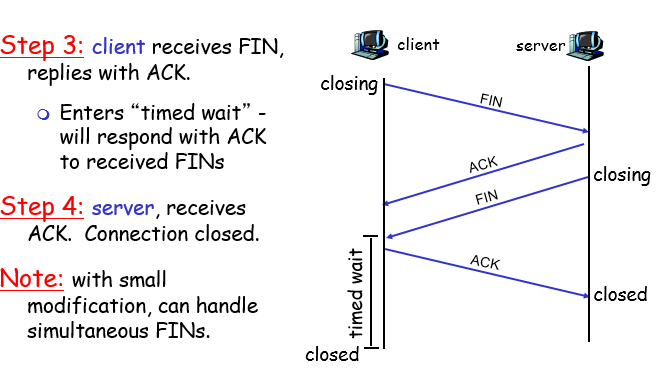
|  |  |
| --- | --- |
| **TCP** | **UDP** |
| Connection-oriented | Connectionless |
| Reliable packet delivery | Unreliable packet delivery |
| Has flow control mechanism | No flow control mechanism |
| Not preferred for real time applications | Preferred for real time applications |
| Has considerable overhead | Lightweight protocol with less overhead |
| Application: FTP, HTTP etc | Application: DNS, DNS |

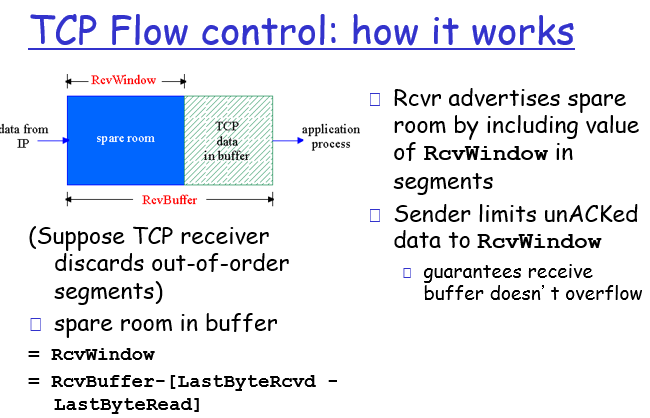


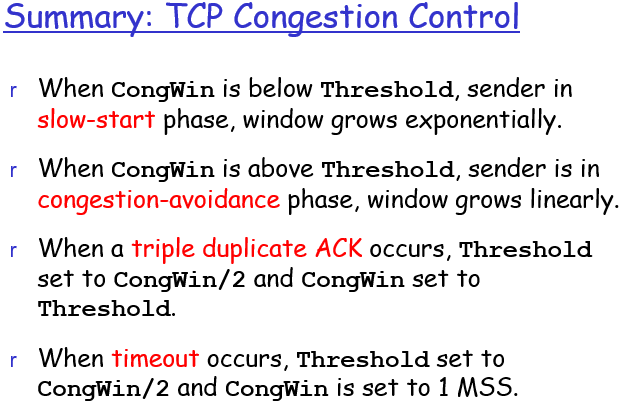










+

**TCP Checksum Calculation and the TCP "Pseudo Header"**   
(Page 2 of 3)

***Increasing The Scope of Detected Errors: the TCP Pseudo Header***

To this end, a change was made in how the TCP checksum is computed. This special TCP checksum algorithm was eventually also [adopted for use by the User Datagram Protocol (UDP)](http://www.tcpipguide.com/free/t_UDPMessageFormat.htm).

Instead of [computing](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm) the checksum over only the actual [data fields](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm) of the TCP segment, a 12-byte TCP *pseudo header* is created prior to checksum calculation. This header [contains](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm) important information taken from fields in both the TCP header and the [IP datagram](http://www.tcpipguide.com/free/t_IPDatagramGeneralFormat.htm) into which the TCP segment will be encapsulated. The TCP pseudo header has [the format](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm) shown in [Table 158](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm#Table_158) and [Figure 217](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm#Figure_217).

|  |  |  |
| --- | --- | --- |
| **Table 158: TCP “Pseudo Header” For Checksum Calculation** | | |
| **Field Name** | **Size (bytes)** | **Description** |
| ***Source Address*** | 4 | ***Source Address:*** The 32-bit IP address of the originator of the datagram, taken from the IP header. |
| ***Destination Address*** | 4 | ***Destination Address:*** The 32-bit IP address of the intended recipient of the datagram, also from the IP header. |
| ***Reserved*** | 1 | ***Reserved:*** 8 bits of zeroes. |
| ***Protocol*** | 1 | ***Protocol:*** The *Protocol* field from the IP header. This indicates what higher-layer protocol is carried in the IP datagram. Of [course](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm), we already know what this protocol is, it's TCP! So, this field will normally have the value 6. |
| ***TCP Length*** | 2 | ***TCP Length:*** The length of the TCP segment, including both header and data. Note that this is not a specific field in the TCP header; it is computed. |

|  |
| --- |
| http://www.tcpipguide.com/free/diagrams/tcppseudoheader.png |
| **Figure 217: TCP “Pseudo Header” For Checksum Calculation** |

Once this 96-bit header has been formed, it is placed in a buffer, following which the TCP segment itself is placed. Then, the checksum is computed over the entire set of data (pseudo header plus TCP segment). The value of the checksum is placed into the *Checksum* field of the TCP header, and the pseudo header is discarded—it is ***not*** an actual part of the TCP segment and is not transmitted. This process is illustrated in [Figure 218](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm#Figure_218).

|  |
| --- |
| **Note:** The *Checksum* field is itself part of the TCP header and thus one of the fields over which the checksum is calculated, creating a “chicken and egg” situation of sorts. This field is assumed to be all zeroes during calculation of the checksum. |

|  |
| --- |
| http://www.tcpipguide.com/free/diagrams/tcppseudocalc.png |
| **Figure 218: TCP Header Checksum Calculation**  To calculate the TCP segment header’s Checksum field, the TCP pseudo header is first constructed and placed, logically, before the TCP segment. The checksum is then calculated over both the pseudo header and the TCP segment. The pseudo header is then discarded. |

When the TCP segment arrives at its destination, the receiving TCP software performs the same calculation. It forms the pseudo header, prepends it to the actual TCP segment, and then performs the checksum (setting the *Checksum* field to zero for the calculation as before). If there is a mismatch between its calculation and the value the source device put in the *Checksum* field, this indicates that an error of some sort occurred and the segment is normally discarded.

**The Checksum Field and the UDP Pseudo Header**

The [UDP](http://www.tcpipguide.com/free/t_UDPMessageFormat-2.htm) *Checksum* field is the one area where the protocol actually is a bit confusing. The concept of a checksum itself is nothing new; they are used widely in [networking](http://www.tcpipguide.com/free/t_UDPMessageFormat-2.htm) protocols to provide protection against errors. What's a bit odd is this notion of [computing](http://www.tcpipguide.com/free/t_UDPMessageFormat-2.htm) the checksum over the regular datagram and also a *pseudo header*. What this means is that instead of calculating the checksum over just the fields in the UDP datagram itself, the UDP [software](http://www.tcpipguide.com/free/t_UDPMessageFormat-2.htm) first constructs a “fake” additional header that contains the following fields ([Figure 201](http://www.tcpipguide.com/free/t_UDPMessageFormat-2.htm#Figure_201)):

* The [IP](http://www.tcpipguide.com/free/t_UDPMessageFormat-2.htm) *Source Address* field.
* The IP *Destination Address* field.
* The IP *Protocol* field.
* The UDP *Length* field.

|  |
| --- |
| http://www.tcpipguide.com/free/diagrams/udppseudoheader.png |
| **Figure 201: UDP Pseudo Header Format** |

The total length of this “pseudo header” is 11 bytes. It is padded to 12 bytes with a byte of zeroes and then prepended to the real UDP message. The checksum is then computed over the combination of the pseudo header and the real UDP message, and the value is placed into the *Checksum* field. The pseudo header is used only for this calculation and is then discarded; it is not actually transmitted. The UDP software in the destination [device](http://www.tcpipguide.com/free/t_UDPMessageFormat-2.htm) creates the same pseudo header when calculating its checksum to compare to the one transmitted in the UDP header.

Computing the checksum over the regular UDP fields protects against bit errors in the UDP message itself. Adding the pseudo header allows the checksum to also protect against other types of problems as well, most notably the accidental delivery of a message to the wrong destination. The checksum calculation in UDP, including the use of the pseudo header is exactly the same as the method used in TCP (except the *Length* field is different in TCP). [See the topic describing TCP checksum calculation](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader.htm) for a full description of why the pseudo header is important, and some of the interesting implications of using IP fields in transport layer datagram calculations.

|  |
| --- |
| **Key Concept:** UDP packages [application](http://www.tcpipguide.com/free/t_UDPMessageFormat-2.htm) layer data into a very simple message format that includes only four header fields. One of these is an optional checksum field; when used, the checksum is computed over both the real header and a “pseudo header” of fields from the UDP and IP headers, in a manner very similar to how the TCP checksum is calculated. |

Note that the use of the *Checksum* field is optional in UDP. If it is not used, it is set to a value of all zeroes. This could potentially create confusion, however, since when the checksum ***is*** used, the calculation can sometimes result in a value of zero. To avoid having the destination think the checksum was not used in this case, this zero value is instead represented as a value of all ones (65,535 decimal).

**Maximum Window Size**   
  
Normally, two types of error control are used in protocols: [**Go-Back-N** and **Selective Repeat**](http://webmuseum.mi.fh-offenburg.de/index.php?view=exh&src=30). Additionally for flow control purposes, sender and receiver maintain a window of acceptable sequence numbers. Let us now have a look at the **maximum window size** in Go-Back-N and Selective Repeat protocols. In the scenario, a 2-bit sequence number is used with the numbers 0, 1, 2 and 3 (in general, a n-bit number ranging from 0�2n-1).  
  
What is the maximum window size? The receiver must be able to distinguish a retransmission of an already received packet from the original transmission of a new packet. Thus the maximum window size is:

* 2n-1 in the case of Go-Back-N. Here the receiver accepts only the next expected packet and discards all out-of-order packets. In the example, with a 2-bit sequence number the maximum window size is 3

2n/2 in Selective Repeat. Since the receiver accepts out-of-order packets, two consecutive windows should not overlap. Otherwise it is not able to distinguish duplicates from new transmissions. Hence, in the example the maximum window size is 2

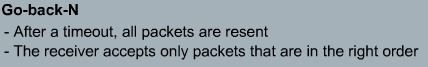
Protocols that provide reliability have to include several functions:

* error detection: detect bit errors, packet loss or duplication.
* acknowledgements ([ACK](javascript:popup(13);)): feedback from the receiver to the sender. A cumulative ACK lets the sender know that several sequential packets are received correctly. In contrast a selective ACK acknowledges just one individual packet.
* retransmissions: lost packets are automatically retransmitted from the sender after a timeout, when no ACK is received.

**RDT:**

**Stop-and Wait**  
Very simple protocol where the sender sends one [frame](javascript:popup(51);) and then waits for the ACK before proceeding.

**GO-Back-N**   
If a timeout occurs the sender resends all packets that have been sent, but not yet been acknowledged.



**Selective Repeat**

Selective Repeat is part of the automatic repeat-request (ARQ). With selective repeat, the sender sends a number of frames specified by a window size even without the need to wait for individual ACK from the receiver as in Go-back N ARQ. However, the receiver sends ACK for each frame individually, which is not like cumulative ACK as used with go-back-n. The receiver accepts out-of-order frames and buffers them. The sender individually retransmits frames that have timed out.

The sender retransmits only those packets that it suspects were lost. This individual retransmissions require that the receiver individually acknowledges correctly received packets.

There are three main **differences** between **HTTP** and **SMTP**:   
  
1) **HTTP** is mainly a **pull** protocol--someone loads information on a web server and users use HTTP to pull the information from the server. On the other hand, **SMTP** is primarily a **push** protocol--the sending mail server pushes the file to receiving mail server.   
  
2) **SMTP** **requires** each **message**, including the body of each message, to be in **seven-bit ASCII format**. If the message contains binary data, then the message has to be encoded into seven-bit ASCII format. **HTTP** does **not** have this restriction.   
  
3) **HTTP** encapsulate **each** **object** of message in its **own response message** while **SMTP** places **all** of the message's **objects** **into one message**.

