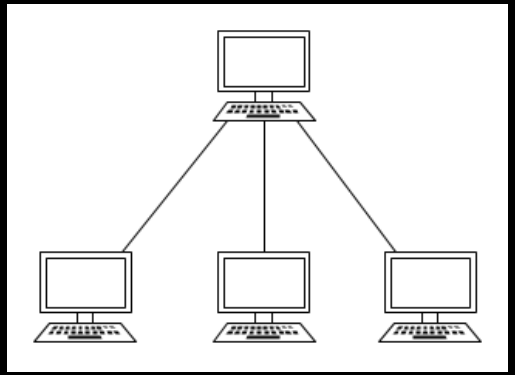
IPv4/IPv6无TCP状态的简单Web服务程序框架及应用

中国科学技术大学
网络信息中心
张焕杰 james@ustc.edu.cn

目录

一、TCP连接复用处理方式



TCP Client/Server模式，一个server服务多个client

服务器端OS内核记录和处理每个TCP的连接状态信息

操作系统可以支持100万并发连接（大约30G内存）

服务器端的服务程序需要处理这些TCP连接的收发事件：

接收新建连接、接收请求、发送应答、拆除连接

常见的TCP复用处理方式有：

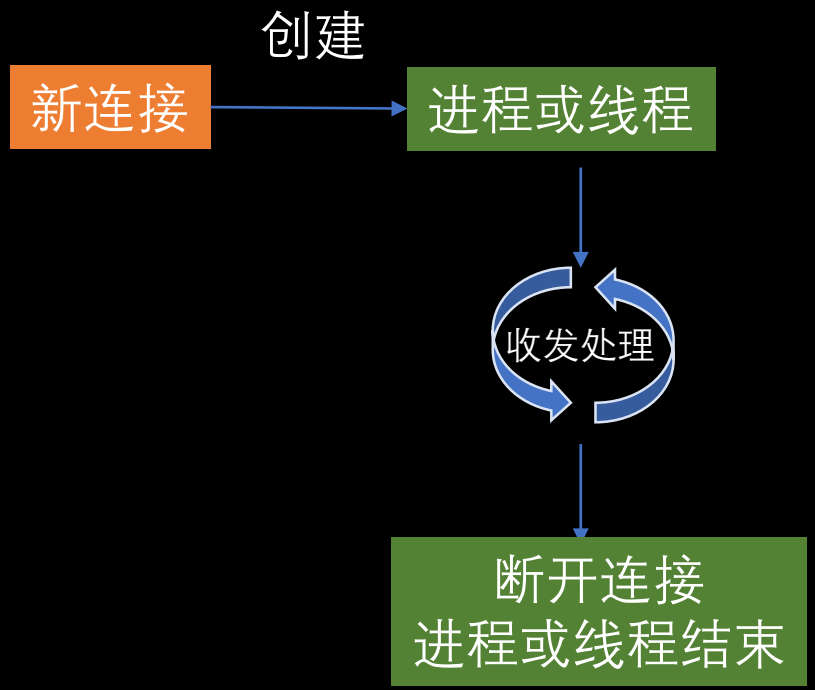
进程、线程、select、epoll

C10K问题，2001年提出（http://www.kegel.com/c10k.html）

单个服务器如何支持1万TCP并发连接的有效处理

目前面临C10M问题

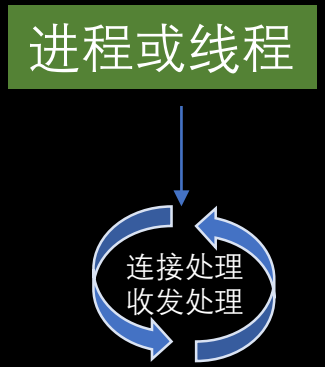
进程、线程复用



每个进程或线程仅仅服务一个连接
有新连接到来时，创建一个进程或线程
进程或线程读写可能会阻塞
处理完毕后断开连接，进程或线程结束

		并发数
apache	进程	N*100
IIS	线程*	N*1000

select、epoll复用

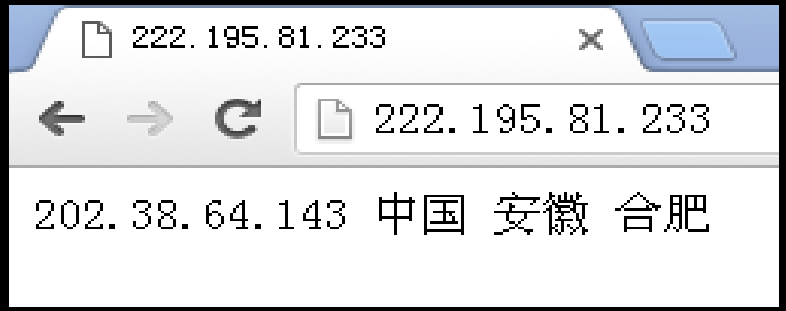


一个进程或线程同时服务多个连接
进程或线程的读写不能阻塞
因此服务端程序需要使用select或epoll之类的系统调用
　　获取有新连接信息
　　获取哪些连接可以读写的信息
select的每次系统调用将所有的连接信息传入传出因此效率低
epoll采用边缘触发，效率高很多，解决了C10K问题

		并发数
squid	select	N*1000
Nginx	epoll	N*100000

简单web服务

简单WEB服务：请求和响应都很短的WEB会话，发送和接收数据包很少



发送的HTTP请求在1个以太网包中传送
接收的HTTP应答在1个或几个以太网包中传送

简单web服务的处理效率分析

使用效率最高epoll接口，每个HTTP会话在服务器端需要如下系统调用

1. epoll_wait 获得新连接通知
2. accept 接受连接
3. fcntl 将新连设置为无阻塞方式
4. epoll_ctl 通知内核监视新连数据可读事件
5. epoll_wait 获得连接可读事件通知
6. read 读取请求
7. write 发送应答
8. close 关闭连接

有其他优化手段吗?

其中1、5的系统调用
可能与其他会话共用

每次系统调用至少有约200 cycles开销
大致是读写1次内存的代价
系统调用执行还需要更多时间
内核里维护TCP连接状态信息要花代价
内存、定时器

二、简单HTTP请求数据包交互

典型通信过程一共8个数据包
其中6个为最小包无实际内容
只有④⑤数据包有信息传送

客户端

服务器端

SYN ①

② SYN+ACK

服务器不主动发送数据包
收到客户端的数据包后应答

ACK ③

HTTP GET/POST
请求

ACK ④

⑤ ACK+FIN+PSH
HTTP应答

ACK ⑥

FIN+ACK ⑦

⑧ ACK

无TCP状态的WEB服务处理

- 如果HTTP请求是幂等的

  无论调用多少次返回结果相同的HTTP请求，如大部分查询类请求

- 服务器端可以不存放和处理TCP连接状态信息，只要：

  服务器端能针对客户端发来数据包给出应答

  并处理好数据包中序列号、校验和等信息

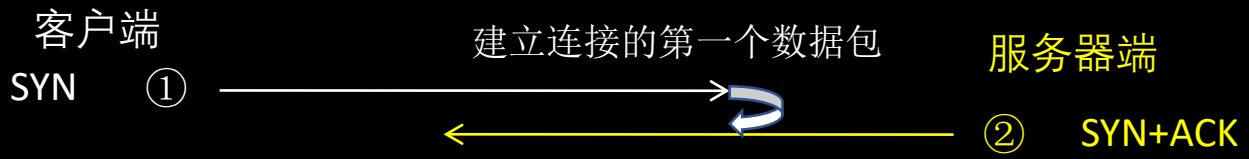
  客户端和服务器的HTTP通信就可以正常进行

重要前提：请求在一个以太网包传输用GET不用POST！！

- 服务器端不存放和处理TCP连接状态信息，意味着服务器可以支持

  无限多的TCP并发连接

数据包处理过程 1仅有SYN标志数据包



1. 交换源、目的MAC地址
   交换源、目的IP地址
   交换源、目的TCP端口
2. 设置SYN+ACK标志
   设置sent_seq
   设置recv_ack为原sent_seq+1
   设置tcp头的数据偏移为0x50
3. 设置ip包长度、ttl，重新计算TCP和IP校验和

对于IPv4/v6处理大同小异

设置sent_seq时，根据源地址、源端口、目的地址、目的端口和程序启动时选择的一个随机数来生成sent_seq，实现类似tcp syn_cookie功能。

数据包处理过程 2有FIN标志数据包

客户端　　服务器端

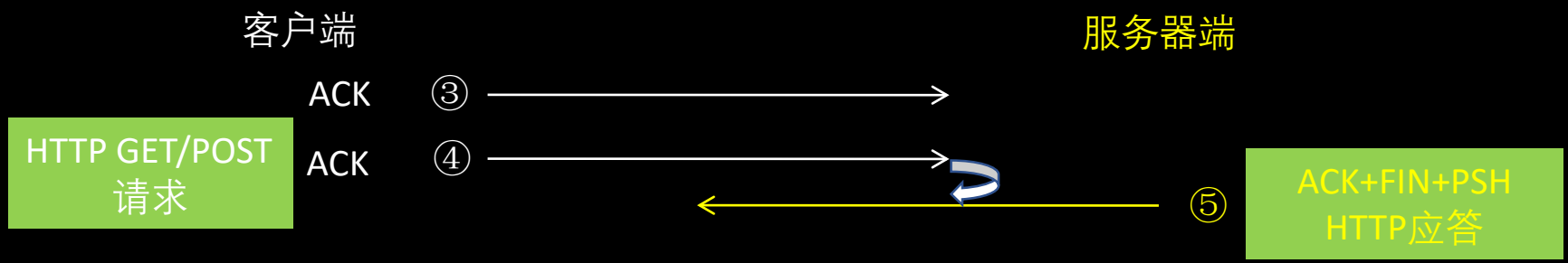
1
交换源、目的MAC地址
交换源、目的IP地址
交换源、目的TCP端口

2
设置ACK标志
设置sent_seq为原recv_ack
设置recv_ack为原sent_seq+1
设置tcp头的数据偏移为0x50

3
设置ip包长度、ttl，重新计算TCP和IP校验和

拆除连接数据包

数据包处理过程 3有ACK但无SYN标志数据包



1

有效性判断：

- 检查tcp包载荷长度，如果小于5字节不再继续处理
- 检查收到的recv_ack是否与1的sent_seq生成的相同，如果不同不再继续处理（伪造的请求，可能被利用进行DDoS攻击）

2

调用相关应用处理过程，解析并处理请求，得到HTTP应答信息

数据包处理过程 3有ACK但无SYN标志数据包

3

交换源、目的MAC地址
交换源、目的IP地址
交换源、目的TCP端口

4

设置ACK+PSH+FIN标志
设置sent_seq为原recv_ack
设置recv_ack为原sent_seq+原tcp包载荷长度
设置tcp头的数据偏移为0x50以清除可能的tcp选项
将HTTP应答信息装入新的tcp载荷

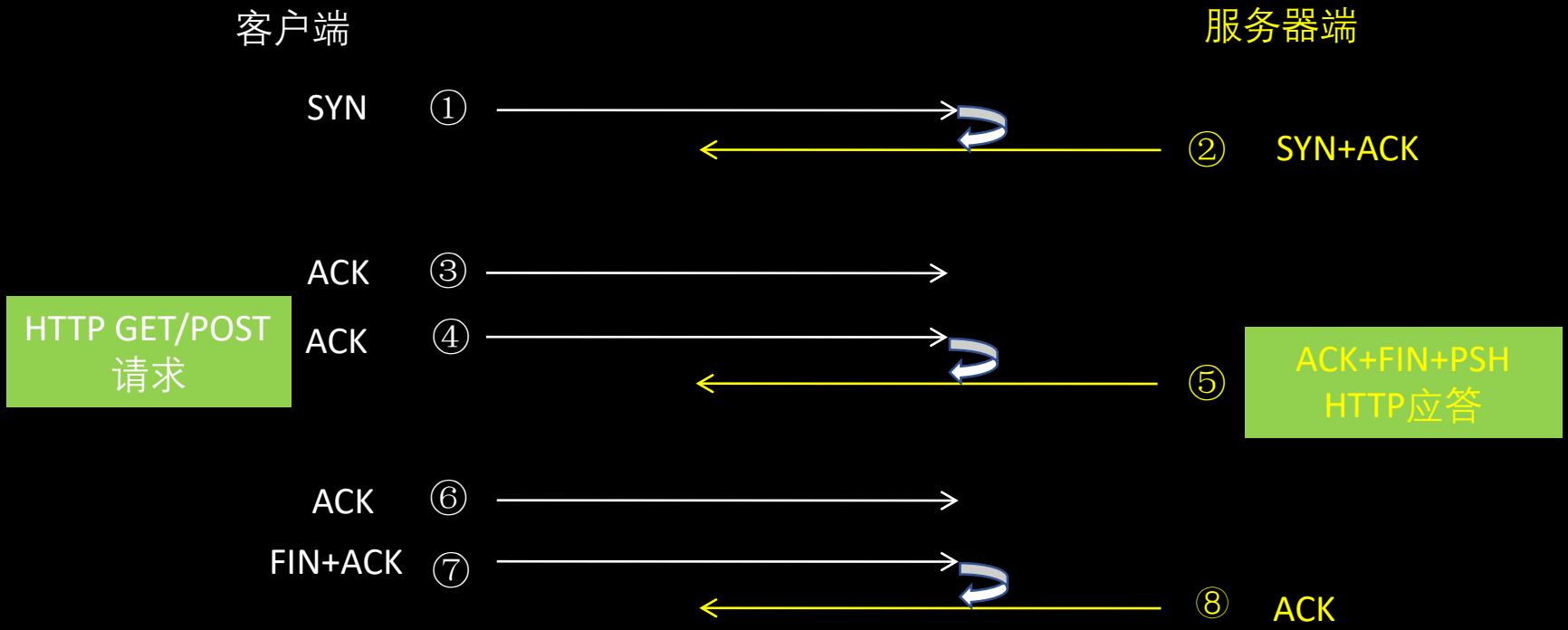
如果HTTP应答信息超过了TCP MSS（Maximum Segment Size，最大报文长度），应答包无法装在1个以太网包内发送，则还需要对TCP包进行分片处理，依次将分片后的应答IP包发送

5

设置ip包长度、ttl，重新计算TCP和IP校验和

简单HTTP请求数据包交互

丢包的影响?
⑤丢失会导致重复处理



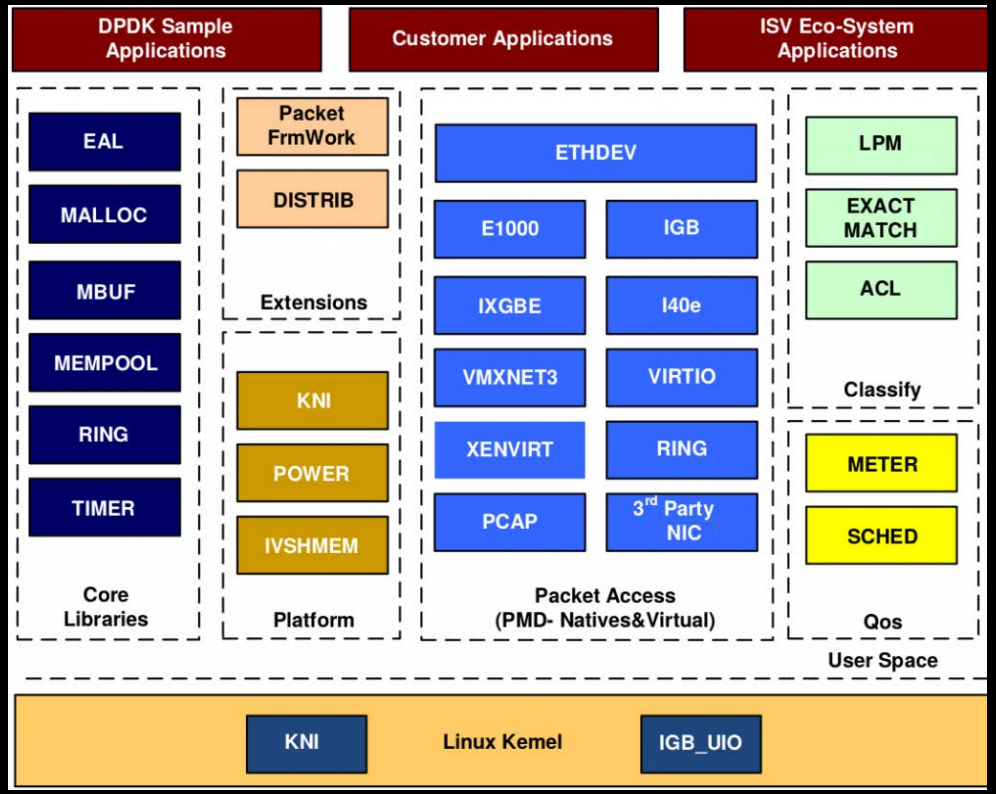
三、无TCP状态的简单WEB服务程序框架

如果高速的接收/发送以太网数据包?
RAW_SOCKET，通过操作系统内核收发网卡数据包
以太网数据包-----网卡-----中断-----内核-----应用程序，不够快

DPDK（Data Plane Development Kit）是6WIND、Intel等多家公司开发
主要基于Linux系统运行，提供了快速数据包处理的函数库与驱动集合

DPDK使用大页内存、cache对齐、预取、SIMD指令、多核、线程绑定、轮询模式、多队列网卡、无锁数据结构等多项技术组合，可以在通用x86平台上实现N*100Mpps的转发速度）。快！对于2GHz处理器：
1G最小包线速1.488Mpps，1344 cycle处理一个数据包
10G最小包线速 14.88Mpps，134 cycle处理一个数据包（约1次访内存时间）

DPDK函数库与驱动


DPDK Sample Applications
Customer Applications
ISV Eco-System Applications
EAL
MALLOC
MBUF
MEMPOOL
RING
TIMER
Core Libraries
Packet FrmWork
DISTRIB
Extensions
KNI
POWER
IVSHMEM
Platform
ETHDEV
E1000
IGB
IXGBE
I40e
VMXNET3
VIRTIO
XENVIRT
RING
PCAP
3rd Party NIC
Packet Access (PMD- Natives&Virtual)
LPM
EXACT MATCH
ACL
Classify
METER
SCHED
Qos
User Space
KNI
Linux Kemel
IGB_UIO

WEB服务程序框架

https://github.com/bg6cq/dpdk-simple-web

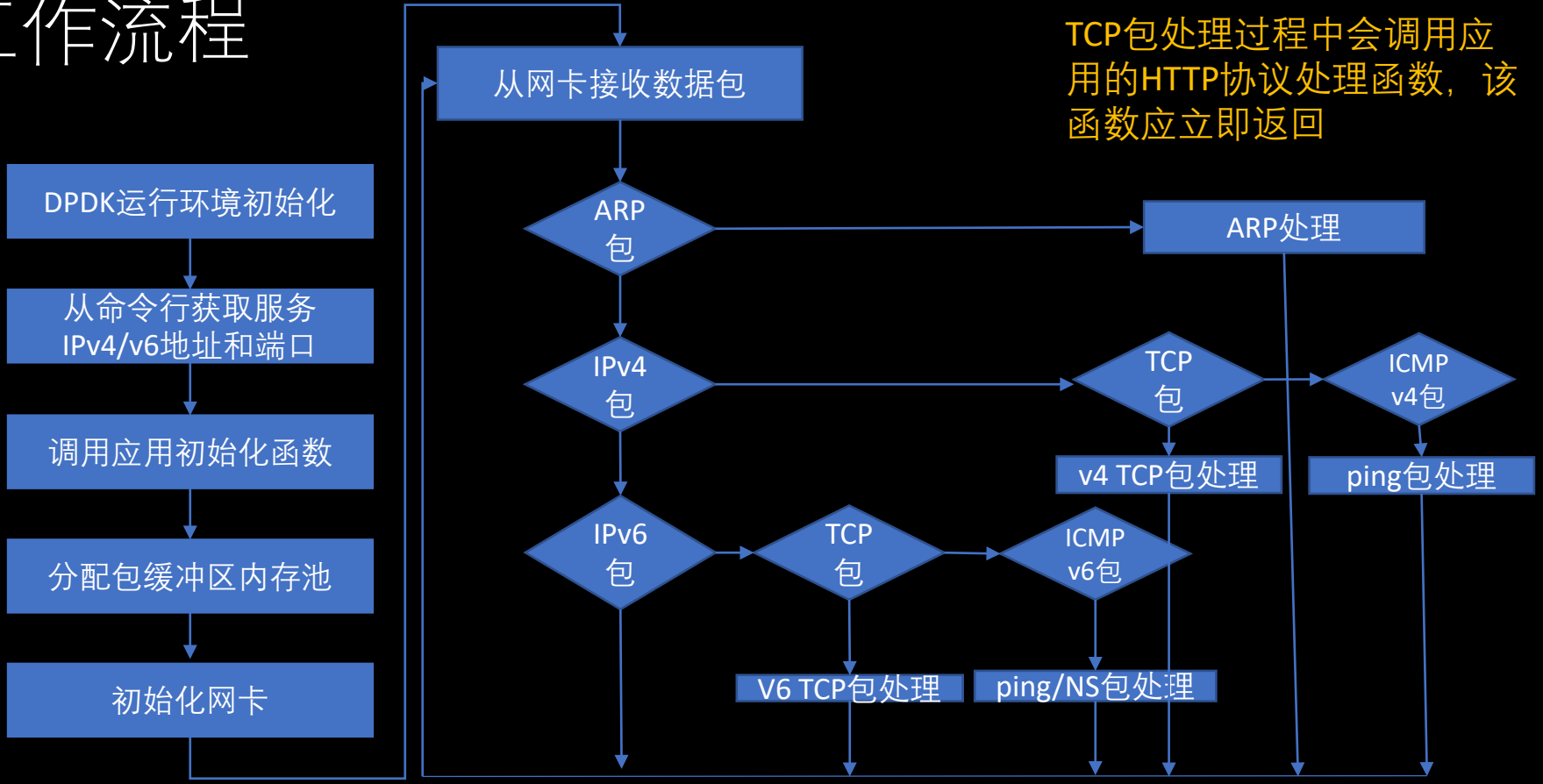
WEB服务程序使用DPDK函数库和驱动，接管网卡
　　使用轮询模式高速接收和发送数据包，实现高性能的简单WEB服务

服务程序运行时，接收和发送数据包不经过操作系统内核处理，不需要系统调用

服务程序对IPv4的ICMP/ARP协议以及IPv6的ICMPv6协议部分处理
　　实现了ping应答、ARP应答与IPv6 NS应答

如果网卡支持各种校验和计算，尽量由网卡来计算，减少CPU负担

工作流程



TCP包处理过程中会调用应用的HTTP协议处理函数，该函数应立即返回

接收数据包片段

```
while (1) {
    struct rte_mbuf *bufs[BURST_SIZE];
    const uint16_t nb_rx = rte_eth_rx_burst(port, 0, bufs, BURST_SIZE);
    if (unlikely(nb_rx == 0))
            continue; // 忙等，未收到数据包时，处于循环，CPU利用率100%
    for (i = 0; i < nb_rx; i++) {.    // 一次性收到了nb_rx个以太网包，对每个包处理
            int len = rte_pktmbuf_data_len(bufs[i]);
            struct ether_hdr *eh = rte_pktmbuf_mtod(bufs[i], struct ether_hdr *);
            if (eh->ether_type == rte_cpu_to_be_16(ETHER_TYPE_IPv4)) {  // IPv4 protocol
                    struct ipv4_hdr *iph;
                    …
            } else if (eh->ether_type == rte_cpu_to_be_16(ETHER_TYPE_ARP)) {          // ARP protocol
                    if (process_arp(bufs[i], eh, len))
                            continue;
            }
            …
            rte_pktmbuf_free(bufs[i]);
    }
}
```

```
rte_memcpy((unsigned char *)&eh->d_addr, (unsigned char *)&eh->s_addr, 6);
rte_memcpy((unsigned char *)&eh->s_addr, (unsigned char *)&my_eth_addr, 6);
iph->dst_addr = iph->src_addr;
iph->src_addr = my_ip;
iph->time_to_live = TTL;
iph->hdr_checksum = 0;
iph->hdr_checksum = rte_ipv4_cksum(iph);
icmph->icmp_type = IP_ICMP_ECHO_REPLY;
icmph->icmp_cksum = 0;
icmph->icmp_cksum = ~rte_raw_cksum(icmph, len - ETHER_HDR_LEN - ipv4_hdrlen);

int ret = rte_eth_tx_burst(0, 0, &mbuf, 1);   // 发送一个以太网包
if (ret == 1) {
        send_icmp_pkts++;
        return 1;
}
```

处理icmp数据包片段

四、IP地址信息查询应用

客户端发送http://serverip/x.x.x.x的请求，服务器返回对应的IP地址信息

使用北京天特信科技有限公司ipip.net免费IP地址库，返回IP地址对应的地理信息
国内IP精确到城市，国外IP精确到国家

程序启动时，应用的初始化函数将免费IP地址信息数据库（2019年1月份发布的版本约为2.7M字节）读入到内存

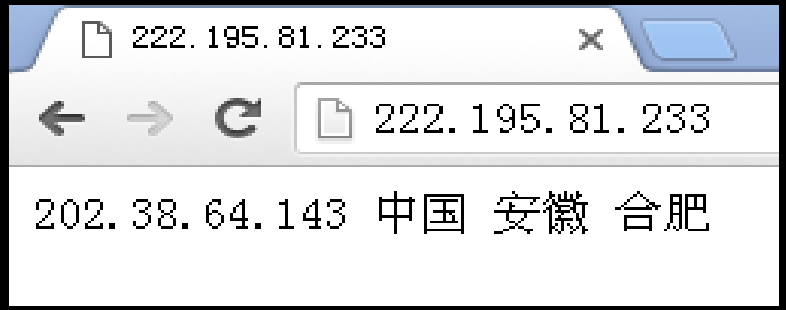
处理客户端的HTTP请求时，查询内存中的IP地址库，给出html内容应答

返回的IP地址信息很短在一个以太网包中可以传输完毕，不涉及到分片处理

程序运行

```
root@dpdk-dev:/usr/src/ipdesc-dpdk# cat run
build/ipdesc-dpdk -n1 -c1 -- 222.195.81.233 80 --ip6 2001:da8:d800:381::233
root@dpdk-dev:/usr/src/ipdesc-dpdk#
root@dpdk-dev:/usr/src/ipdesc-dpdk# sh run
EAL: Detected 4 lcore(s)
EAL: No free hugepages reported in hugepages-1048576kB
EAL: Multi-process socket /var/run/.rte_unix
EAL: Probing VFIO support...
EAL: PCI device 0000:0b:00.0 on NUMA socket -1
EAL:   Invalid NUMA socket, default to 0
EAL:   probe driver: 15ad:7b0 net_vmxnet3
EAL: PCI device 0000:13:00.0 on NUMA socket -1
EAL:   Invalid NUMA socket, default to 0
EAL:   probe driver: 15ad:7b0 net_vmxnet3
My IP is: 222.195.81.233, port is 80
My IPv6 is: 2001:da8:d800:381::233
My IPv6 node multicast address is: ff02::1:ff00:233
user_init_func: argc=1
Version: 643e-dirty
RX TCP  checksum: support
TX TCP  checksum: support
I will not use hardware checksum
Port 0 MAC: 00 50 56 b8 fd 24
My ether addr is: 00:50:56:B8:FD:24
Core 0 forwarding packets. [Ctrl+C to quit]
```

IP地址信息查询应用



使用本框架的服务端代码
https://github.com/bg6cq/ipdesc-dpdk

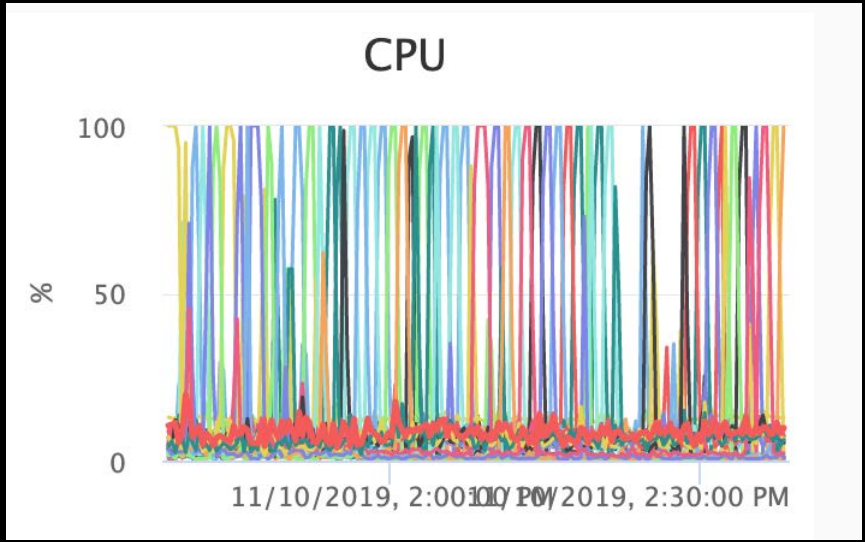
使用epoll处理tcp复用的服务端代码
https://github.com/bg6cq/ipdesc

CPU占用率 100%

```
top - 14:35:57 up 554 days,  5:42,  2 users,  load average: 1.09, 0.98, 0.92
Tasks: 113 total,   2 running, 109 sleeping,   1 stopped,   1 zombie
%Cpu(s): 25.0 us,  0.1 sy,  0.0 ni, 74.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2044688 total,    76192 free,  1925540 used,    42956 buff/cache
KiB Swap:  2093052 total,  2032716 free,    60336 used.    18708 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
17821 root      20   0 1888096   4192    888 R  99.7  0.2   8:07.88 ipdesc-dpdk
```

物理服务器CPU占用率



性能对比

受限于测试机器，高性能并未发挥

epoll版本：`ab -n 10000 -c 1000 http://222.195.81.234/202.38.64.1`

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   36   8.7     35      73
Processing:    23   38   6.9     37      73
Waiting:        0   25   9.5     24      50
Total:         55   74   9.1     74     109
```

12002请求/秒，2.080MB/s

本文版本：`ab -n 10000 -c 1000 http://222.195.81.233/202.38.64.1`

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   20   5.7     20      37
Processing:    12   22   6.0     22      46
Waiting:        0   10   7.5      8      38
Total:         37   42   4.3     40      59
```

21178请求/秒，3.796MB/s

五、考试分数查询应用

数据量小	每个考生2000字节信息，50万考生1GB，轻松读入内存
并发量大	公布分数的瞬间，人人都想查
逻辑清晰	输入页面，用户输入座位号、身份证号，单击查询
处理简单	以座位号为关键字查找、验证身份证号，输出
时效性强	一小时内是高峰，后面减少，保障这一小时非常重要

考试分数查询应用性能估算

假定HTTP应答正好占满以太网MTU 1500字节数据包，服务器产生的3个应答数据包分别是46字节、1500字节、46字节，加上每个包的以太网开销18字节和包间隔20字节，一次查询在以太网上传输时实际占用46*2+1500+38*3=1706字节。

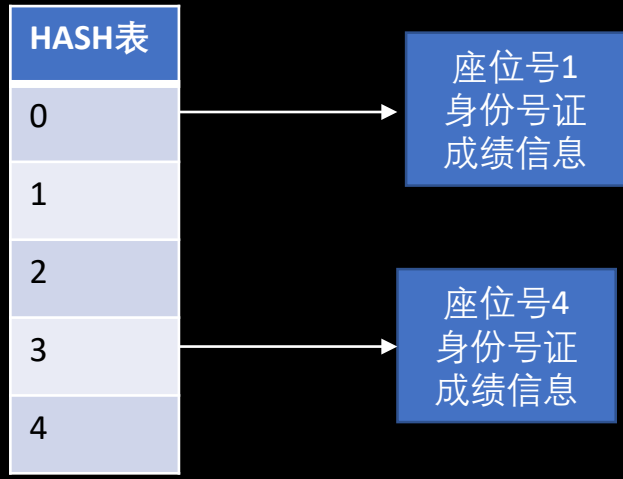
对于1Gbps以太网线路，每秒钟平均可对客户端应答数据包1000000000/8/1706*3= 219812个。对应的每个包平均到达时间是1000000/219812 = 4.55μs。

对于通常的2GHz主频处理器，4.55μs相当于9100个CPU时钟周期，只要不读写硬盘，单个线程在9100个CPU时钟周期内完成处理并响应是很容易实现的。

用户每次查询分数访问2次页面，则每秒钟可以服务219812/3/2=36653人次的查询。

不考虑丢包的情况下，1Gbps的线路，10多秒钟就能让全省的40万考生查询一次成绩。

HASH表



Hash(座位号1)=Hash(座位号6)=0
Hash(座位号4)=3

座位号为关键字建立HASH
选择简单关键字加权求模HASH
40万考生，HASH表100万查询时
70%的查询1次找到
13%的查询2次找到
7%的查询3次找到
4%的查询4次找到
4%的查询5次找到
2%的查询6次找到

考试分数查询实现

①将DBF格式的考生分数库预处理转换成文本文件，一人一行

“座位号_身份证号_返回的HTML内容”

40万考生分数文本文件长度约为400MB，转换大约需要20秒钟

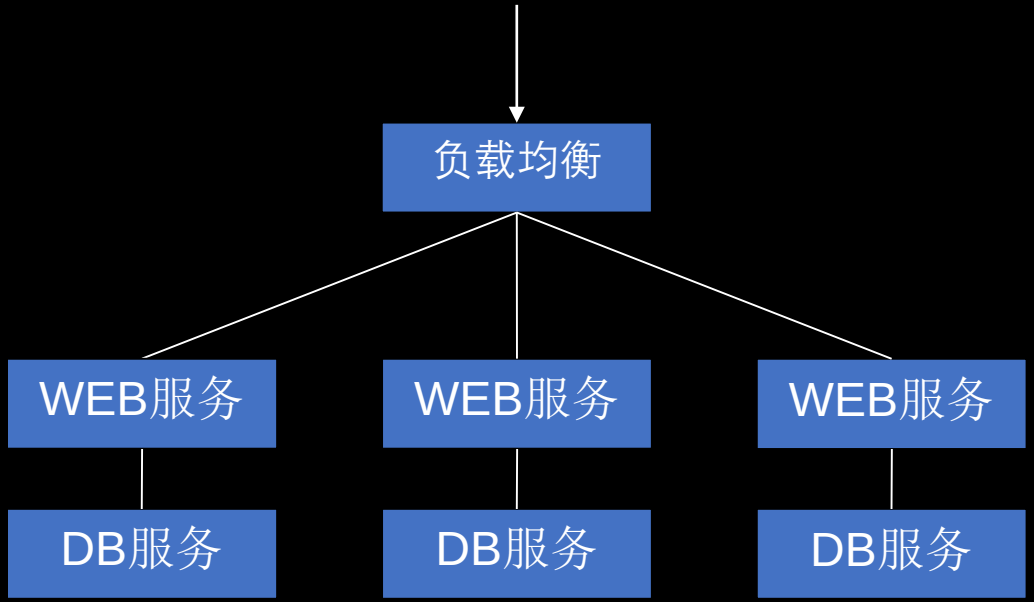
②分数查询应用程序启动时，将转换后的文本文件读入到内存

按考生座位号为key建立hash表，采用链表方式解决hash冲突

程序启动过程大约需要15秒钟

③处理用户的HTTP请求时，如果用户没有提供合法的座位号或身份证号码，返回输入界面的html信息；如果输入了座位号和身份证号码，根据座位号查找hash表，并核对身份证号，给出返回的html内容应答

传统架构



本文架构

结语

功　能	WEB服务程序框架，适合处理幂等简单HTTP请求
优　点	服务器端不保留TCP状态、无并发概念，支持无限连接
优　点	高性能、高可扩展性
缺　点	用C语言开发，框架约1200行代码，应用约500行
缺　点	无法解决重放攻击 （幂等请求不会导致数据被篡改，但会浪费带宽）
缺　点	应答包超过MTU进行分片时，部分用户访问可能出现故障