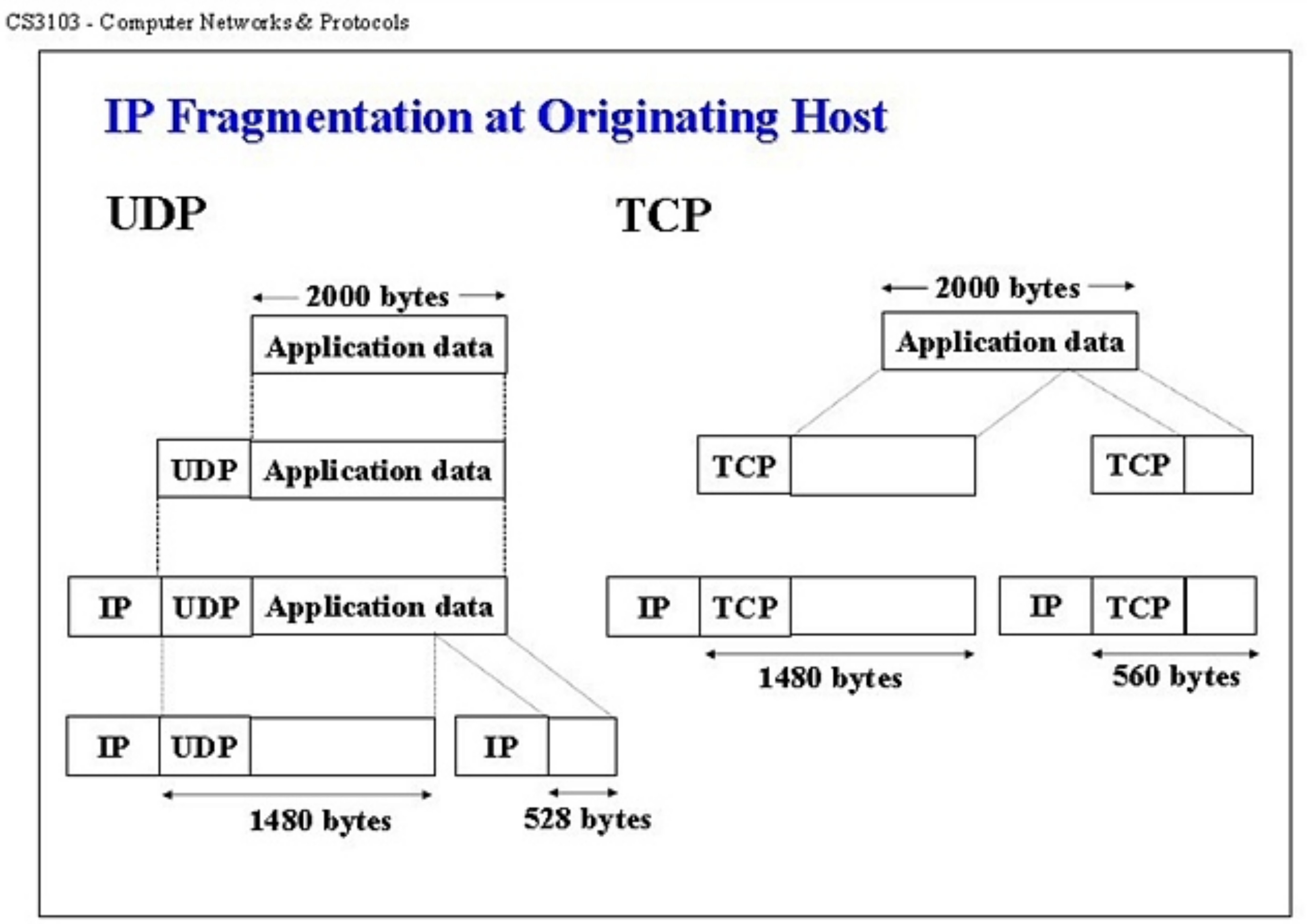
IP分片、TCP分段、IP分片中的UDP首部_Robin's blog_ousys的和讯博客

为什么会有IP分片？直接原因是上层协议企图发送一段数据，其长度超过了MTU（Maxitum Transmission Unit）。什么情况，或者说什么协议会尝试发送这么长的数据？常见的有UDP和ICMP，需要特别注意的是，TCP一般不会。

为什么TCP不会造成IP分片呢？原因是TCP自身支持分段：当TCP要传输长度超过MSS（Maxitum Segment Size）的数据时，会先对数据进行分段，正常情况下，MSS小于MTU，因此，TCP一般不会造成IP分片。

而UDP和ICMP就不支持这种分段功能了，UDP和ICMP认为网络层可以传输无限长（实际上有65535的限制）的数据，当这两种协议发送数据时，它们不考虑数据长度，仅在其头部添加UDP或ICMP首部，然后直接交给网络层就万事大吉了。接着网络层IP协议对这种“身长头短”的数据进行分片，不要指望IP能很“智能”地识别传给它的数据上层头部在哪里，载荷又在哪里，它会直接将整个的数据切成N个分片，这样做的结果是，只有第一个分片具有UDP或者ICMP首部，而其它分片则没有。

于是又更进一步理解TCP存在的必要性了！



造成差别的原因上面说的很清楚， tcp 有segmentation, 每次收到切片都会ack , udp 不管这些。

下面的一段来自cisco

http://www.cisco.com/en/US/tech/tk827/tk369/technologies_white_paper09186a00800d6979.shtml

讲的主要是关于TCP MSS 的

Issues with IP Fragmentation

There are several issues that make IP fragmentation undesirable. There is a small increase in CPU and memory overhead to fragment an IP datagram. This holds true for the sender as well as for a router in the path between a sender and a receiver. Creating fragments simply involves creating fragment headers and copying the original datagram into the fragments. This can be done fairly efficiently because all the information needed to create the fragments is immediately available.

Fragmentation causes more overhead for the receiver when reassembling the fragments because the receiver must allocate memory for the arriving fragments and coalesce them back into one datagram after all of the fragments are received. Reassembly on a host is not considered a problem because the host has the time and memory resources to devote to this task.

But, reassembly is very inefficient on a router whose primary job is to forward packets as quickly as possible. A router is not designed to hold on to packets for any length of time. Also a router doing reassembly chooses the largest buffer available (18K) with which to work because it has no way of knowing the size of the original IP packet until the last fragment is received.

Another fragmentation issue involves handling dropped fragments. If one fragment of an IP datagram is dropped, then the entire original IP datagram must be resent, and it will also be fragmented. You see an example of this with Network File System (NFS). NFS, by default, has a read and write block size of 8192, so a NFS IP/UDP datagram will be approximately 8500 bytes (including NFS, UDP, and IP headers). A sending station connected to an Ethernet (MTU 1500) will have to fragment the 8500 byte datagram into six pieces; five 1500 byte fragments and one 1100 byte fragment. If any of the six fragments is dropped because of a congested link, the complete original datagram will have to be retransmitted, which means that six more fragments will have to be created. If this link drops one in six packets, then the odds are low that any NFS data can be transferred over this link, since at least one IP fragment would be dropped from each NFS 8500 byte original IP datagram.

Firewalls that filter or manipulate packets based on Layer 4 (L4) through Layer 7 (L7) information in the packet may have trouble processing IP fragments correctly. If the IP fragments are out of order, a firewall may block the non-initial fragments because they do not carry the information that would match the packet filter. This would mean that the original IP datagram could not be reassembled by the receiving host. If the firewall is configured to allow non-initial fragments with insufficient information to properly match the filter, then a non-initial fragment attack through the firewall could occur. Also, some network devices (such as Content Switch Engines) direct packets based on L4 through L7 information, and if a packet spans multiple fragments, then the device may have trouble enforcing its policies.

Avoiding IP Fragmentation: What TCP MSS Does and How It Works

The TCP Maximum Segment Size (MSS) defines the maximum amount of data that a host is willing to accept in a single TCP/IP datagram. This TCP/IP datagram may be fragmented at the IP layer. The MSS value is sent as a TCP header option only in TCP SYN segments. Each side of a TCP connection reports its MSS value to the other side. Contrary to popular belief, the MSS value is not negotiated between hosts. The sending host is required to limit the size of data in a single TCP segment to a value less than or equal to the MSS reported by the receiving host.

Originally, MSS meant how big a buffer (greater than or equal to 65496K) was allocated on a receiving station to be able to store the TCP data contained within a single IP datagram. MSS was the maximum segment (chunk) of data that the TCP receiver was willing to accept. This TCP segment could be as large as 64K (the maximum IP datagram size) and it could be fragmented at the IP layer in order to be transmitted across the network to the receiving host. The receiving host would reassemble the IP datagram before it handed the complete TCP segment to the TCP layer.

Below are a couple of scenarios showing how MSS values are set and used to limit TCP segment sizes, and therefore, IP datagram sizes.

Scenario 1 illustrates the way MSS was first implemented. Host A has a buffer of 16K and Host B a buffer of 8K. They send and receive their MSS values and adjust their send MSS for sending data to each other. Notice that Host A and Host B will have to fragment the IP datagrams that are larger than the interface MTU but still less than the send MSS because the TCP stack could pass 16K or 8K bytes of data down the stack to IP. In Host B's case, packets could be fragmented twice, once to get onto the Token Ring LAN and again to get onto the Ethernet LAN.

TCP MTU 计算的一个例子：

A Packet Fragmentation Example

If a 2,366 byte packet （TCP）enters an Ethernet network with a default MTU size, it must be fragmented into two packets.

The first packet will:
Be 1,500 bytes in length. 20 bytes will be the IP header, 24 bytes will be the TCP header, and 1,456 bytes will be data.
Have the DF bit equal to 0 to mean "May Fragment" and the MF bit equal to 1 to mean "More Fragments."
Have a Fragmentation Offset of 0.

The second packet will:
Be 910 bytes in length. 20 bytes will be the IP header, 24 bytes will be the TCP header, and 866 bytes will be data.
Have the DF bit equal to 0 to mean "May Fragment" and the MF bit equal to 0 to mean "Last Fragment."
Have a Fragmentation Offset of 182 (Note: 182 is 1456 divided by 8).
The Packet Fragmentation Attack

Packet fragmentation can be utilized to get around blocking rules on some firewalls.

This is done by cheating with the value of the Fragment Offset. The trick is to set the value of the Fragment Offset on the second packet so low that instead of appending the second packet to the first packet, it actually overwrites the data and part of the TCP header of the first packet.

Let's say you want to `telnet` into a network where TCP port 23 is blocked by a packet filtering firewall. However, SMTP port 25 is allowed into that network.

What you would do is to send two packets:

The first packet would:
Have a Fragmentation Offset of 0.
Have the DF bit equal to 0 to mean "May Fragment" and the MF bit equal to 1 to mean "More Fragments."
Have a Destination Port in the TCP header of 25. TCP port 25 is allowed, so the firewall would allow that packet to enter the network.

The second packet would:
Have a Fragmentation Offset of 1. This means that the second packet would actually overwrite everything but the first 8 bits of the first packet.
Have the DF bit equal to 0 to mean "May Fragment" and the MF bit equal to 0 to mean "Last Fragment."
Have a Destination Port in the TCP header of 23. This would normally be blocked, but will not be in this case!

The packet filtering firewall will see that the Fragment Offset is greater than zero on the second packet. From this data, it will deduce that the second packet is a fragment of another packet and it will not check the second packet against the rule set.

When the two packets arrive at the target host, they will be reassembled. The second packet will overwrite most of the first packet and the contents of the combined packet will go to port 23.

需要注意的：

上面主要说的是从Host发送的处理，当到ROUTER 的时候， 到达的IP datagram会decap 和encap了再次发送， 可是router 只负责到network layer, 它不会区别对待UDP 和TCP,如果遇到较小的MTU , 它会把收到的IP 数据（不包括IP header)分开，然后每个分片都会有IP header(没有怎么传），

如果host端传到这里的是tcp 数据的话，显然每个ip 的分片的IP header 后面是有TCP header（但这个不是router拷贝操作的，而是host在transport layer 上做的mss 的操作）， 如果host传到这里的是UDP 数据的话，只有第一个IP 的分片的IP header会有udp header(这个是因为Host做的分片在network layer 上操作）。

TCP--- HOST 会在transport layer 上操作， MSS , 每个分片有TCP header.

UDP--- Host 在network layer 上操作，只有第一个分片有UDP header

Router--- 不管它上层协议（TCP 或UDP),始终在network layer 上操作。