网络基础之 TSO,UFO,GSO,LRO,GRO

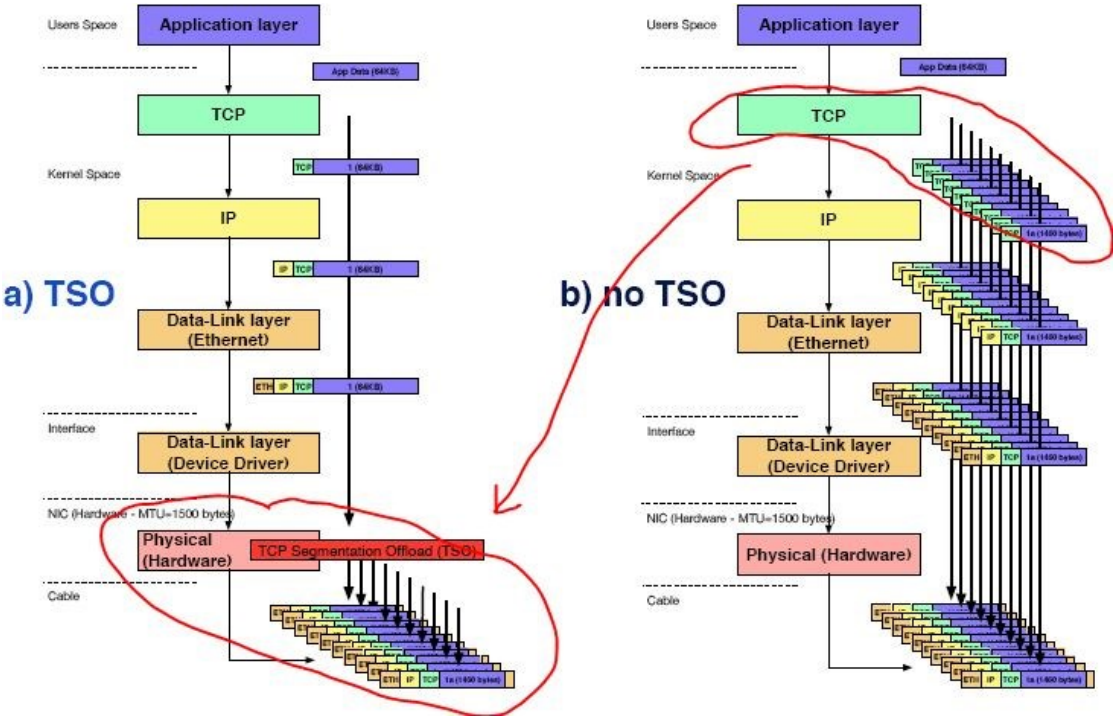
mnstory.net

本文原创部分占比不多，图片多是盗用，不知道最开始是谁画的，我是从 http://geek.csdn.net/news/detail/67260 取的。此文主要是做一个自己的梳理。

TSO/UFO

TCP 的每个数据包大小不能超过 MSS 值，在发送端超过的，需按照 MSS 进行分段(segment)。分段这事情，老是让 CPU 来做，那是很耗精力的事，做网卡的乐意分享此任务，将其 offload 到网卡，当网卡支持 TCP 分段这功能的时候，我们就说它支持 TSO(TCP segmentation offload)。

如果网卡不支持 TSO 功能，TCP 数据会在内核协议栈的 TCP 层按照 MSS 分成若干段(segmentation)，然后经 IP 层，经链路层，经设备驱动层，再达网卡。有了 TSO 功能，在 TCP 大数据块不超过 64K 的情况下，可以一个大包走到网卡。对比一下，两者的差异：



图：有 tso vs 没有 tso

有 TSO，内核协议栈上少了很多数据包的划分、小包传输、处理、checksum、上下文切换等，从而减轻了 CPU 的任务。

IP 分片后，只有第一个分片包带有上层协议的头部(例如，UDP 头部、ICMP 头部)其余分片只有 IP 头。当 IP 分片到达最终目的地后，根据 IP 头部中的信息，在网络层进行重

组，而不会在路径上重组。
TCP 分段，会在每个分段上都附上 TCP 的协议头，到达端点后，根据 TCP 头部信息在传输层进行重组。如果一个 TCP 分段需要多个 IP 分片来承接，那么当其中一个 IP 分片丢失，这个 TCP 分段将重传(意味着多个 IP 分片重传)，IP 层本身是不可靠的，不具备重传机制，需要 TCP 层来进行，所以，应该尽量避免一个 TCP 分段需要多个 IP 分片来承接，默认 MSS 根据 MTU 来计算的，也正是这个原因。
UDP 协议本身不具备类似 TCP 的分段功能，所以，大的 UDP 数据报文，需要由 IP 层来进行分片传送，和 TSO 类似，网卡将 UDP 报文的 IP 分片功能 offload 到网卡来做，那就叫 **UFO**(udp-fragmentation-offload)技术。

查看一下，网卡是否支持 TSO/UFO:

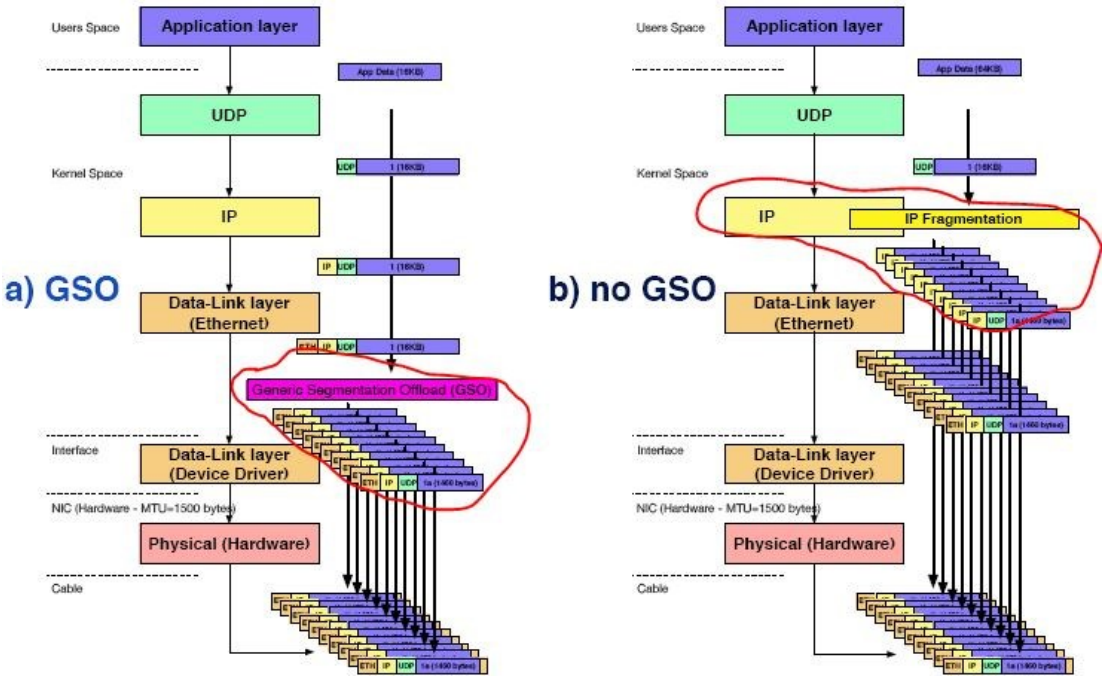
```
# ethtool -k eth0
tcp-segmentation-offload: on
    tx-tcp-segmentation: on
    tx-tcp-ecn-segmentation: off [fixed]
    tx-tcp6-segmentation: on
udp-fragmentation-offload: off [fixed]
```

GSO

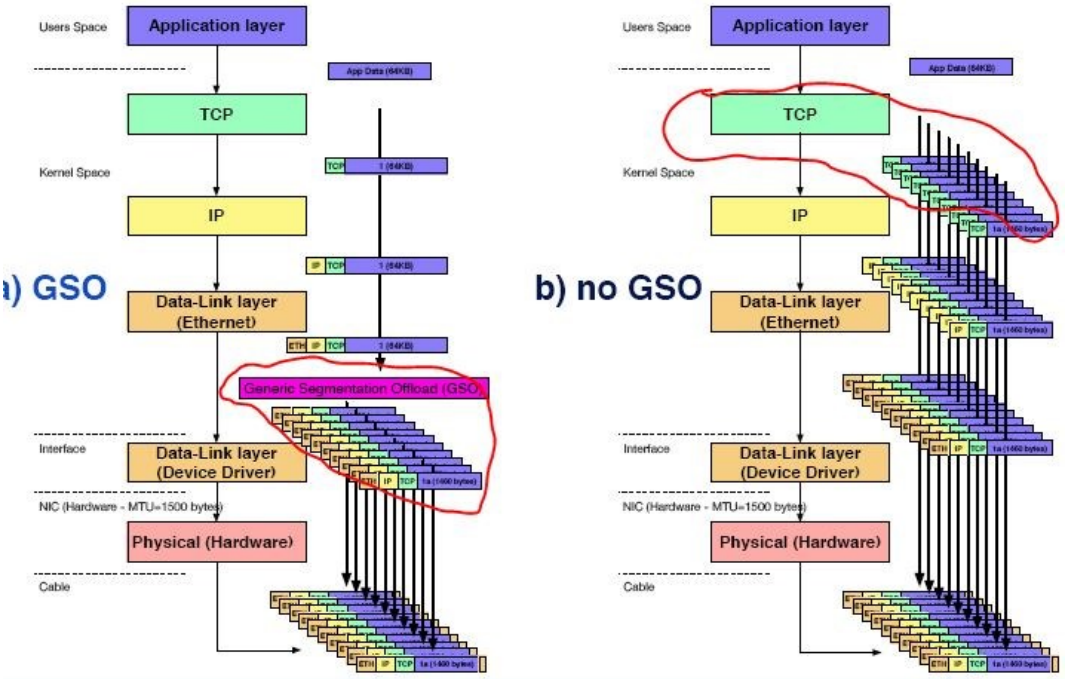
就 TSO 功能而言，它的主要优点有二:

1. TCP/IP 协议栈上传送一次大块数据，而非多次小块数据，节约 CPU。
2. 让网卡来分段功能，节约 CPU，这点需要硬件支持。

如果硬件不支持，我们就做不了 TSO，但是，我们可以做 GSO(Generic Segmentation Offload)，GSO 将 TSO 功能泛化，让数据跨过 IP 层，链路层，在数据离开协议栈，进入网卡驱动前进行分段，不论是 TCP 还是 UDP，都是分段(每个包都附加 TCP/UDP 头部)，这样，当一个段丢失，不需要发送整个 TCP/UDP 报文。其次，路径上的 CPU 消耗也会减少，所以可以说 GSO 是对 UFO 的一种改良(UFO 是 IP 分片，只有头包带有 UDP 头，丢掉一片要整段重传)



图：对于 UDP，在物理网卡不支持 UFO 时，使用和不使用 GSO 的情形



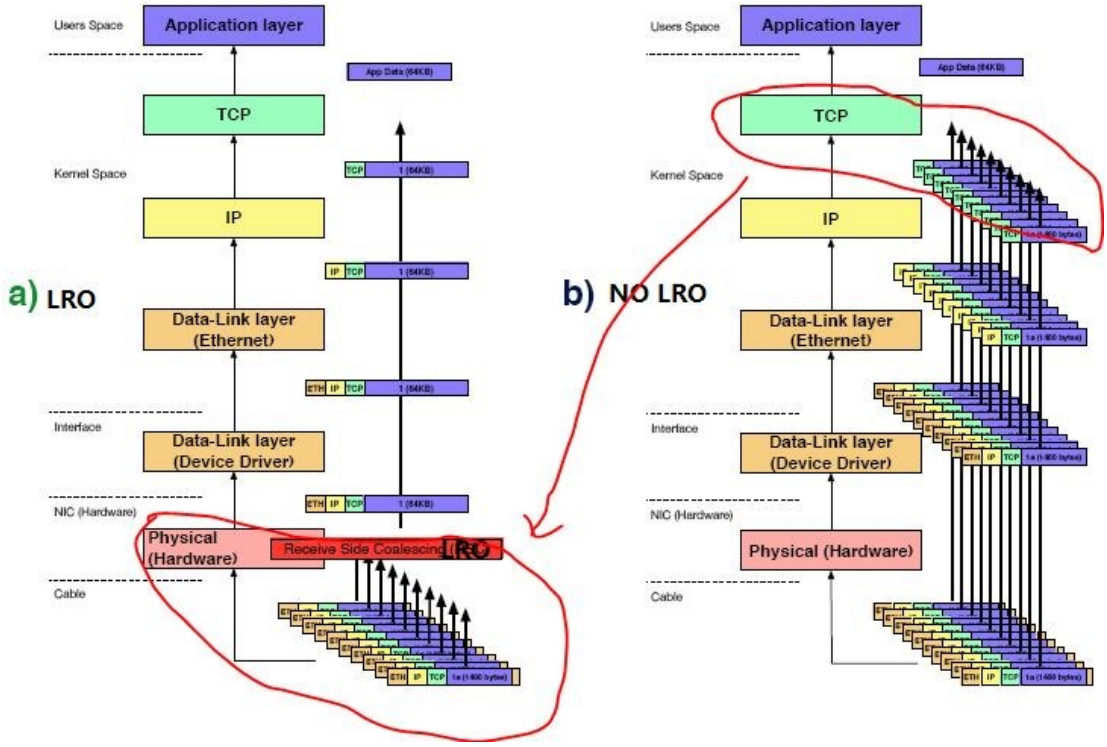
图：对 TCP，在网卡不支持 TSO 时，使用和不使用 GSO 的情形

查看一下，GSO 是否开启：

```
# ethtool -k eth0
generic-segmentation-offload: on
```

LRO

发包的过程说完，接收 TCP 包的时候，如果网卡能将多个 TCP 分段合并成一个超级 SKB，然后递交到上层网络，以减少 CPU 消耗，这个就叫网卡支持 LRO(large receive offload)。



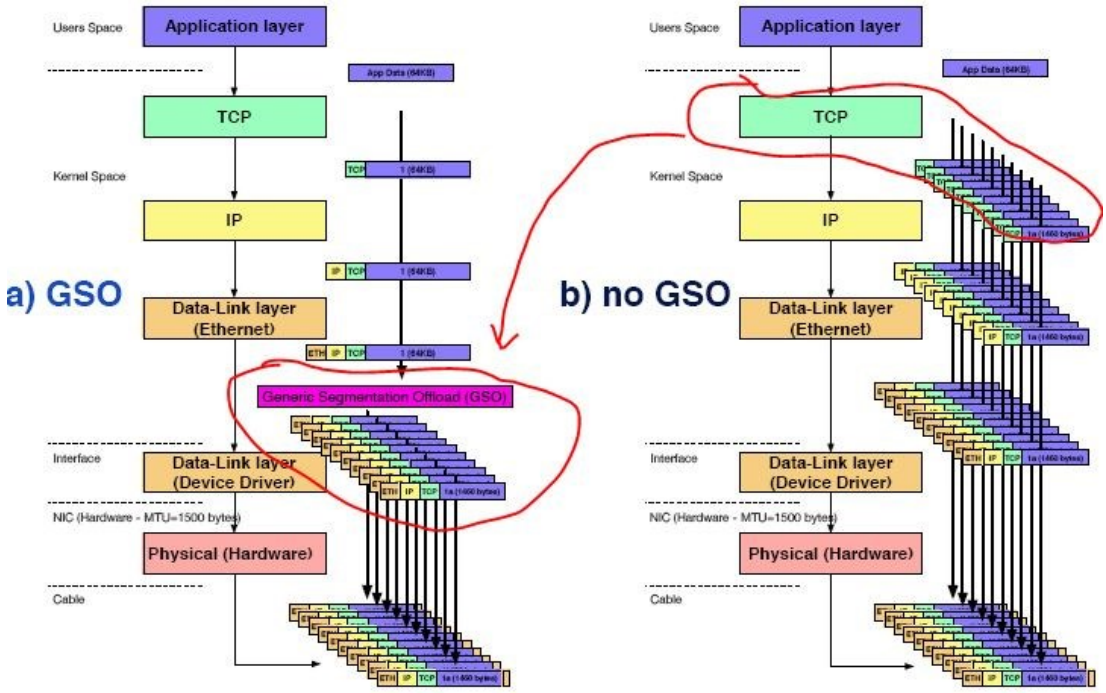
图：物理网卡支持 LRO vs 不支持 LRO

查看一下，网卡是否支持 LRO:

```
# ethtool -k eth0
large-receive-offload: off [fixed]
```

GRO

LRO 本身实现上有一些问题，比如多个包的状态信息不一致，合并后会导致状态被破坏，并且也依赖于网卡。于是有了更通用的版本 GRO(generic-receive-offload)，他将合并操作放到了设备启动层，保留了每个数据包的一些熵信息(五元组)供后面解析使用。



图：在不支持 LRO 的情况下，对 TCP 使用和不使用 GRO 的情形

Offload	传输段还是接收端	针对的协议	Offloading 的位置	ethtool 命令输出中的项目	
TSO	传输段	TCP	NIC	tcp-segmentation-offload	需要网卡支持，多数都支持
UFO	传输段	UDP	NIC	udp-fragmentation-offload	需要网卡支持，多数不支持
GSO	传输段	TCP/UDP	NIC 或者离开 IP 协议栈进入网卡驱动前	generic-segmentation-offload	内核特征，主要是延迟分段
LRO	接收段	TCP	NIC	large-receive-offload	需要硬件支持，多数都支持
GRO	接收段	TCP/UDP	NIC 或者离开网卡驱动进入 IP 协议栈前	generic-receive-offload	内核特征，主要是提前合并

查看一下，GRO 是否开启：

```
# ethtool -k eth0
generic-receive-offload: on
```

2017/7/10