

MTU是一个老概念了，是属于以太网数据链路层的概念，而MSS是新的概念，由于MTU和MSS概念都十分重要，且容易混淆，为了讨论清晰，单独拎一章节来讨论它们俩。

首先我们要说明下讨论前提，本文基于以太网协议、IP协议版本使用的是IPv4版本讨论。

概括来讲，MTU是以太网数据链路层中约定的数据载荷部分最大长度，数据不超过它时就无需分片。

而MSS是传输层的概念，由于数据往往很大，会超出MTU，所以我们之前在网络层中学习过IP分片的知识，将很大的数据载荷分割为多个分片发送出去。

而TCP为了IP层不用分片主动将数据包切割为MSS大小。一个等式可见他两关系匪浅：

$$MSS = MTU - IP \text{ header头大小} - TCP \text{ 头大小}$$

以上就是本文内容摘要，下面我们展开细说。

一、MTU复习

MTU全称是Maximum Transmission Unit，即最大传输单元。

在学习数据链路层时，我们学习过以太网协议，以太网定义了一个叫做帧的概念，一个帧中包含如下信息：

DST 地址 (接收方 MAC)	SRC 地址 (发送方 MAC)	第 3 层 使用的协议	要发送的信息	CRC
---------------------	---------------------	----------------	--------	-----

此外，我们学习了帧的大小，其中帧头大小为：

- 接收方和发送方的 MAC 地址分别占用 6 个字节；
- 第 3 层的协议用 2 个字节编码；
- CRC 用 4 个字节编码。

$6 \times 2 + 2 + 4 = 18$ 。因此以太网的帧头一共有 18 个字节。并且以太网中还规定了最小帧长和最大帧长：

- 以太网帧的最小尺寸是 64 字节，那么一帧中最少报文长度为46字节。
- 以太网帧的最大尺寸是 1518 字节，那么一帧中中最大报文长度为1500字节。

其中1500字节往往就是以太网的MTU值了，传输的数据小于它时，就无需切片。

太大的数据就需要切分，就像一个超级大包裹需要切分为若干个小包裹才方便托运。假设传输100KB的数据，则需要切分为多少个帧进行传输呢？

100KB=100*1024B，由于帧中最大的报文长度是1500B，那么
100KB/1500B≈68.27，显然需要69个以太网帧才能承载。

二、MTU与木桶效应

一台机器上，不同网卡的MTU也不一样，比如我的一台虚拟机上的网卡MTU为：

```
[root@VM-0-13-centos ~]# netstat -i
Kernel Interface table

```

Iface	MTU	RX-OK	RX-ERR	RX-DROP	RX-OVR	TX-OK	TX-ERR	TX-DROP	TX-OVR	Flg
docker0	1500	149	0	0	0	95	0	0	0	BMU
eth0	1500	210476842	0	0	0	230295272	0	0	0	BMRU
lo	65536	4842412	0	0	0	4842412	0	0	0	LRU

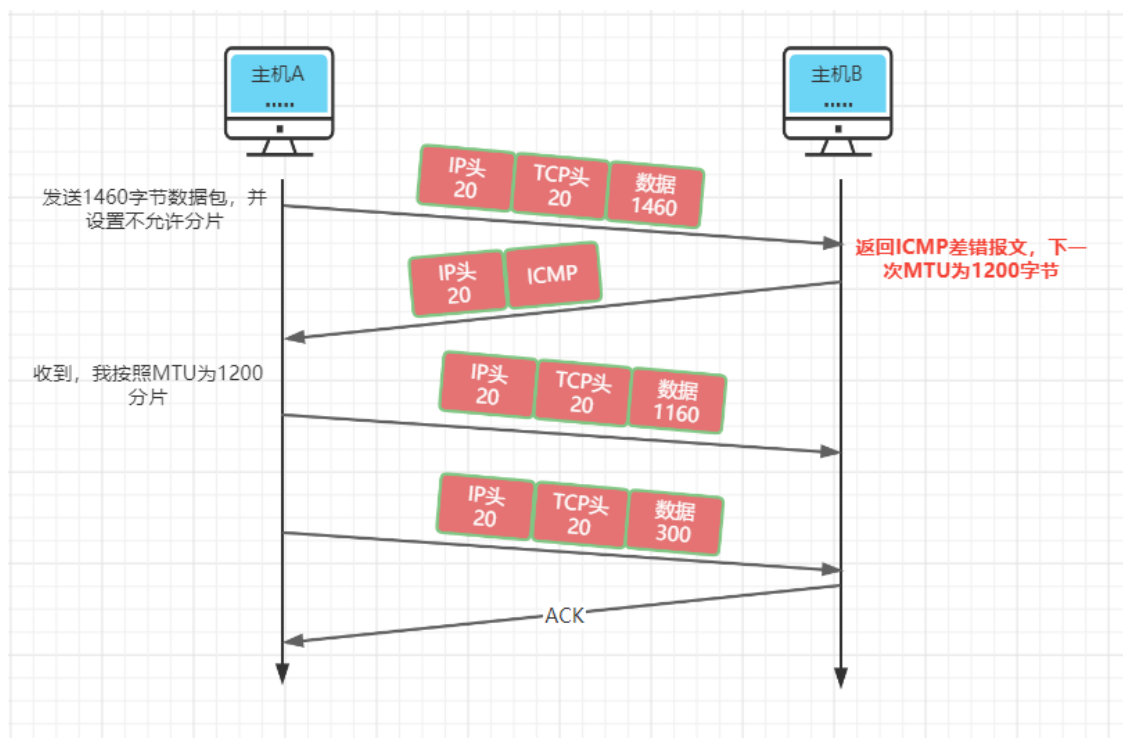
```
[root@VM-0-13-centos ~]#
```

容易想到，一个包从发送端传输到接收端，中间要跨越很多个网络，每条链路的MTU都可能不一样，就像开车，有的时候可以经过宽敞的四车道，有的时候不得行驶于乡间小路，这个通信过程中最小的MTU称为路径MTU（Path MTU）。

比如第一段链路MTU为1200字节，第二段链路MTU为800字节，第三段链路MTU为1500字节，那么路径MTU就是最小的800字节。

路径 MTU 就跟木桶效应是一个道理，木桶的盛水量由最短的那条短板决定，路径MTU也是由通信链条中最小的MTU决定。

基本原理十分简单，当一方发送了超过MTU的数据包后，对方会返回一个ICMP错误包，告知发送方包太大需要分片，并且告知发送方下一个分片的大小按照比如MTU=1200来计算，由于MTU取小者，那么A就需要随之调整MTU为1200字节：



下面我们实际抓包验证下，不过在验证前，我们需要重新认识下ping命令。

三、用ping命令小试牛刀

我们之前学习过了ping命令，其原理是基于ICMP协议，而ICMP协议实际上是封装在IP数据中的，首部为8字节长度，关于ICMP我们已经讨论过，这里不再赘述。

ping后面是可以跟着一些参数满足我们的一些测试用例的，我们将使用到的命令是：

```
ping 192.168.56.102 -l 1472 -f -n 1
```

如何查看后续参数的含义呢，我们在windows上的窗口界面输入：

```
ping -help
```

```
[C:\~]$ ping -help
选项 -help 不正确。

用法: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS]
          [-r count] [-s count] [[-j host-list] | [-k host-list]]
          [-w timeout] [-R] [-S srcaddr] [-c compartment] [-p]
          [-4] [-6] target_name

选项:
    -t          Ping 指定的主机，直到停止。
                若要查看统计信息并继续操作，请键入 Ctrl+Break；
                若要停止，请键入 Ctrl+C。
    -a          将地址解析为主机名。
    -n count    要发送的回显请求数。
    -l size     发送缓冲区大小。
    -f          在数据包中设置“不分段”标记(仅适用于 IPv4)。
    -i TTL      生存时间。
    -v TOS      服务类型(仅适用于 IPv4。该设置已被弃用，
                对 IP 标头中的服务类型字段没有任何影响)。
    -r count    记录计数跃点的路由(仅适用于 IPv4)。
    -s count    计数跃点的时间戳(仅适用于 IPv4)。
    -j host-list 与主机列表一起使用的松散源路由(仅适用于 IPv4)。
    -k host-list 与主机列表一起使用的严格源路由(仅适用于 IPv4)。
    -w timeout  等待每次回复的超时时间(毫秒)。
    -R          同样使用路由标头测试反向路由(仅适用于 IPv6)。
                根据 RFC 5095，已弃用此路由标头。
                如果使用此标头，某些系统可能丢弃回显请求。
    -S srcaddr  要使用的源地址。
    -c compartment 路由隔离舱标识符。
    -p          Ping Hyper-V 网络虚拟化提供程序地址。
    -4          强制使用 IPv4。
    -6          强制使用 IPv6。
```

很容易理解，-l 1472表示发送的数据包大小，单位是字节；-n表示只发送一个请求，因为windows下默认会自动发送四个数据包请求。-f表示不分片，实际上就是IPv4固定首部中的标志位中的DF字段：

标志位中间位即第2位记为DF (Don't Fragment)，意思是原数据报能否分片。当值为1时，表示该数据报不允许分片，当值为0时，表示该数据报允许分片。

好了，我们下面分别执行两条命令，来看下神奇情况的发生：

```
[C:\~]$ ping 192.168.56.102 -l 1472 -f -n 1
```

正在 Ping 192.168.56.102 具有 1472 字节的数据:
来自 192.168.56.102 的回复: 字节=1472 时间<1ms TTL=64

192.168.56.102 的 Ping 统计信息:
数据包: 已发送 = 1, 已接收 = 1, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
最短 = 0ms, 最长 = 0ms, 平均 = 0ms

```
[C:\~]$
```

```
[C:\~]$ ping 192.168.56.102 -l 1473 -f -n 1
```

正在 Ping 192.168.56.102 具有 1473 字节的数据:
需要拆分数据包但是设置 DF。

192.168.56.102 的 Ping 统计信息:
数据包: 已发送 = 1, 已接收 = 0, 丢失 = 1 (100% 丢失),

```
[C:\~]$
```

可以看到，我们前后发出了两个ping请求，第一次携带1472字节数据，第二次携带1473字节数据，并且都设置了DF为1即不允许分片。

仅仅相差一个字节，为什么在结果上出现了天壤之别呢？

首先ICMP首部固定为8字节，IP首部固定部分是20字节，我们这里没有额外部分，加上1472字节的数据，正好就是以太网帧中最大1500字节大小，即 $8+20+1472=1500$ 字节，对本次ping的抓包结果如下：

192.168.56.1	192.168.56.102	ICMP	1514 Echo (ping) request	id=0x0001, seq=59/15104, ttl=64 (reply in 4
192.168.56.102	192.168.56.1	ICMP	1514 Echo (ping) reply	id=0x0001, seq=59/15104, ttl=64 (request in

< Frame 3: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0

✓ Ethernet II, Src: 0a:00:27:00:00:0f (0a:00:27:00:00:0f), Dst: PcsCompu_7f:24:6e (08:00:27:7f:24:6e)

> Destination: PcsCompu_7f:24:6e (08:00:27:7f:24:6e)

> Source: 0a:00:27:00:00:0f (0a:00:27:00:00:0f)

Type: IPv4 (0x0800)

✓ Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.102

0100 = Version: 4

.... 0101 = Header Length: 20 bytes (5)

> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)

Total Length: 1500

Identification: 0x7b24 (31524)

✓ Flags: 0x4000, Don't fragment

0..... = Reserved bit: Not set

.1.. = Don't fragment: Set

..0. = More fragments: Not set

...0 0000 0000 0000 = Fragment offset: 0

Time to live: 64

Protocol: ICMP (1)

Header checksum: 0xc844 [validation disabled]

[Header checksum status: Unverified]

Source: 192.168.56.1

Destination: 192.168.56.102

✓ Internet Control Message Protocol

Type: 8 (Echo (ping) request)

Code: 0

Checksum: 0x400c [correct]

[Checksum Status: Good]

1472(数据)+8(icmp)+20(ip)=1500字节

不难理解，第一次请求正好是1500字节，没有超过MTU限制，所以可以传输成功，而第二次超出了一个字节，又不允许分片，因此传输失败。

对于第二种情况，如果ping命令后面不携带-f参数，也是可以ping成功的，只是在路上会被切分为两个包。

```
[C:\~]$ ping 192.168.56.102 -l 1473 -n 1

正在 Ping 192.168.56.102 具有 1473 字节的数据:
来自 192.168.56.102 的回复: 字节=1473 时间<1ms TTL=64

192.168.56.102 的 Ping 统计信息:
    数据包: 已发送 = 1, 已接收 = 1, 丢失 = 0 (0% 丢失),
    往返行程的估计时间 (以毫秒为单位):
        最短 = 0ms, 最长 = 0ms, 平均 = 0ms

[C:\~]$
```

我们继续来抓包证明自己的想法，去掉-f选项，允许分片，执行命令：

```
ping 192.168.56.102 -l 1473 -n 1
```

我们看到了fragmented ip protocol字眼，中文意思是分段ip协议，说明1473字节的数据被分片了，我们且来看第一个分片报文详情：

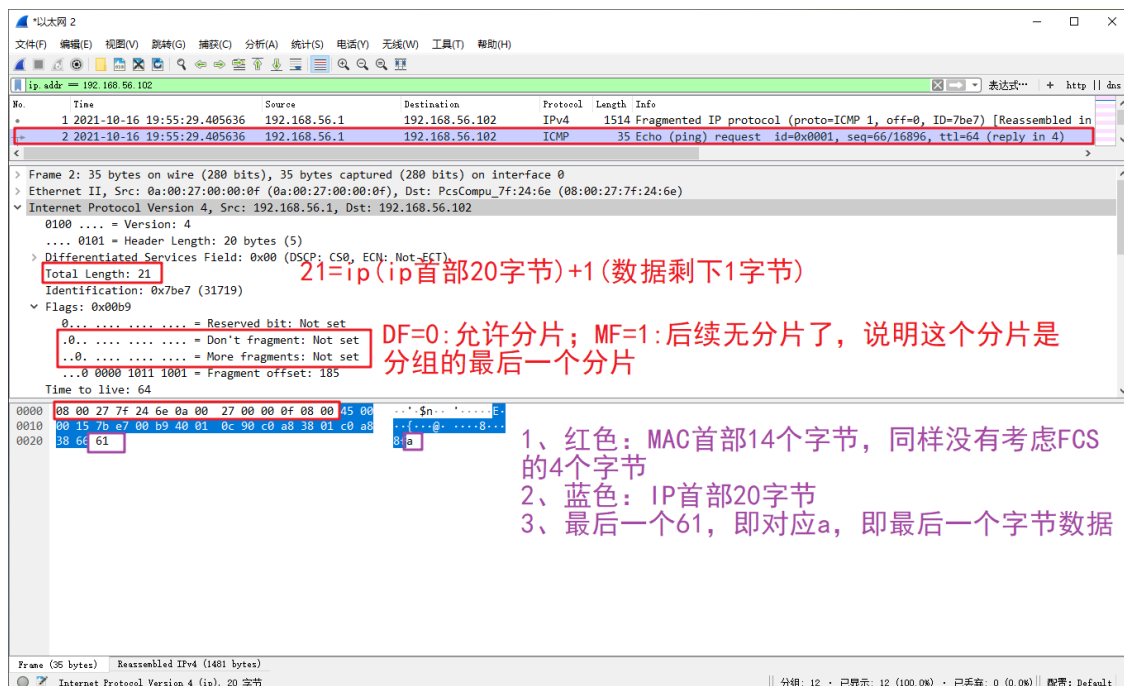
The image shows a Wireshark packet capture of a ping request. The packet list shows a fragmented IP protocol packet (1514 bytes) from 192.168.56.1 to 192.168.56.102. The packet details pane shows the following information:

- Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
- Ethernet II, Src: 0a:00:27:00:00:0f (0a:00:27:00:00:0f), Dst: PcsCompu_7f:24:6e (08:00:27:7f:24:6e)
- Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.102
- 0100 = Version: 4
- 0101 = Header Length: 20 bytes (5)
- > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
- Total Length: 1500
- Identification: 0x0be7 (31719)
- Flags: 0x2000, More fragments
- 0... .. = Reserved bit: Not set
- .0... .. = Don't fragment: Not set
- ...1... .. = More fragments: Set
- ...0 0000 0000 0000 = Fragment offset: 0
- Time to live: 64

The packet bytes pane shows the raw data of the packet. The following table summarizes the color-coded annotations in the image:

Color	Description
Green	MAC头14字节 (Wireshark去除了FCS的4字节)
Blue	IP首部20字节
Yellow	ICMP首部8字节
Purple	data数据 (1472字节)

继续看下一个报文详情：



也可以注意到一个细节，第二个报文段中不包含ICMP首部。此外可以注意到wireshark上显示的包length为35长度，之前不是说过至少是60字节的吗？（加上FCS校验应该是64字节）

实际上，如果数据部分小于以太网帧需要的最小的46字节时，就会在以太网层自动填充0，使得数据达到46字节，从而达到最小64字节的帧长度要求。而这里显示35字节大概率是因为wireshark捕获的时候还未进行填充，从而显示出了这个异常的长度包（对于这一点如果有错误还请指正）。

四、压轴登场：MSS

MSS的英文全称叫Max Segment Size，是TCP最大段大小。

在建立TCP连接的同时，也可以确定发送数据包的单位，我们称为MSS，这个MSS正好是IP中不会被分片处理的最大数据长度。

TCP在传送大量数据时，是以MSS的大小将数据进行分割发送的，重发时也是以MSS为单位。

MSS是在三次握手的时候，在两端主机之间被计算得出，两端主机在发出建立连接的请求时，会在TCP首部中写入MSS选项，告诉对方自己的接口能够适应的MSS的大小，然后在两者之间选择一个较小的值投入使用。

从以上描述中可以看出：

$MSS = MTU - IP \text{ header头大小} - TCP \text{ 头大小}$

这样一个 MSS 的数据恰好能装进一个 MTU 而不用分片。

在我们熟悉的以太网中，TCP的MSS最大值是：以太网MSS=1500(MTU)-20(IP首部长度)-20(TCP首部大小)=1460字节

好了，理论介绍完毕，下面我们来看下实际抓包。

我的虚拟机上安装了一个nginx，端口使用的是熟知端口号80，我在本地客户端通过curl命令访问nginx的服务：

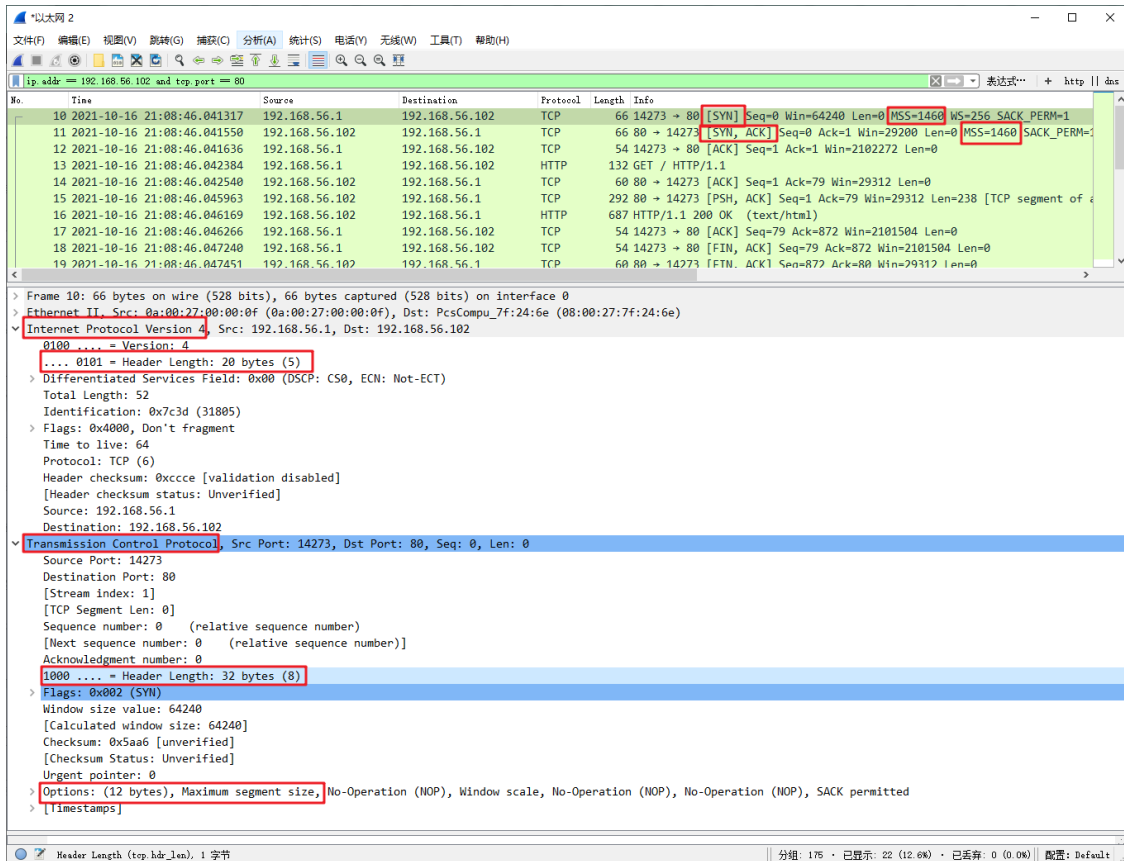
```
[C:\~]$ curl http://192.168.56.102
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %             Dload  Upload  Total  Spent    Left  Speed
100  633  100  633    0     0   633      0  0:00:01 --:--:-- 0:00:01 42200
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to BACKUP-92.168.56.102 nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

从下图抓包中可以看到，MSS的值是在三次握手的SYN报文中商量出来的，可以看到互相说明自己允许的最大段大小都是1460字节，那么MSS的值就可以取为1460。

在以太网协议中，一般情况下MTU是1500字节，MSS为1460字节(相差20字节的IP首部+20字节的TCP首部)，不过以上是基于IPv4协议讨论的，在IPv6中，IP首部长度是40字节，那么MSS一般就是1440字节了。



MSS选项只能出现在SYN报文中，为此SYN报文TCP头部里包含了12字节的选项（Options）字段，可以清晰看到里面有一个MSS选项，所以本次的TCP的握手机文中的TCP首部长为32字节，即20字节的固定首部加12字节的可变首部，整体为4字节的整数倍。

五、既然IP层会分片，为什么TCP层还需要MSS？

问题等同于：如果TCP整个报文每次都交给IP层进行分片，会有什么问题？

我们已经知道，当IP层有一个超过MTU大小的数据（TCP头部 + TCP数据）要发送，那么IP层就要进行分片，把数据分片成若干片，保证每一个分片都小于MTU。把一份IP数据报进行分片以后，由目标主机的IP层来进行重新组装后，再交给上一层TCP传输层。

这看起来井然有序，但这存在隐患的，那么当如果一个IP分片丢失，整个IP报文的所有分片都得重传。

因为IP层本身没有超时重传机制，它由传输层的TCP来负责超时和重传。

当某一个IP分片丢失后，接收方的IP层就无法组装成一个完整的TCP报文（头部 + 数据），也就无法将数据报文送到TCP层，所以接收方不会响应ACK给发送方，因为发送方迟迟收不到ACK确认报文，所以会触发超时重传，就会重发「整个TCP报文（头部 + 数据）」。

因此，可以得知由IP层进行分片传输，是非常没有效率的。

所以，为了达到最佳的传输效能 TCP 协议在建立连接的时候通常要协商双方的 MSS 值，当 TCP 层发现数据超过 MSS 时，则就先会进行分片，当然由它形成的 IP 包的长度也就不会大于 MTU，自然也就不用 IP 分片了。

经过 TCP 层分片后，如果一个 TCP 分片丢失后，进行重发时也是以 MSS 为单位，而不用重传所有的分片，大大增加了重传的效率。