计算科学与工程中的并行编程技术
Parallel Programming Technology in Computational Science and Engineering

都志辉

清华大学计算机系

Email：duzh@tsinghua.edu.cn

Phone: 62782530

http://hpclab.cs.tsinghua.edu.cn/~duzh

如何加快消息的传递速度？

- 硬件
- 软件

VIA（Virtual Interface Architecture）
用户层网络通信

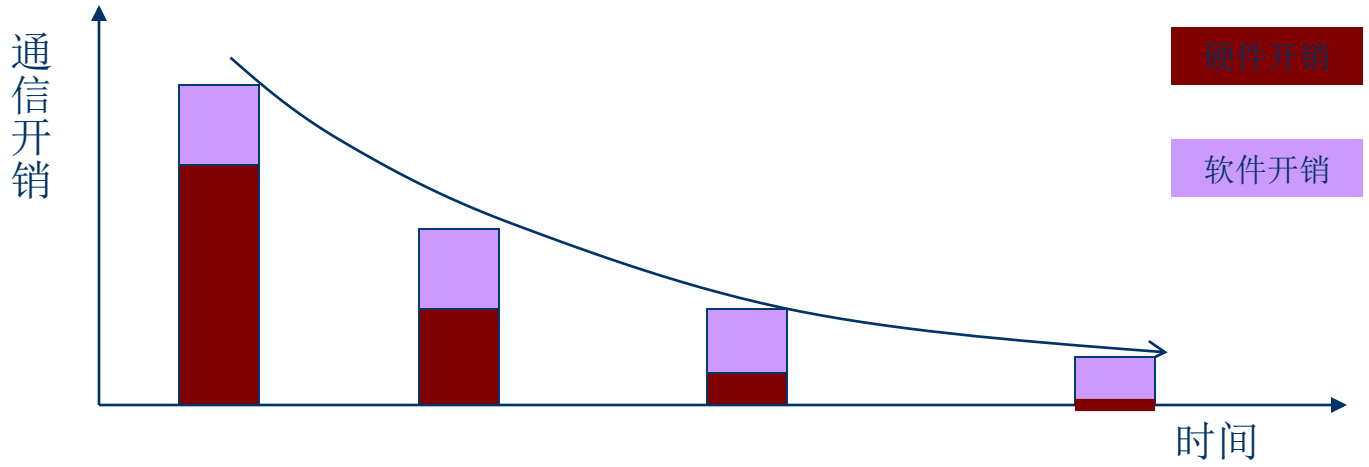
什么是用户层通信

- 用户层通信
  - 通信关键路径不通过操作系统

- 核心层通信
  - 通信关键路径通过操作系统

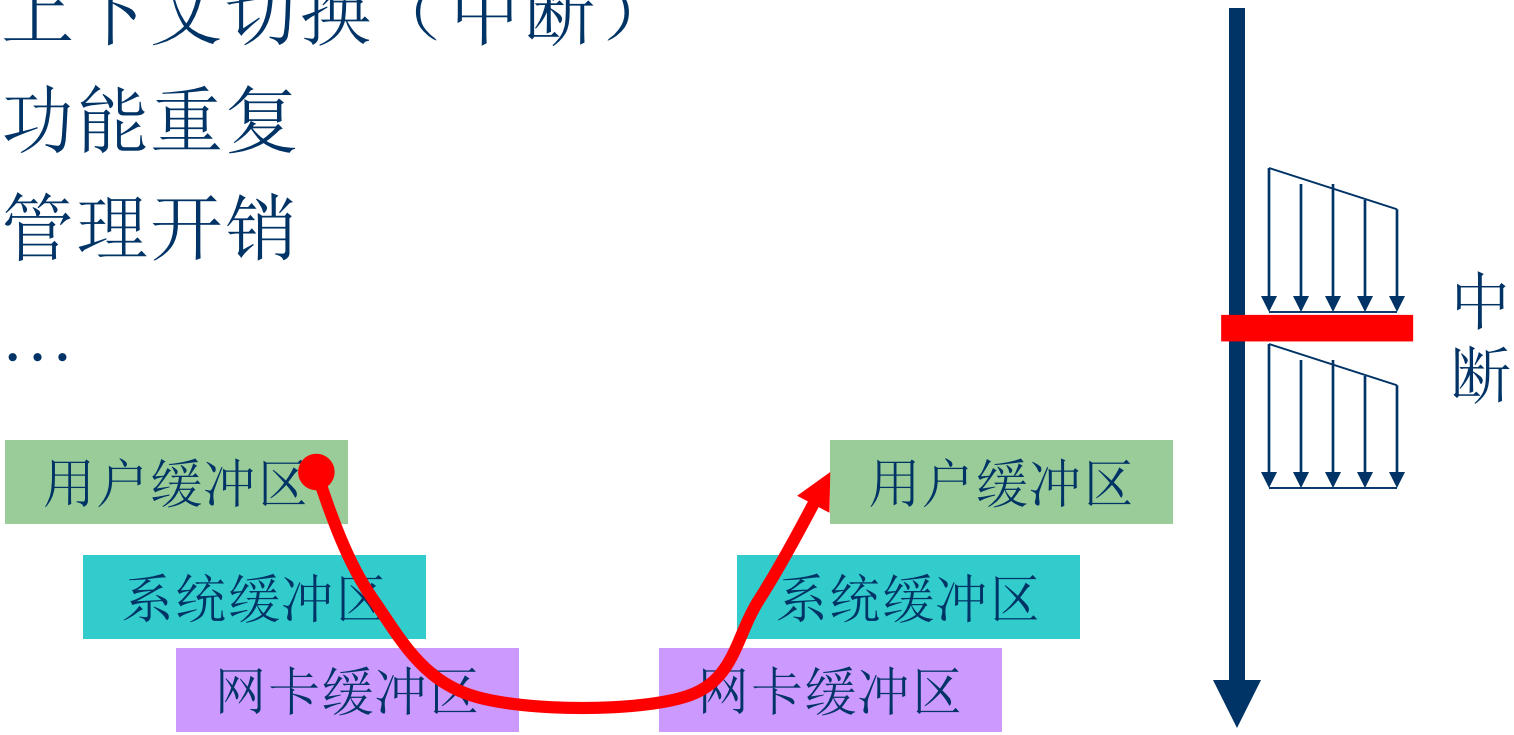
背景

- 网络硬件通信速度与带宽的提高（摩尔定律）
- 网络硬件通信可靠性的提高
- 通信软件与硬件开销对比的变化
  - 软件开销（发送和接受端数据准备与管理的开销）
  - 硬件开销（网络链路上的传输开销）



软件开销的组成

- 缓冲区拷贝
- 上下文切换（中断）
- 功能重复
- 管理开销
- …



如何降低软件开销

- 初级阶段：精简协议
- 后来状况：避开操作系统
  - 内存拷贝
  - 上下文切换
  - 中断指令执行顺序
  - 管理开销
  - …

用户层通信的发展

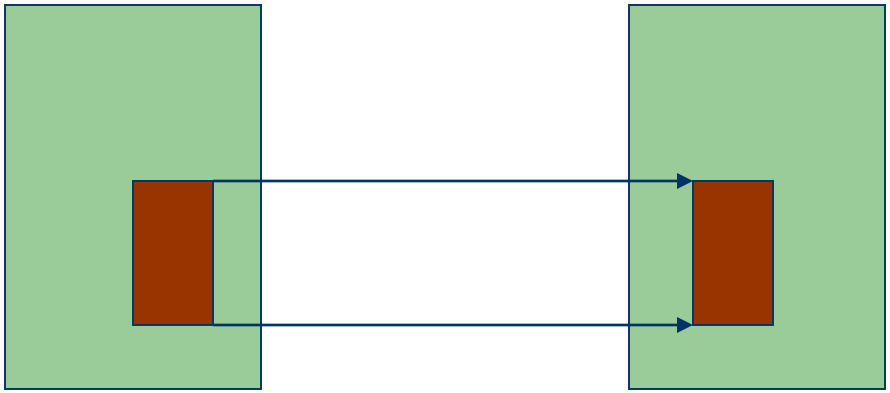
- AM(UC Berkeley,1992)/FM(Illinois,1995)
- SHRIMP (Scalable High-performance Really Inexpensive Multi-Processor,1994 Princeton)
- U-NET(Connell,1995)
- VIA

主动消息AM/快速消息FM

- 特点
  - 消息头部放置一个对该消息处理的句柄
  - SPMD程序

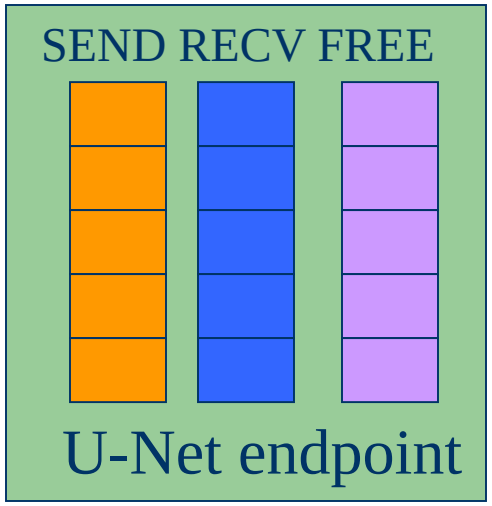
SHRIMP

- 特点
  - 通过虚拟内存映射建立连接
  - 将通信转化为内存访问(VMMC)



U-NET

- 特点
  - 提出并实现了一套通过用户层访问网络的机制
  - 提出了一些基本的概念
    - 访问点
    - 发送/接收队列

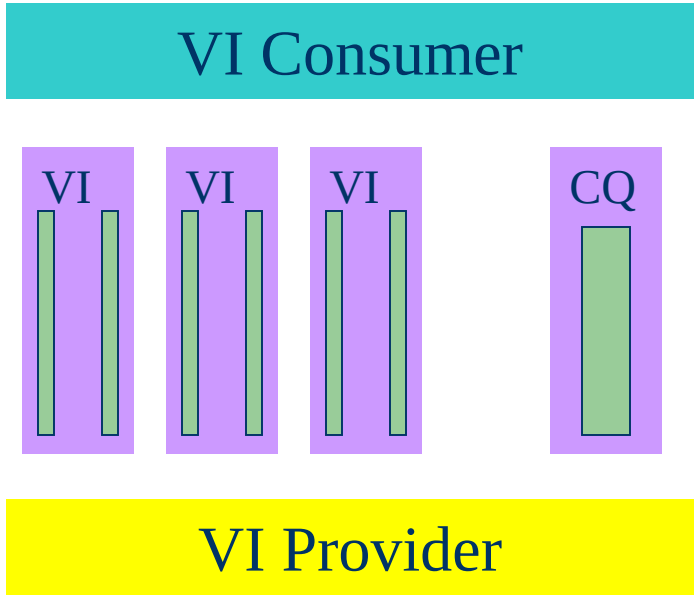


VIA的提出

- 时间 1997
- 促成者Intel, Compaq, Microsoft
- 背景：SAN应用的需求
- 目的：开放的、可扩展的用户层通信标准

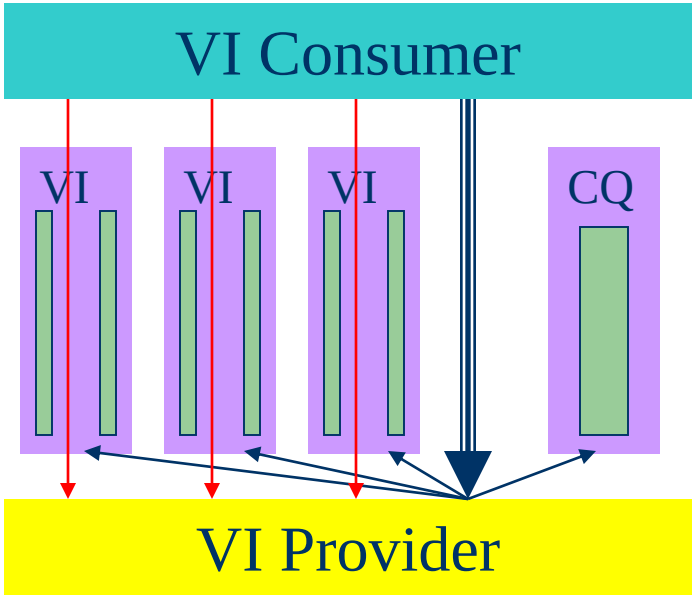
VIA的基本组成

- VI(Virtual Interface)
- CQ(Complete Queue)
- VI Provider
- VI Consumer



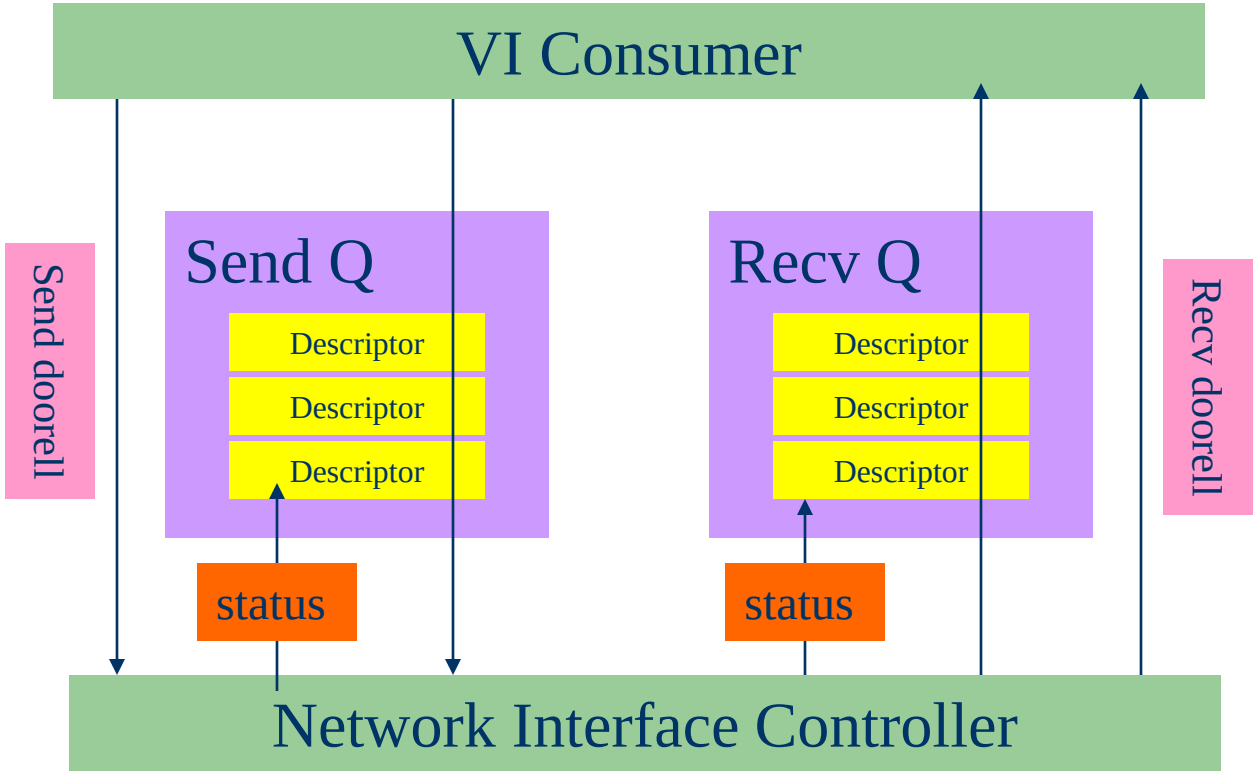
VIA通信步骤

- VI Consumer调用VI Provider提供的功能创建VI(通过操作系统)
- VI Consumer使用创建的VI进行用户层通信（不经过操作系统）

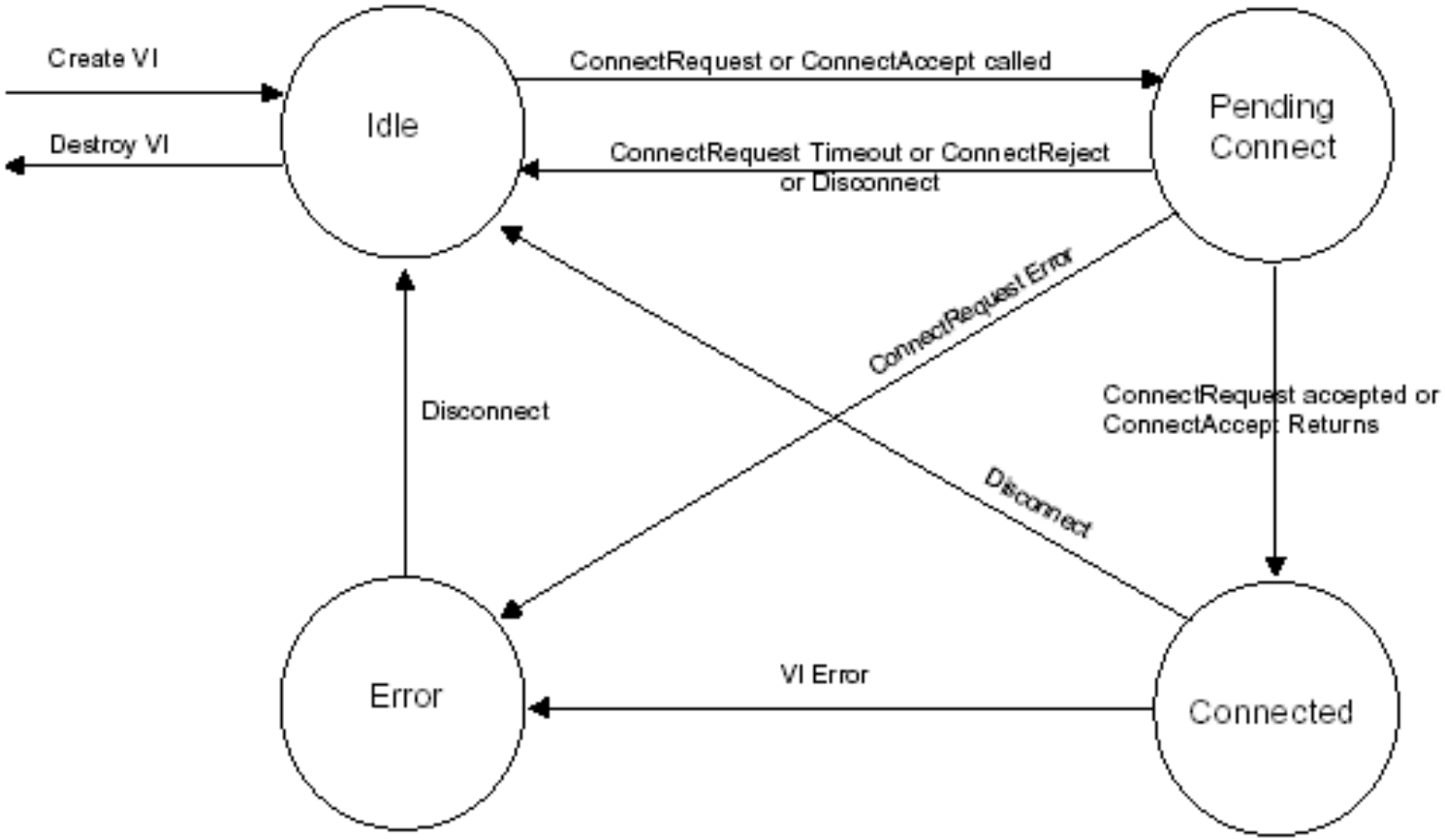


用户层通信

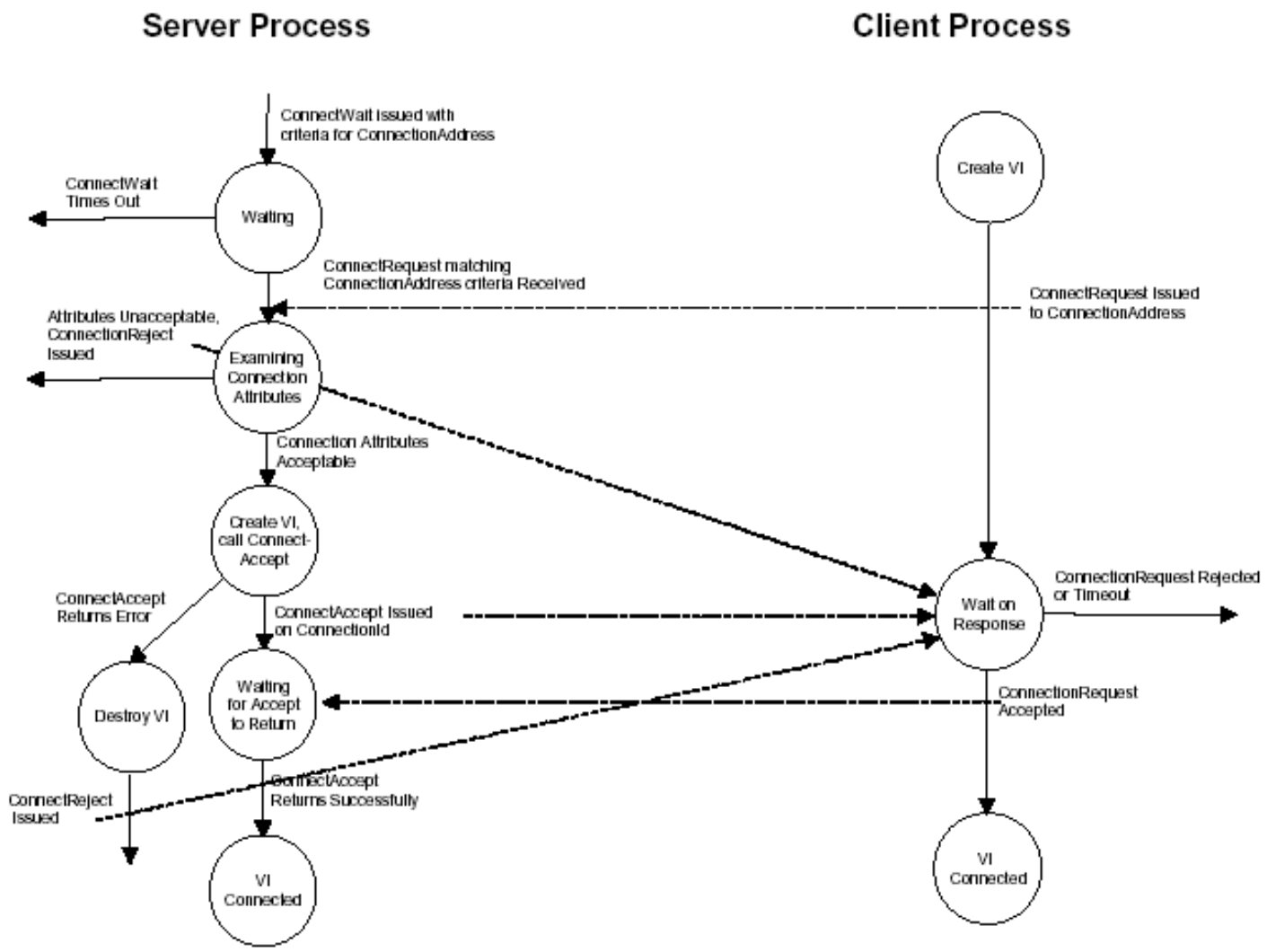
- 示意图


VI Consumer
Send doorell
Send Q
Descriptor
Descriptor
Descriptor
Recv Q
Descriptor
Descriptor
Descriptor
Recv doorell
status
status
Network Interface Controller

VI状态


Create VI
Destroy VI
Idle
ConnectRequest or ConnectAccept called
ConnectRequest Timeout or ConnectReject
or Disconnect
Pending
Connect
ConnectRequest Error
Disconnect
ConnectRequest accepted or
ConnectAccept Returns
Disconnect
Error
VI Error
Connected

VI连接过程


Server Process
Client Process
ConnectWait Issued with criteria for ConnectionAddress
ConnectWait Times Out
Waiting
Create VI
ConnectRequest matching ConnectionAddress criteria Received
ConnectRequest Issued to ConnectionAddress
Attributes Unacceptable, ConnectionReject Issued
Examining Connection Attributes
Connection Attributes Acceptable
Create VI, call Connect-Accept
ConnectAccept Returns Error
ConnectAccept Issued on ConnectionId
ConnectionRequest Rejected or Timeout
Wait on Response
Destroy VI
Waiting for Accept to Return
ConnectionRequest Accepted
ConnectReject Issued
ConnectAccept Returns Successfully
VI Connected
VI Connected

VI描述子的处理

- 申请描述子空间并注册
- 为描述子各部分赋初值
- 将描述子推入工作队列
- 网卡完成对描述子的处理（发送或者接收数据）
- 循环使用该描述子
- 释放描述子空间

通信模式

- SEND/RECV
  - 先启动接收，再启动发送
  - 发送时，用户摁动门铃启动网卡发送
  - 接收时，网卡摁动门铃启动用户接收
- RDMA READ/WRITE
  - 单方面完成读写操作

零拷贝

- 内存注册
  - VI空间
  - 数据空间
  - 描述子空间
- 内存管理

用户层通信程序的框架

- 设备初始化
- 申请并注册空间
- 创建VI与描述子
- 连接VI
- 处理描述子（通信）
- 注销空间
- 断开连接，释放VI

主要调用

- 网卡存取VipOpenNic/ VipCloseNic
- VI的创建与删除VipCreateVi/ VipDestroyVi
- 连接管理
  - VipConnectWait　　VipConnectRequest
  - VipConnectAccept/VipConnectReject
  - VipDisconnect　　VipDisconnect
- 内存保护与注册
  - VipCreatePtag/VipDestroyPtag
  - VipRegisterMem/VipDeregisterMem

通信调用

- 数据传输与完成
  - VipPostSend VipSendDone/VipSendWait
  - VipPostRecv VipRecvDone/VipRecvWait

例子

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
#include <unistd.h>

#include <assert.h>

#include <vipl.h>
```

例子

```
char * amalloc(int size, int align);
void   fatal(char *);

double get_seconds(void) {
  struct timeval t;
  gettimeofday(&t,NULL);
  return
   (double)t.tv_sec+((double)t.tv_usec/(double)1e6);
}
```

例子

```
char usage[] = "usage: vpingpong device r|s remote packets size\n\n"
               "       device          Device in /dev, such as via_lo or via_eth0\n"
               "       r|s             Sender starts pingpong; receiver relays.
   Start receiver first)\n"
               "       remote          Name of remote machine, as in
   /etc/vip_hosts\n"
               "       packets         Number of packets to send (roundtrip)\n"
               "       size            Size (in bytes) of each packet\n";
```

例子

```
int
main(int argc, char *argv[])
{
    double t1, t2;
    char *deviceName, *remoteName;
    int packets;
    int size;
    int sending = 0;
    int i;
```

例子

```
VIP_CONN_HANDLE  connHand;

    VIP_NET_ADDRESS      *localAddr;
    VIP_NET_ADDRESS      *remoteAddr;

    VIP_NIC_HANDLE       nicHand;
    VIP_NIC_ATTRIBUTES   nicAttrs;

    VIP_VI_HANDLE viHand;
    VIP_VI_ATTRIBUTES    viAttrs;
    VIP_VI_ATTRIBUTES    remoteViAttrs;
```

例子

```
VIP_MEM_ATTRIBUTES      memAttrs;
   VIP_PROTECTION_HANDLE ptag;

   VIP_DESCRIPTOR       *descp;
   VIP_DESCRIPTOR       *desc;
   char                 *buff;

   VIP_RETURN    retval;


   VIP_MEM_HANDLE       descHand, buffHand;
```

例子

```
if(argc != 6) {
 fatal(usage);
    }
    switch(argv[2][0]) {
    case 's':
 sending = 1;
 break;
```

例子

```
case 'r':
 sending = 0;
 break;
    default:
 fatal(usage);
    }
```

例子

```
deviceName = argv[1];
   remoteName = argv[3];
   packets = atoi(argv[4]);
   size = atoi(argv[5]);
   printf("pid = %d\n",getpid());
   if(VipOpenNic(deviceName, &nicHand) !=
 VIP_SUCCESS) {
 fatal("VipOpenNic failed\n");
   }
```

例子

```
 if(VipNSInit(nicHand, NULL) != VIP_SUCCESS) {
  fatal("VipNSInit failed\n");
    }
    if(VipQueryNic(nicHand, &nicAttrs) != VIP_SUCCESS) {
  fatal("VipQueryNic failed\n");
    }
localAddr =
  malloc(sizeof(VIP_NET_ADDRESS)+nicAttrs.NicAddressLen);
    localAddr->HostAddressLen = nicAttrs.NicAddressLen;
```

例子

```
localAddr->DiscriminatorLen = 0;
   memcpy(localAddr->HostAddress,
     nicAttrs.LocalNicAddress,
 nicAttrs.NicAddressLen);
   remoteAddr =
 malloc(sizeof(VIP_NET_ADDRESS)+nicAttrs.NicAddres
 sLen);
   memset(remoteAddr, 0, sizeof(VIP_NET_ADDRESS));
   remoteAddr->HostAddressLen =
 nicAttrs.NicAddressLen;
   remoteAddr->DiscriminatorLen = 0;
```

例子

```
if(VipNSGetHostByName(nicHand, remoteName,
 remoteAddr, 0) != VIP_SUCCESS) {
 fatal("VipNSGetHostByName failed\n");
   }
 printf("localAddr->HostAddress:%d, remoteAddr-
 >HostAddress:%d\n", localAddr->HostAddress[0],
 remoteAddr->HostAddress[0]);
   desc = (VIP_DESCRIPTOR *)
 amalloc(2*sizeof(VIP_DESCRIPTOR),
                          VIP_DESCRIPTOR_ALIGNMENT);
   buff = amalloc(2*size, 32);
```

例子

```
if(VipCreatePtag(nicHand, &ptag) !=
 VIP_SUCCESS) {
 fatal("Failed to create ptag\n");
   }
   memAttrs.Ptag = ptag;
   memAttrs.EnableRdmaWrite = VIP_FALSE;
   memAttrs.EnableRdmaRead = VIP_FALSE;
```

例子

```
 if(VipRegisterMem(nicHand, desc,
  2*sizeof(VIP_DESCRIPTOR),
              &memAttrs, &descHand) != VIP_SUCCESS) {
  fatal("Failed to register descriptors\n");
    }
if(VipRegisterMem(nicHand, buff, (2*size)+1,
              &memAttrs, &buffHand) != VIP_SUCCESS) {
  fatal("Failed to register buffers\n");
    }
viAttrs.ReliabilityLevel = VIP_SERVICE_UNRELIABLE;
```

例子

```
viAttrs.Ptag = ptag;
   viAttrs.EnableRdmaWrite = VIP_FALSE;
   viAttrs.EnableRdmaRead = VIP_FALSE;
   viAttrs.QoS = 0;
   viAttrs.MaxTransferSize = size;

   if(VipCreateVi(nicHand, &viAttrs, NULL, NULL,
 &viHand) != VIP_SUCCESS) {
 fatal("Failed to create VI\n");
   }
```

例子

```
for(i=0; i<2; i++) {
 desc[i].CS.Length = size;
 desc[i].CS.Status = 0;
 desc[i].CS.Control = VIP_CONTROL_OP_SENDRECV |
 VIP_CONTROL_IMMEDIATE;
       desc[i].CS.SegCount = 1;
       desc[i].CS.ImmediateData = 0x1234;
 desc[i].DS[0].Local.Data.Address = &buff[i*size];
 desc[i].DS[0].Local.Handle = buffHand;
 desc[i].DS[0].Local.Length = size;
   }
```

例子

```
if(sending) {
 if(VipConnectRequest(viHand, localAddr,
  remoteAddr, 50000, &remoteViAttrs) !=
 VIP_SUCCESS) {
     fatal("VipConnectRequest failed\n");
 }
 for(i=0; i < 5; i++) {
     VipPostRecv(viHand, &desc[0], descHand);
     VipPostSend(viHand, &desc[1], descHand);
     VipSendWait(viHand, VIP_INFINITE, &descp);
     VipRecvWait(viHand, VIP_INFINITE, &descp);
 }
```

例子

```
t1 = get_seconds();
for(i=0; i < packets; i++) {
    VipPostRecv(viHand, &desc[0], descHand);
    VipPostSend(viHand, &desc[1], descHand);
    VipSendWait(viHand, VIP_INFINITE, &descp);
    VipRecvWait(viHand, VIP_INFINITE, &descp);
}
t2 = get_seconds();
```

例子

```
printf("RT: %f (us)\tLat: %f (us) BW: %f
(MB/s)\n",
((t2-t1)*1e6)/packets, (((t2-t1)*1e6)/packets)/2,
        ((packets*size)/(((t2-t1)*1e6)/2)));
  } else {
VipPostRecv(viHand, &desc[0], descHand);
descp = &desc[1];
printf("now waiting for connecting\n");
if(VipConnectWait(nicHand, localAddr,
VIP_INFINITE, remoteAddr, &remoteViAttrs,
  &connHand) != VIP_SUCCESS) {
fatal("VipConnectWait failed\n");
}
```

例子

```
if(VipConnectAccept(connHand, viHand) !=
VIP_SUCCESS) {
    fatal("VipConnectAccept failed\n");
}
for(i=0; i < 5; i++) {
    VipPostRecv(viHand, descp, descHand);
    VipRecvWait(viHand, VIP_INFINITE, &descp);
    VipPostSend(viHand, descp, descHand);
    VipSendWait(viHand, VIP_INFINITE, &descp);
}
```

例子

```
    for(i=0; i<packets; i++) {
        VipPostRecv(viHand, descp, descHand);
        VipRecvWait(viHand, VIP_INFINITE, &descp);
        VipPostSend(viHand, descp, descHand);
        VipSendWait(viHand, VIP_INFINITE, &descp);
    } }
      VipDeregisterMem(nicHand, buff, buffHand);
      VipDeregisterMem(nicHand, desc, descHand);
      VipDisconnect(viHand);
      VipDestroyVi(viHand);
      exit(0);
}
```

例子

```
char *
amalloc(int size, int align)
{
    char *ptr;
    unsigned mask = align - 1;
    ptr = malloc(size + align - 1);
    if(ptr != NULL && ((unsigned)ptr & mask) != 0)
  {
  ptr = (char *) (((unsigned)ptr + mask) & ~mask);
    }
    return ptr;
}
```

例子

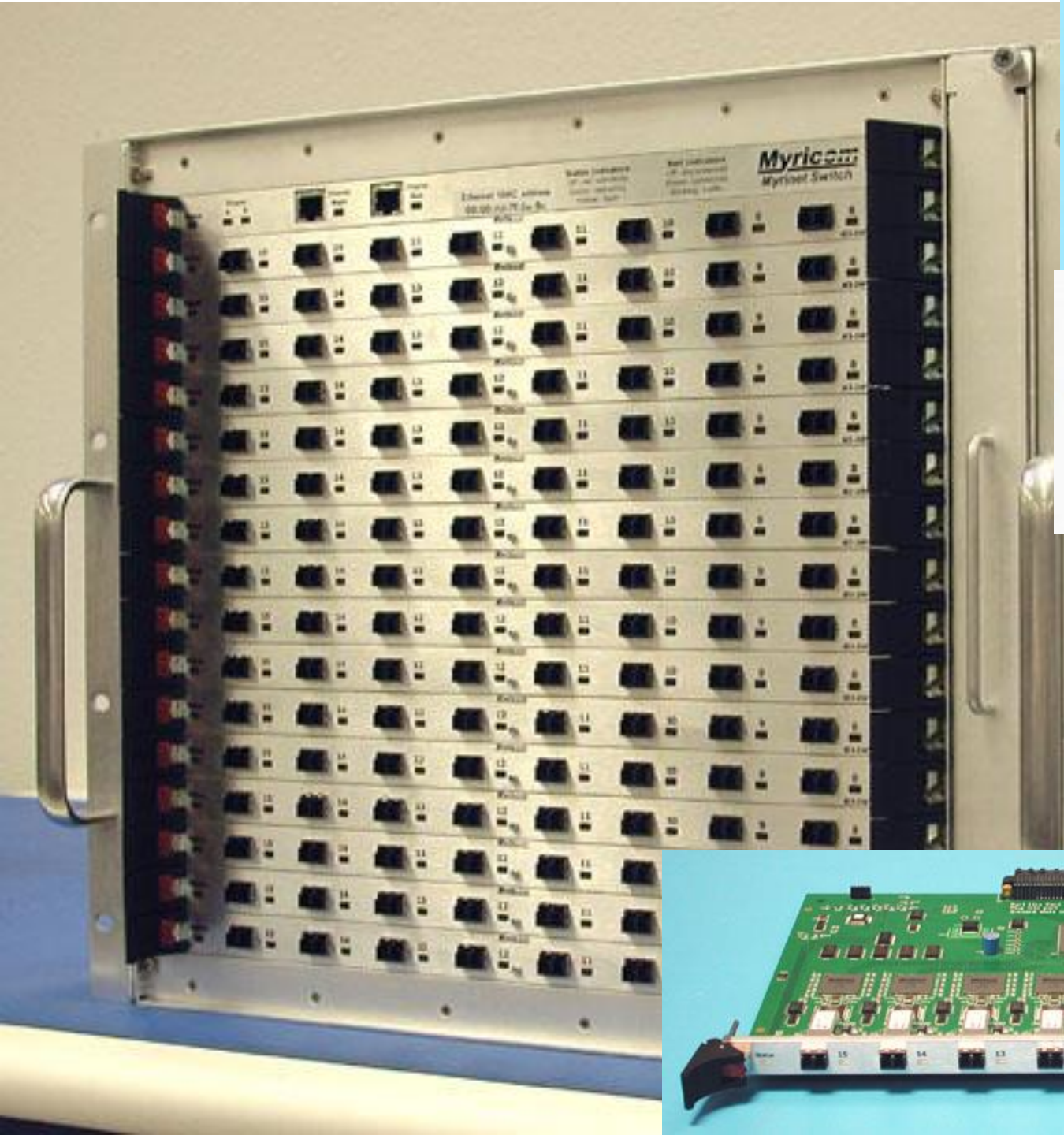
```
void fatal(char *msg)
{
  fprintf(stderr, "%s\n", msg);
  exit(-1);
}
```

用户层与核心层通信的对比

对比	用户层	核心层
通信空间	用户层系统/操作系统	操作系统
内存拷贝次数	0/1	多次
发送接收是否需要中断	不需要	需要
性能	高	低

集群式系统的高速互连网络

- Myrinet/GM（专用系统）
- Infiniband/VAPI（工业标准）
- Quadrics/Elanlib（专用系统）
- Gigabit Ethernet（IEEE 标准）


Myricom
Myrinet Switch


Myricom
Lanai 2XP


Myricom

性能

Myrinet-2000 User-level Performance	
MX or MPI latency	3.2µs (D-card NICs) 2.6µs (E- or F-card NICs)
MX or MPI unidirectional data rate	247 MBytes/s (one-port NICs) 495 MBytes/s (two-port NICs)
TCP/IP data rate (MX ethernet emulation)	1.98 Gbits/s (one-port NICs) 3.95 Gbits/s (two-port NICs)

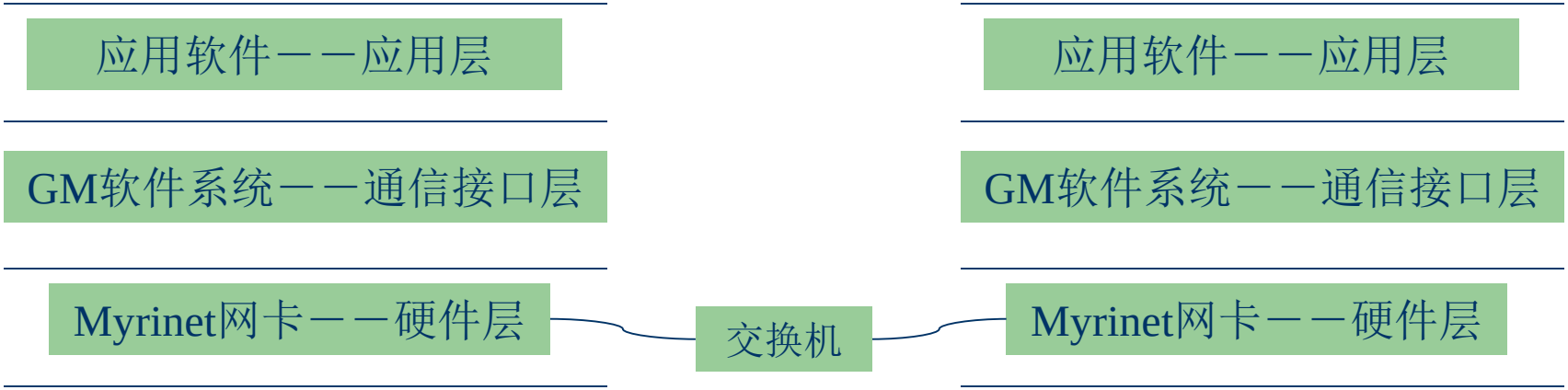
2004年11月的TOP500统计

193 (38.6%)

Rank	Rmax (Gflops)	Site	Name, if any	Country/year	Manufacturer
4	20530	Barcelona Supercomputer Center	MareNostrum	Spain/2004	IBM
10	9819	NCSA	Tungsten	United States/2003	Dell
13	8770	Army Research Laboratory	John Von Neumann	USA/2004	Linux Networx
17	8061	Shanghai Supercomputer Center	Red Grid	China/2004	Dawning
18	8051	Los Alamos National Laboratory	Lightning	United States/2003	Linux Networx
22	7215	NCSA	Teragrid	United States/2004	IBM
23	7185	Army Research Laboratory		United States/2004	IBM
28	6155	Grid Technology Research Center, AIST	AIST Super Cluster P-32	Japan/2004	IBM
31	5510	University of Southern California	HPC	United States/2004	IBM, Sun, self-made
37	4298	Caltech/JPL	Cosmos	United States/2004	Dell
40	4152	University of Texas / Texas Advanced Computing Center	Lonestar	United States/2004	Dell-Cray
57	3379.5	Walt Disney Feature Animation	The Hive	United States/2004	HP
58	3337	Forecast Systems Laboratory - NOAA	Jet	United States/2002	Aspen/HPTi
60	3231	CINECA		Italy/2004	IBM
61	3231	Nankai University / Institute of Scientific Computing	Nankai Star	China/2004	IBM
62	3152	UCSD/San Diego Supercomputer Center	Teragrid	United States/2004	IBM
64	3067	Korea Institute of Science and Technology		Korea, South/2003	IBM
70	2880	Sandia National Laboratories	Institutional Cluster	United States/2004	HP
71	2880	Sandia National Laboratories	Institutional Cluster	United States/2004	HP
81	2223.15	Sandia National Laboratories		United States/2004	IBM
82	2207	Louisiana State University	SuperMike	United States/2002	Atipa
83	2200	Sandia National Laboratories		United States/2004	HP
84	2200	Sandia National Laboratories		United States/2004	HP
97	2044	Oracle Corporation		United States/2004	HP

总体结构

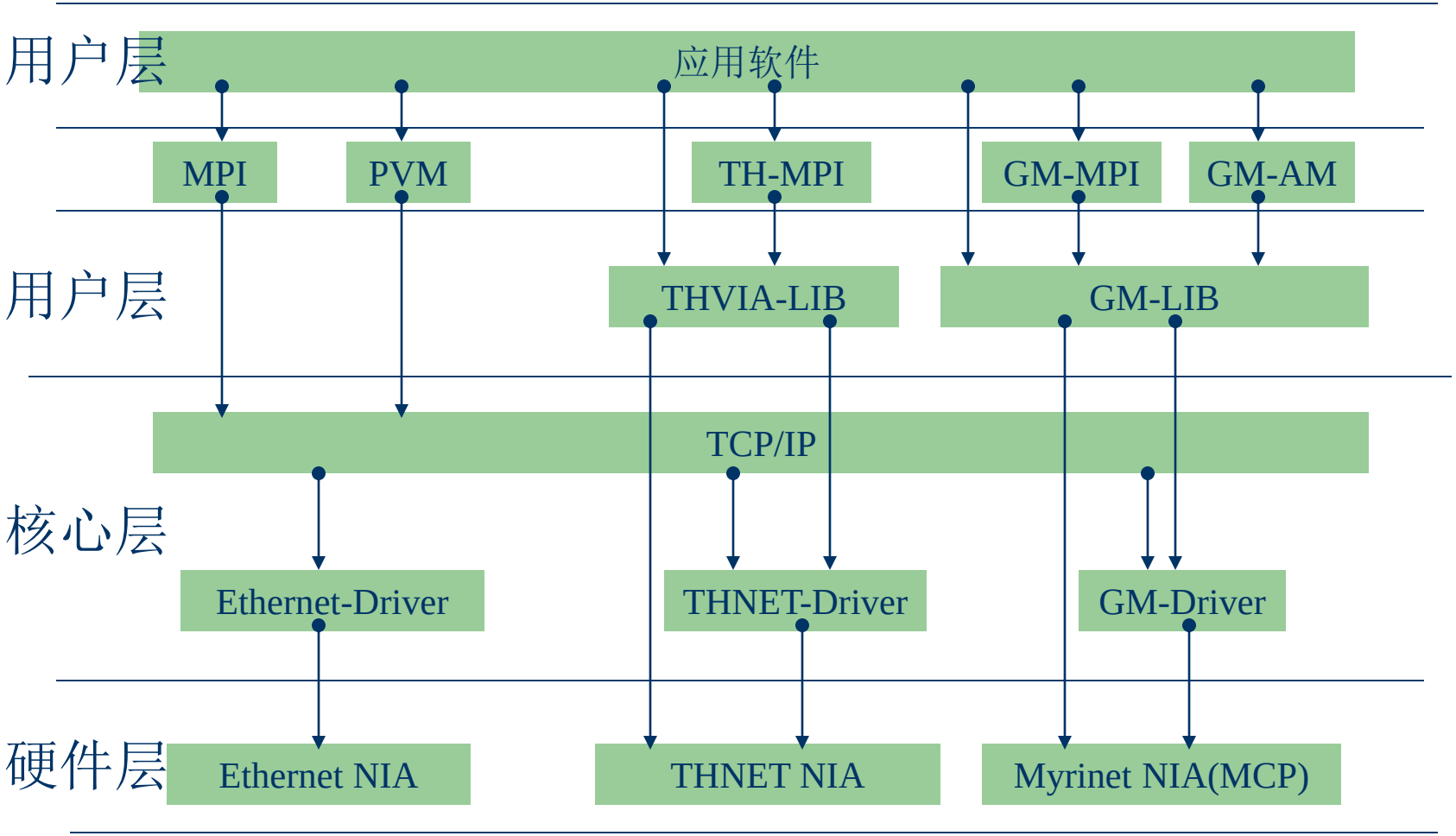
- GM/MX在整个通信网络中的地位



软件组成

- 通信库（gmlib.a）--用户层
- 驱动程序（gm）--核心层
  - 主机部分
  - 网卡部分（MCP）

与相关软件的对比


用户层
应用软件
MPI
PVM
TH-MPI
GM-MPI
GM-AM
用户层
THVIA-LIB
GM-LIB
TCP/IP
核心层
Ethernet-Driver
THNET-Driver
GM-Driver
硬件层
Ethernet NIA
THNET NIA
Myrinet NIA(MCP)

Middleware over GM

- **MPICH-GM**
- **VI-GM**
- **PVM-GM**
- **Sockets-GM**
- **DAPL-GM**
  - （Direct Access Provider Library to allow Myrinet networks to be used for applications using the Direct Access Transport）
- **Myrinet Protocol Module for Sun HPC ClusterTools**

实现平台

- Linux
- Windows
- Solaris
- AIX
- Mac OS X
- Tru64
- FreeBSD
- VxWorks

GM通信的特点

- 事件驱动
- 接收方预先准备DMA缓冲区
- 通信优先级
- token管理

查询接收（非阻塞）

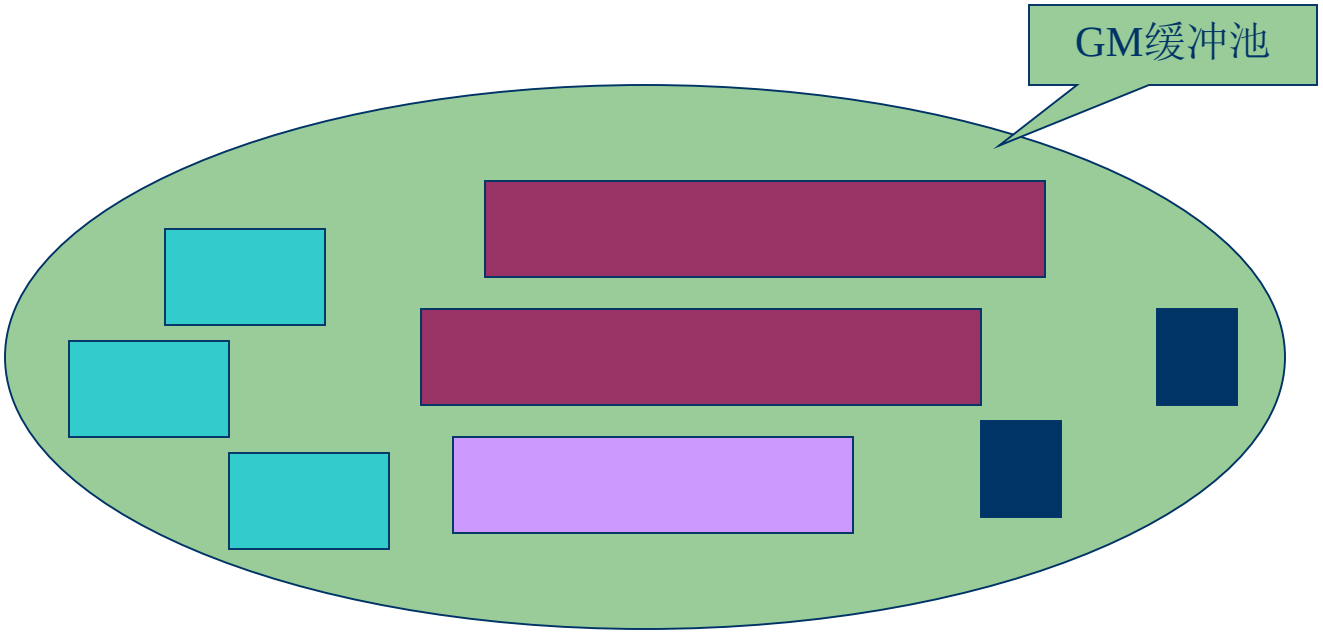
```
while (1) {
event=gm_recv(port);
switch(event->recv.type)
case :
        break;
case:
        break;
}
```

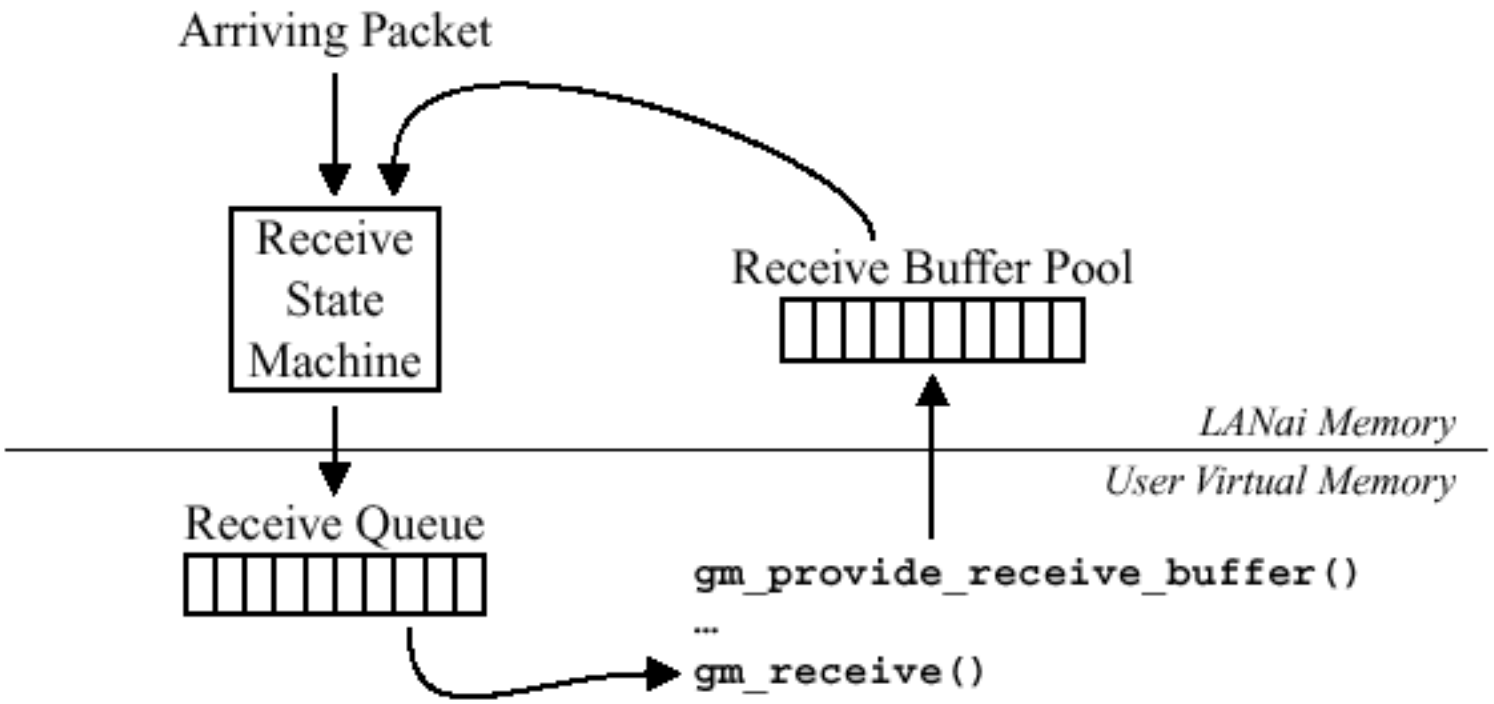
阻塞接收

- gm_blocking_receive()
  - 若无消息先等待一阵
- gm_blocking_receive_no_spine()
  - 若无消息立即睡眠

提供接收缓冲区（形成缓冲池）

- gm_dma_calloc()
- gm_provide_receive_buffer()

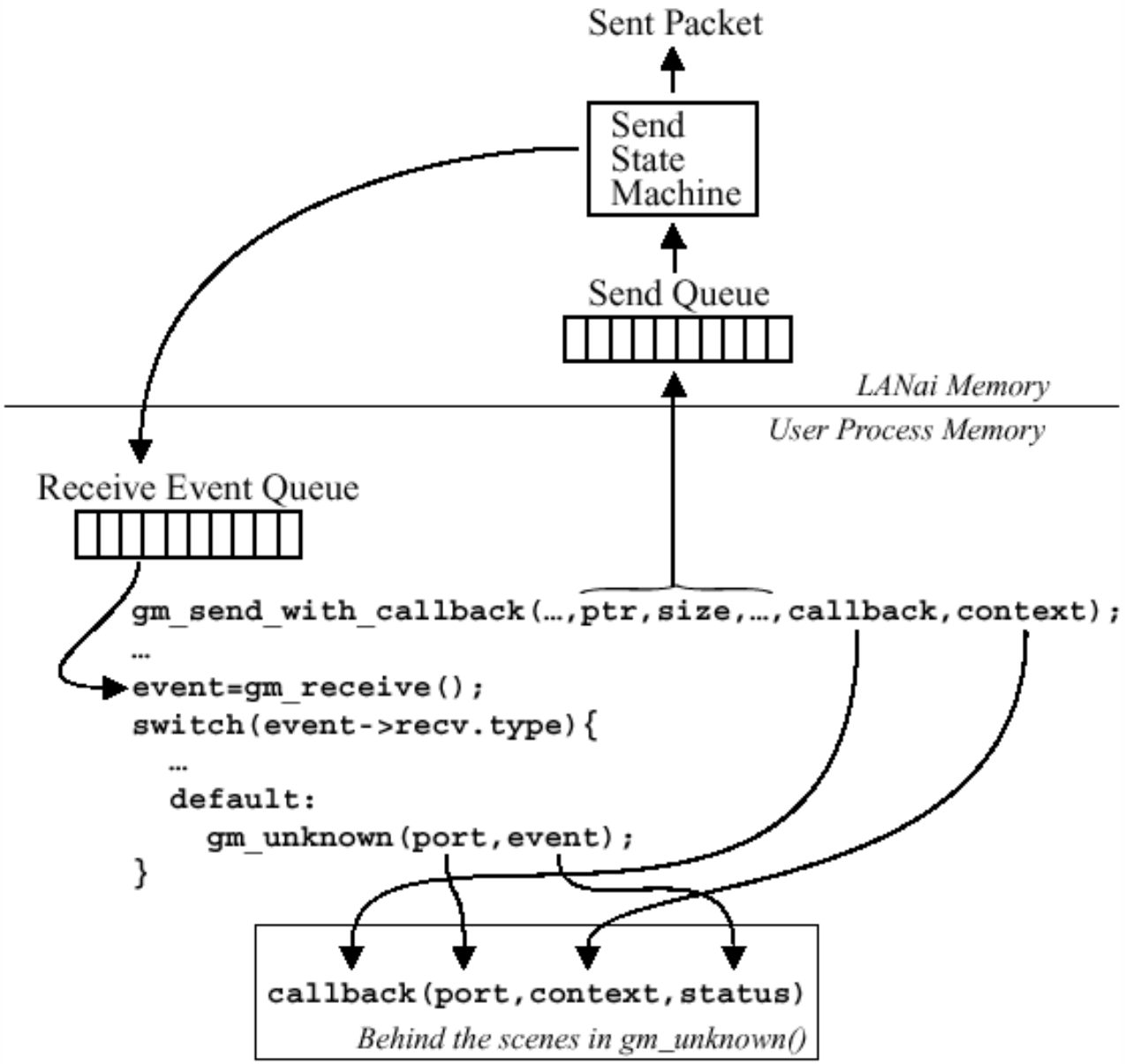



Arriving Packet
Receive State Machine
Receive Buffer Pool
LANai Memory
User Virtual Memory
Receive Queue
gm_provide_receive_buffer()
…
gm_receive()

Receiving

发送（非阻塞）

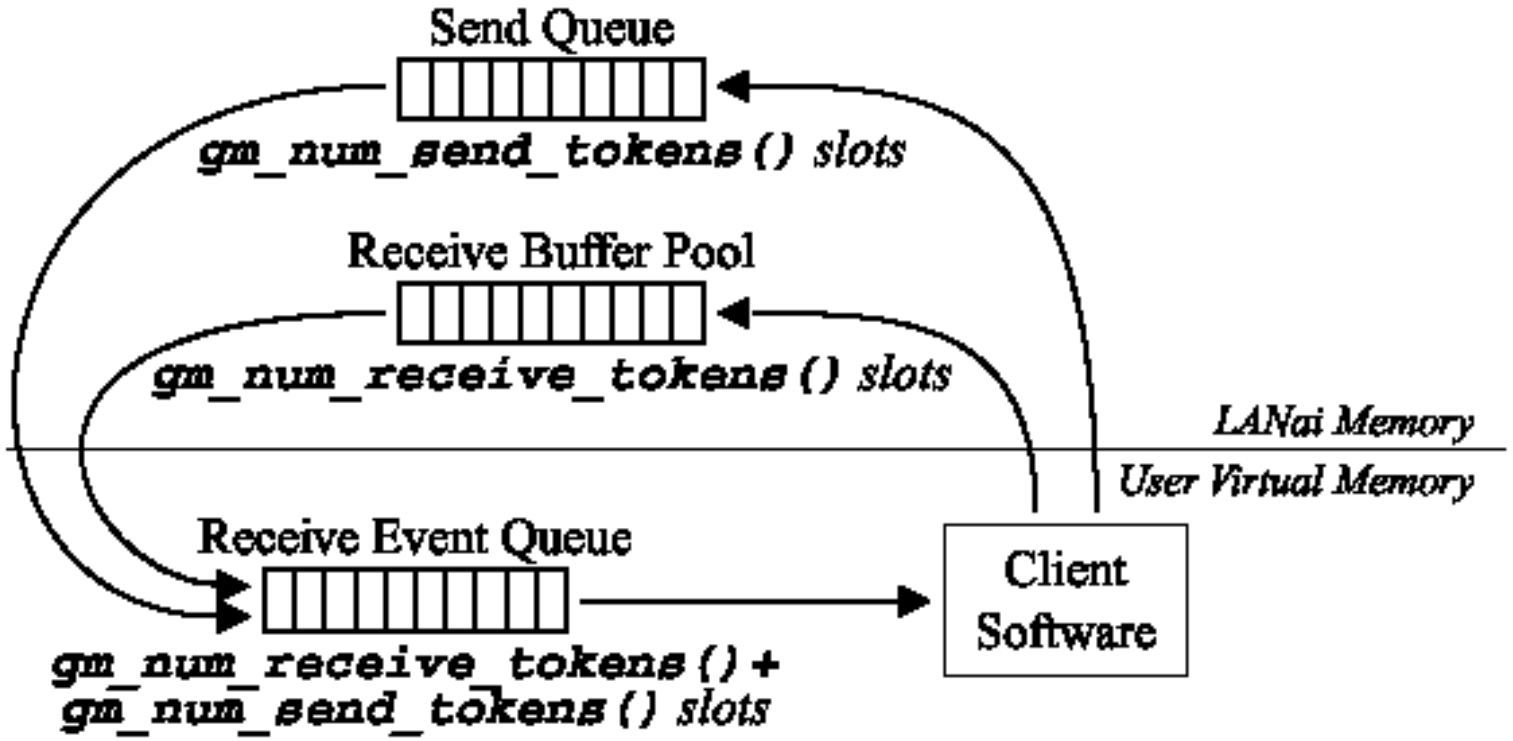
- gm_send()
  - 指定优先级GM_LOW_PRIORITY /GM_HIGH_PRIORITY
- gm_send_with_callback()
  - 指定优先级GM_LOW_PRIORITY /GM_HIGH_PRIORITY


Sent Packet
Send State Machine
Send Queue
LANai Memory
User Process Memory
Receive Event Queue
gm_send_with_callback(…,ptr,size,…,callback,context);
…
event=gm_receive();
switch(event->recv.type){
…
default:
gm_unknown(port,event);
}
callback(port,context,status)
Behind the scenes in gm_unknown()

Sending

TOKEN的管理

- 发送前必须得到发送token
  - gm_alloc_send_token (port, GM_LOW_PRIORITY)
- 接收前也必须保证提供接收token
  - gm_provide_receive_buffer()



User Token Flow

程序流程

- 准备
  - 初始化gm_init() gm_open ( )
  - 申请DMA空间与令牌gm_dma_calloc ( )
    gm_alloc_send_token ( )
    gm_provide_receive_buffer ( )
- 发送/接收
  - gm_send_with_callback (）
  - gm_receive ()

gm在LINUX下的安装

- 解包
  - tar zxvf gm-1.5_Linux.tar.gz
- 自动配置
  - cd gm;./configure
- 编译
  - make
- 装载驱动
  - cd binary;./GM_INSTALL
- 网络映射
  - cd ~/gm/binary/sbin;./mapper active.args

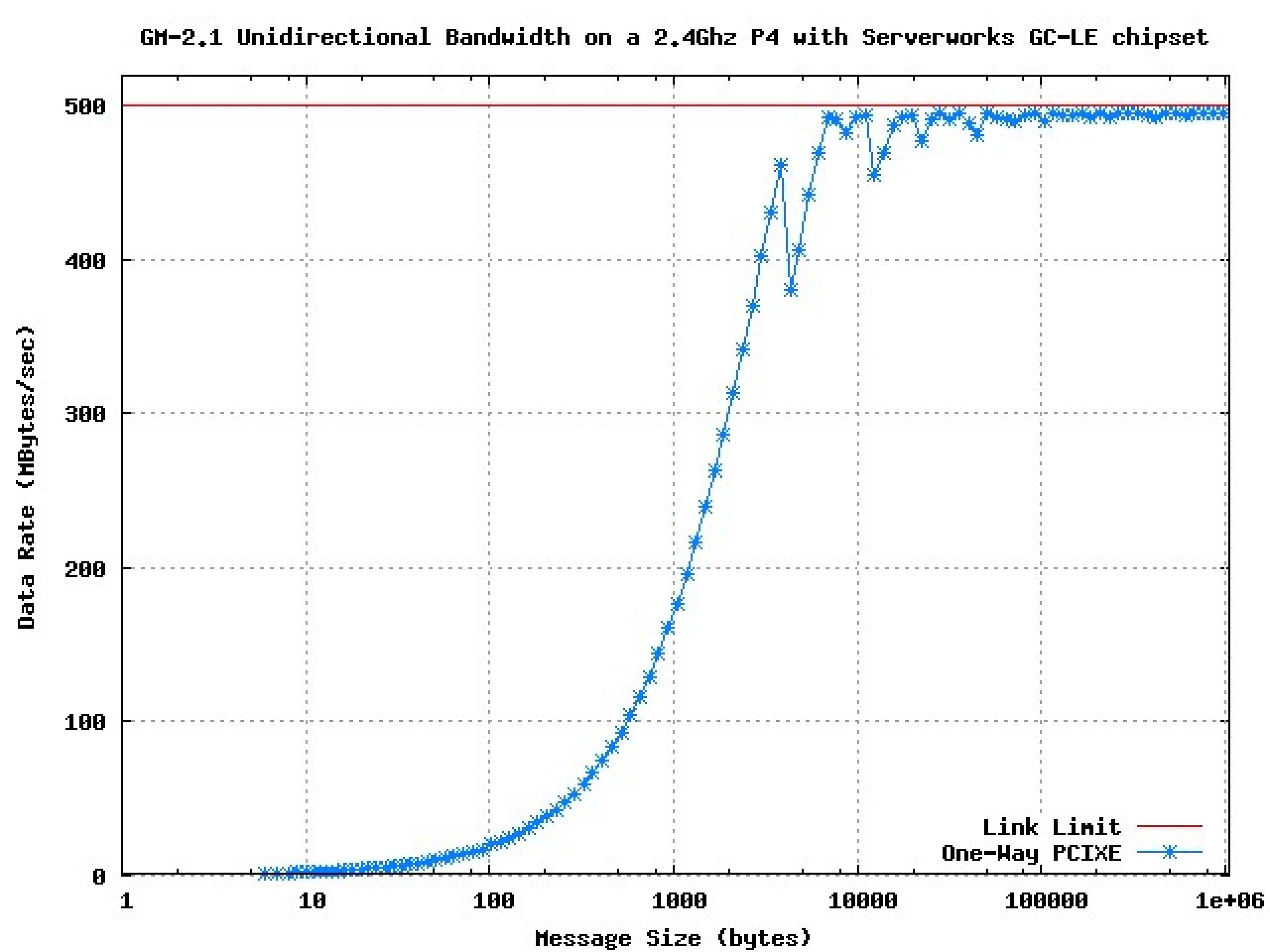
gm在LINUX下的安装（续）

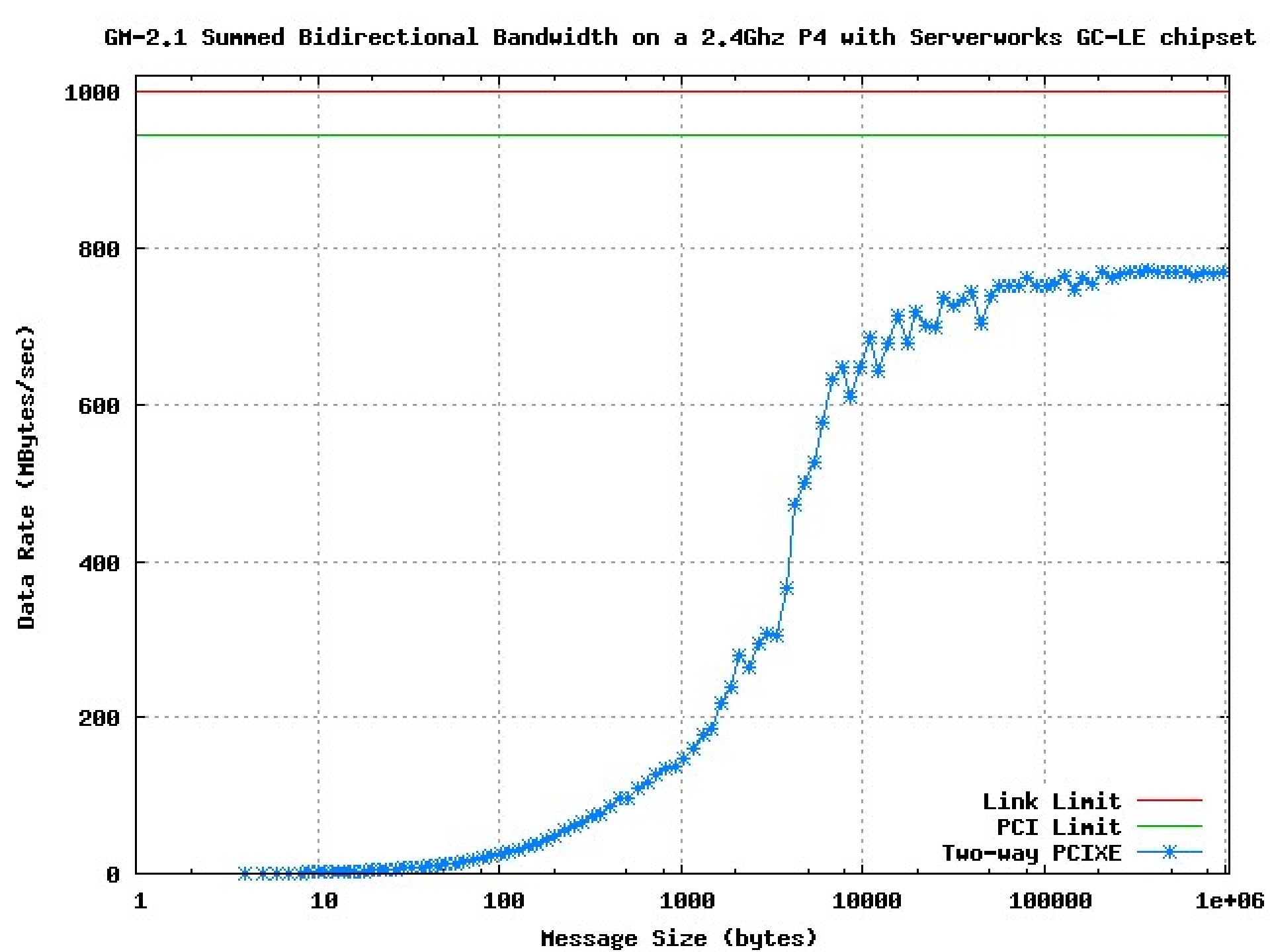
- 自动配置脚本
  - 在~/gm/binary/share/gm中修改IP地址
  - 拷贝到/etc/rc.d/init.d
  - 建立连接
- 驱动准备
  - 将~/gm/binary/sbin/gm拷贝到/lib/modules/版本号/net/.
- 在一台机器上加上mapper启动命令

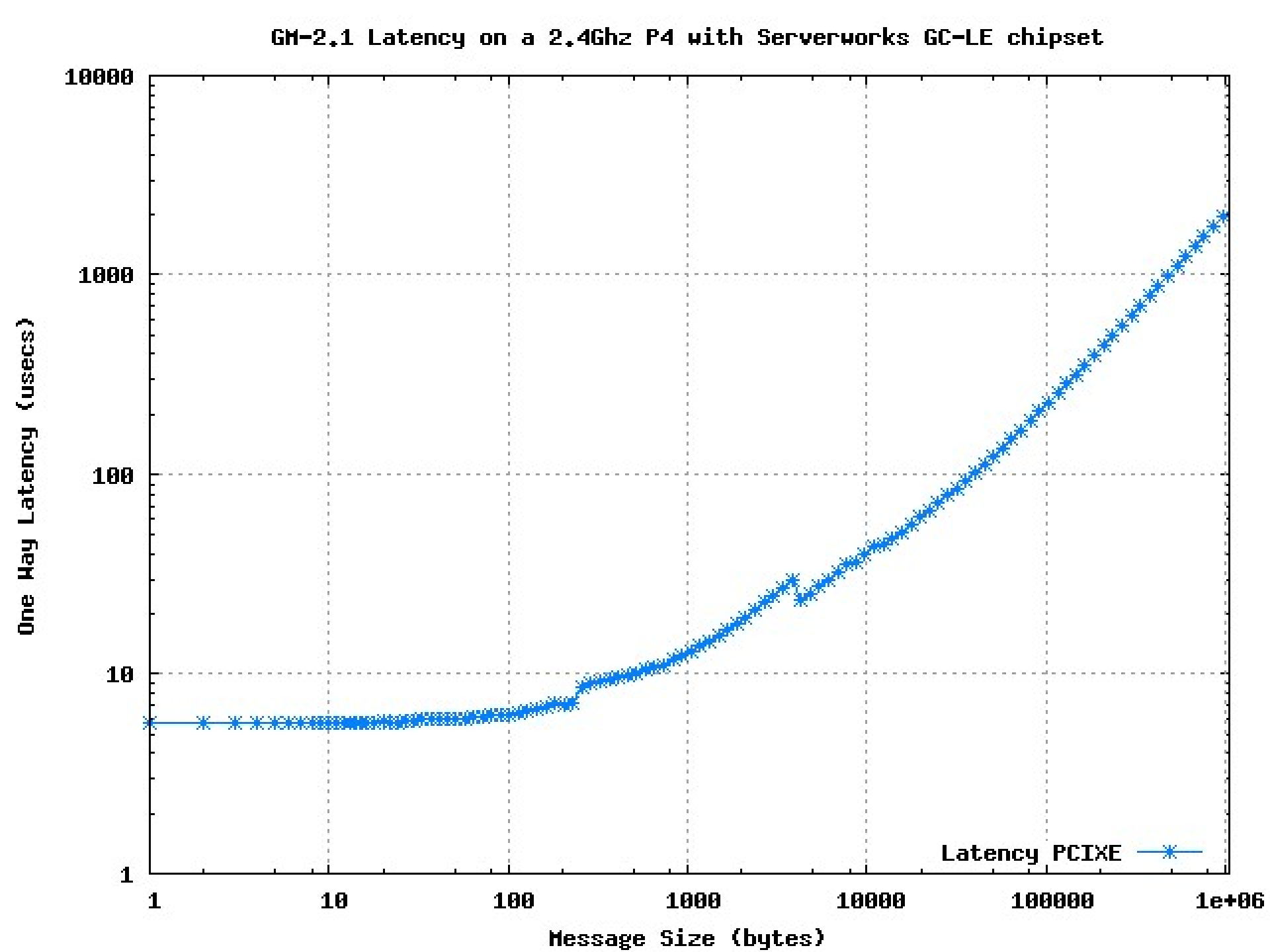
gm在Win2000下的安装

- 解包
- 运行setup.exe
- 重新安装驱动程序
- 通过增加myrinet网卡设备的service来启动mapper

性能


GM-2.1 Unidirectional Bandwidth on a 2.4Ghz P4 with Serverworks GC-LE chipset
Data Rate (MBytes/sec)
Message Size (bytes)
0
100
200
300
400
500
1
10
100
1000
10000
100000
1e+06
Link Limit
One-Way PCIXE

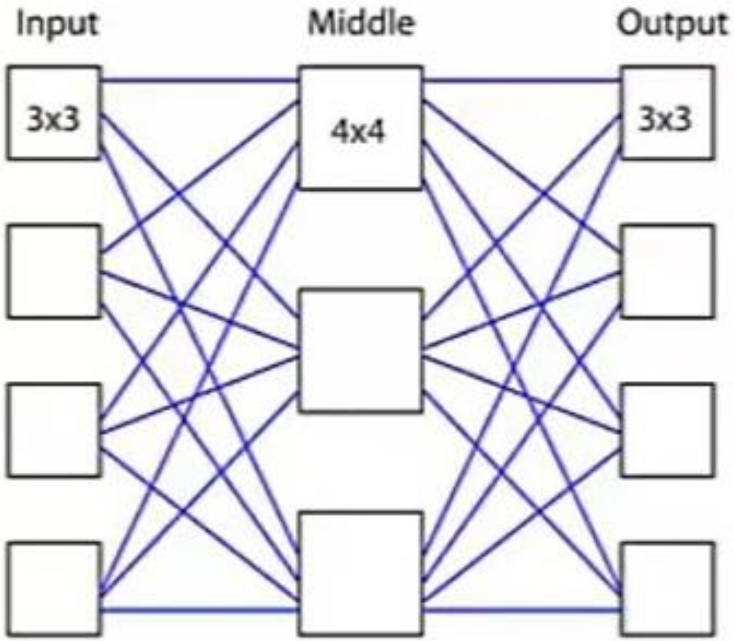

GM-2.1 Summed Bidirectional Bandwidth on a 2.4Ghz P4 with Serverworks GC-LE chipset
Data Rate (MBytes/sec)
Message Size (bytes)
0
200
400
600
800
1000
1
10
100
1000
10000
100000
1e+06
Link Limit
PCI Limit
Two-way PCIXE


GM-2.1 Latency on a 2.4Ghz P4 with Serverworks GC-LE chipset
10000
1000
100
10
1
One Way Latency (usecs)
1
10
100
1000
10000
100000
1e+06
Message Size (bytes)
Latency PCIXE

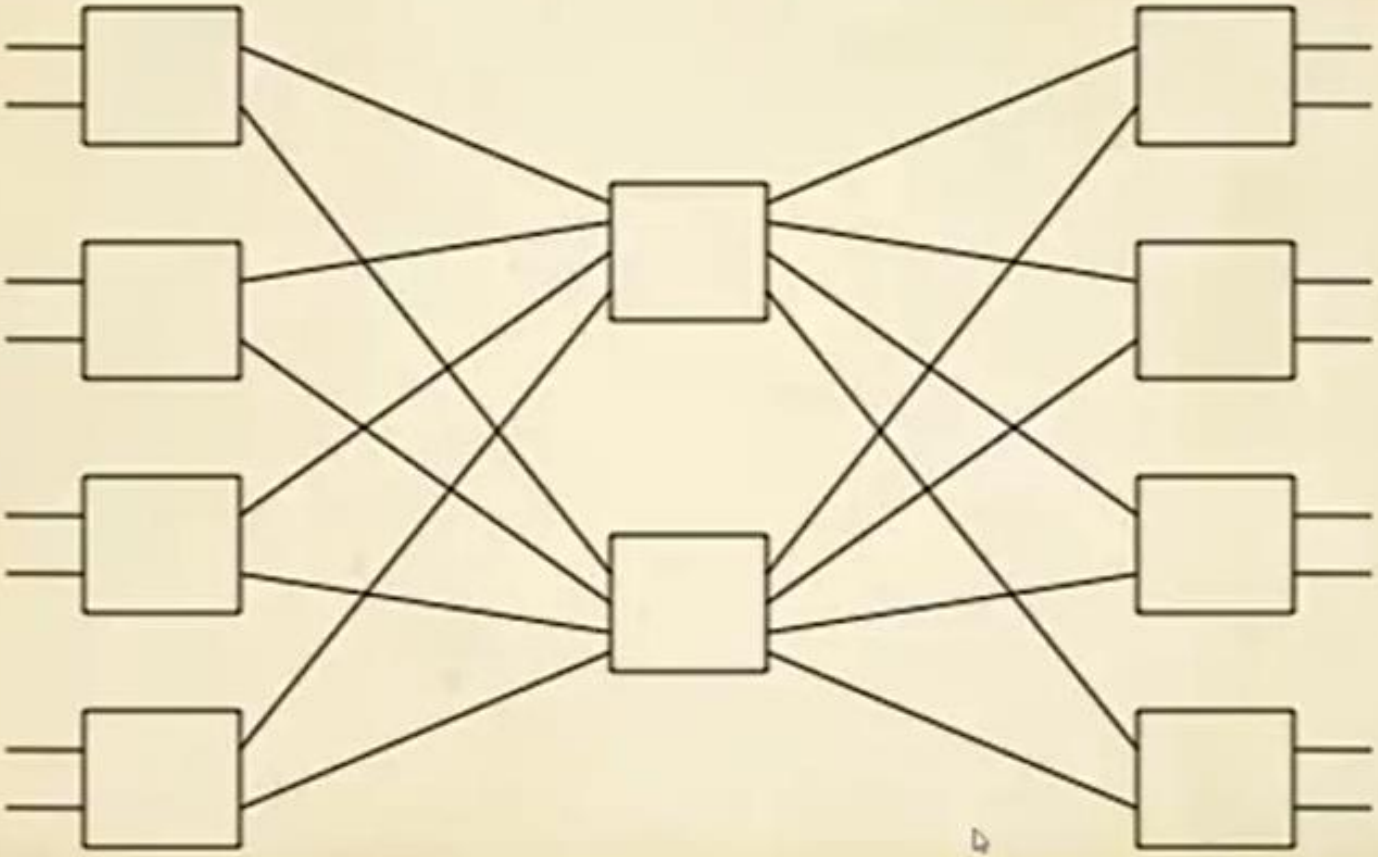
Myrinet的连接（少于256个结点）

- Clos网络
- 使用8/16口交换机
- 使用交换机架
  - 16
  - 32
  - 64
  - 128
  - 256

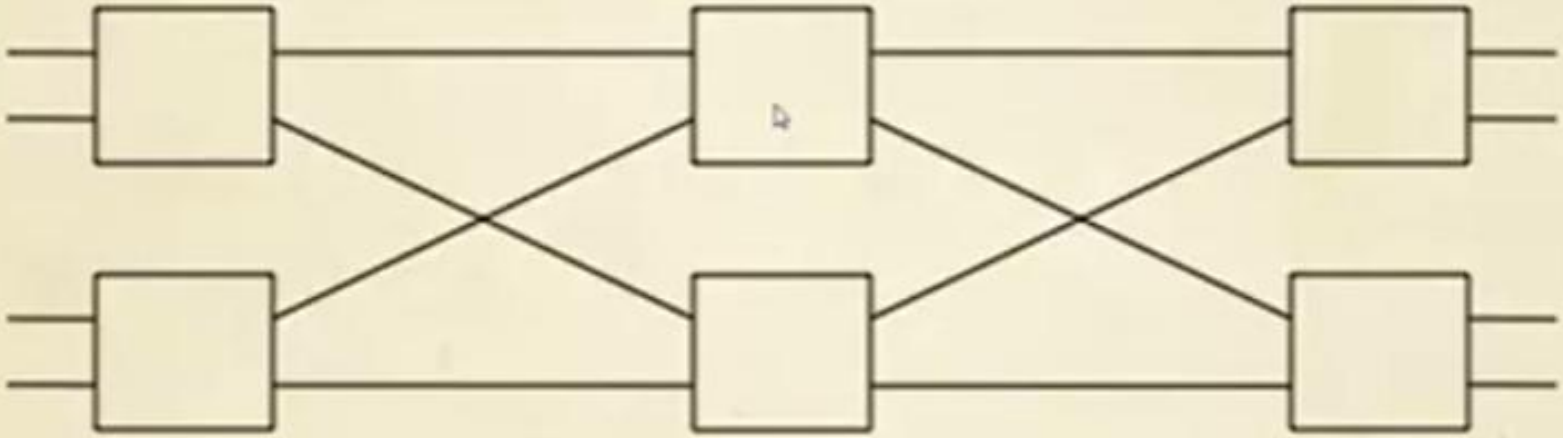
Clos network



Example: Step 1

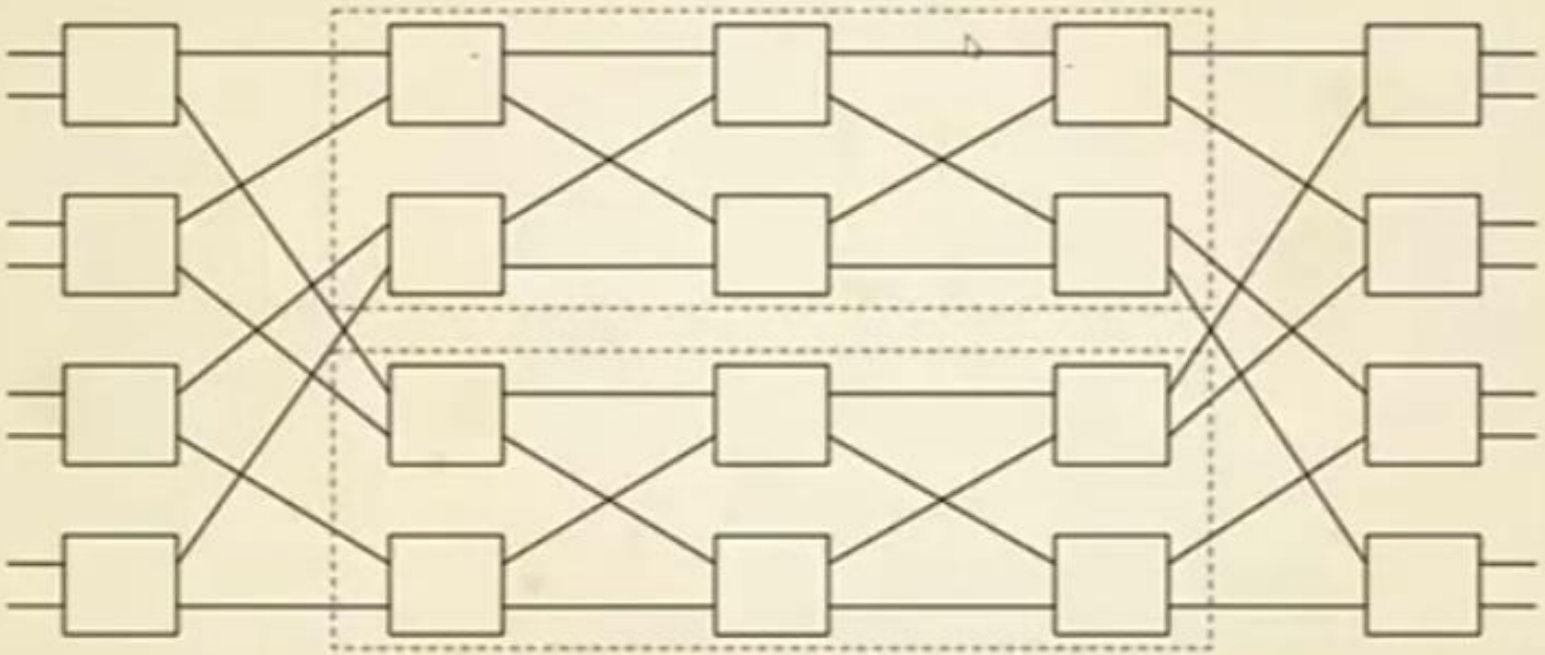


Step 2

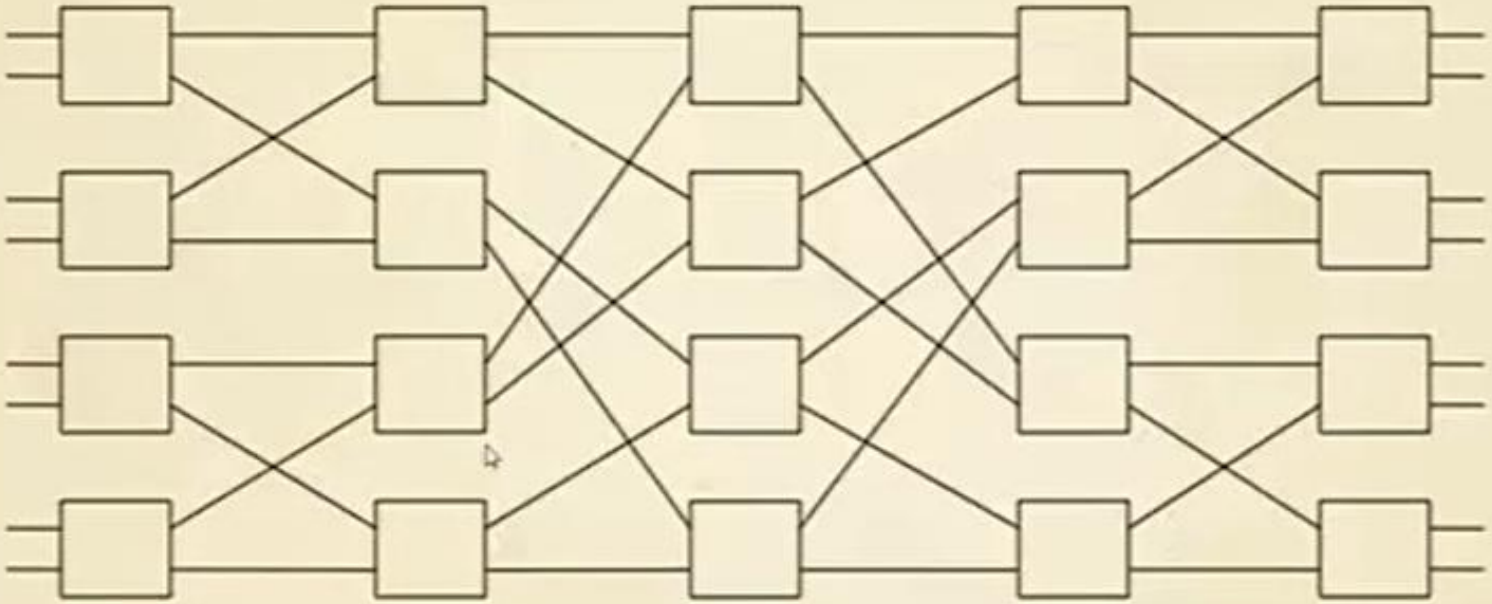


16:43

Step 3

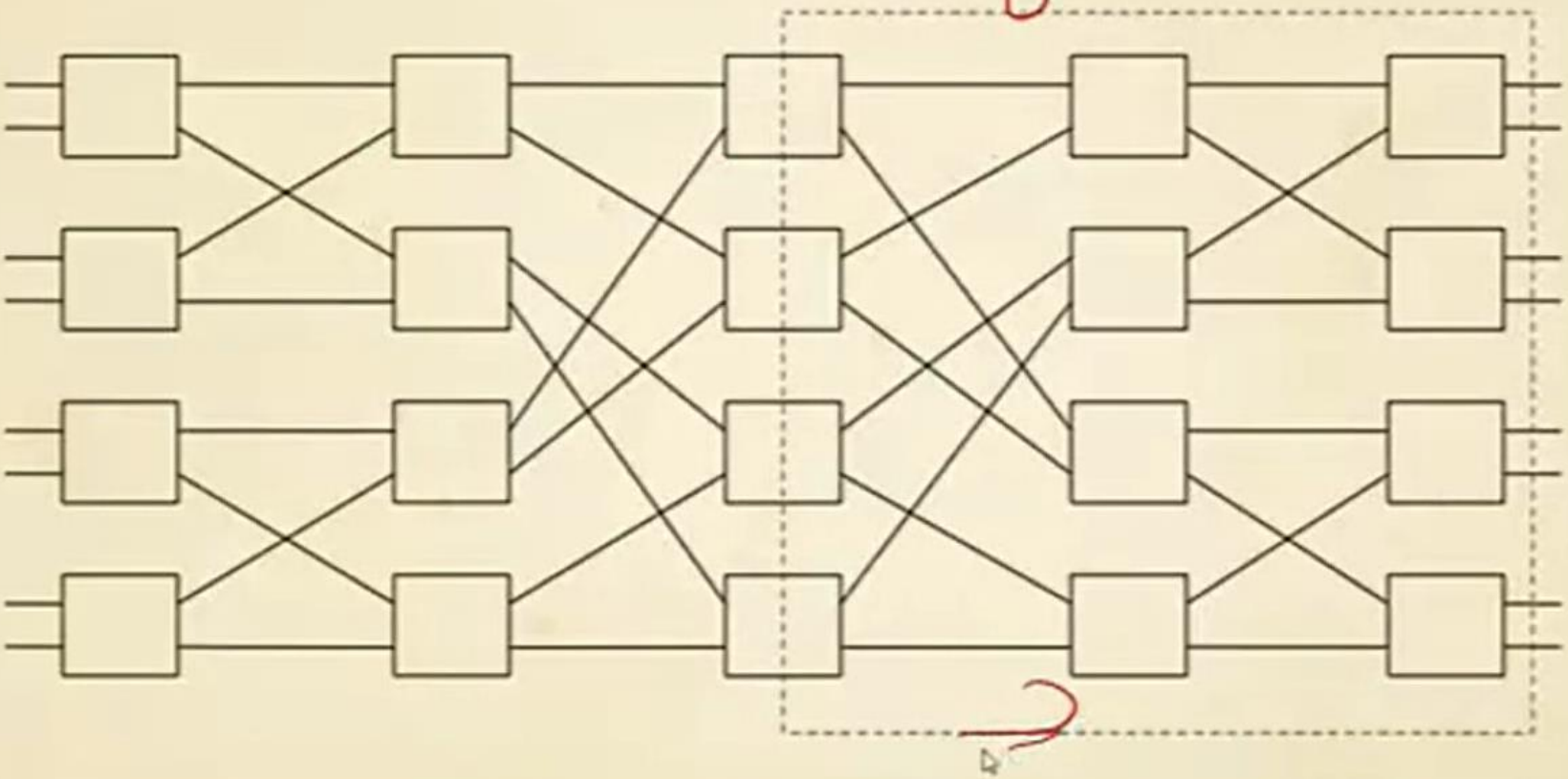


Step 4

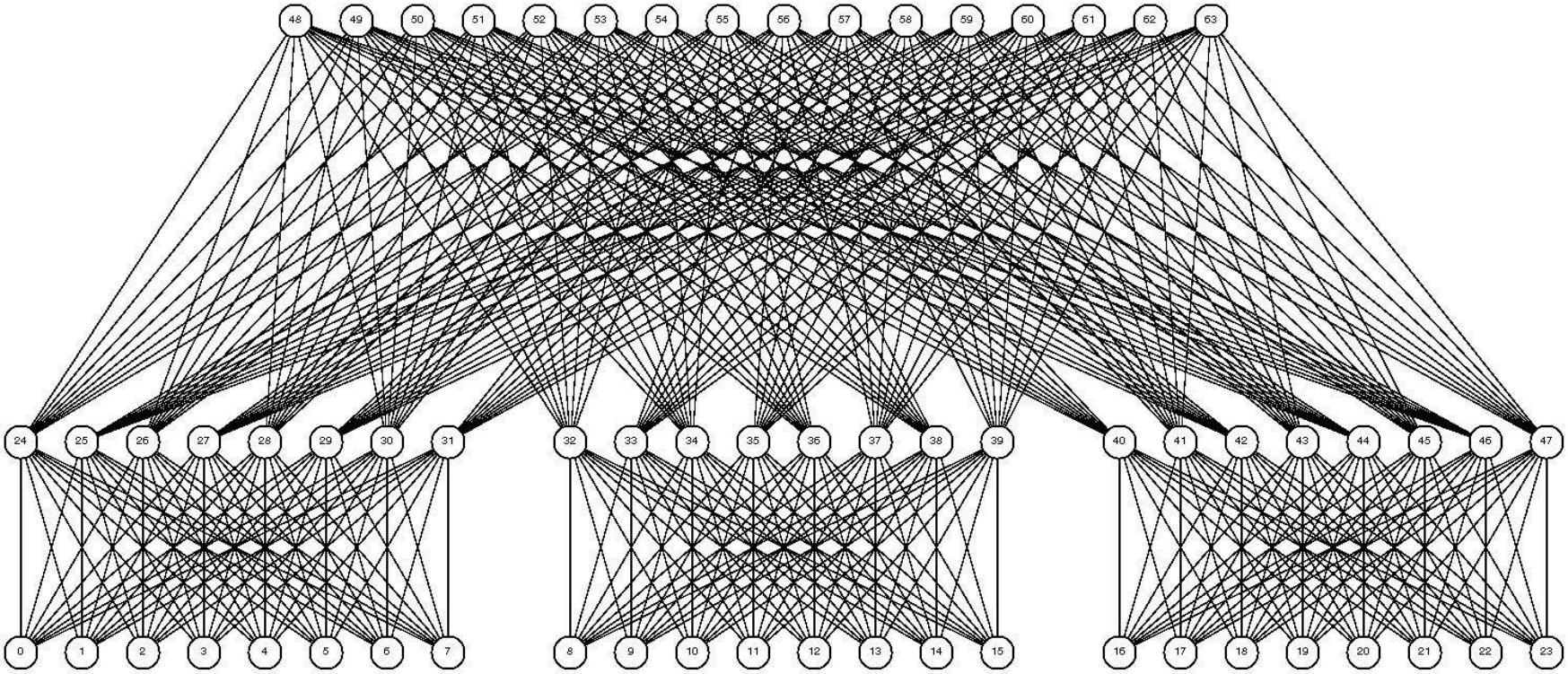


Step 5

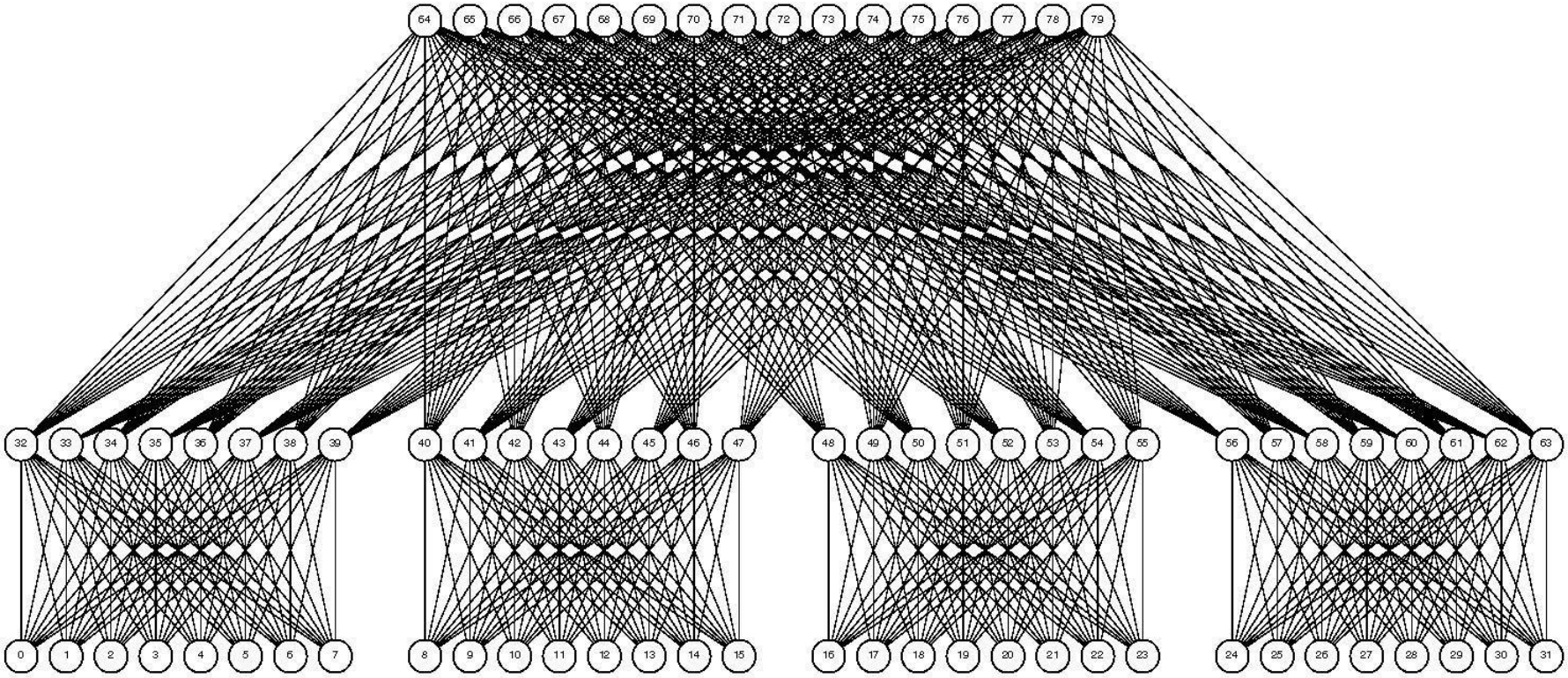
Folding



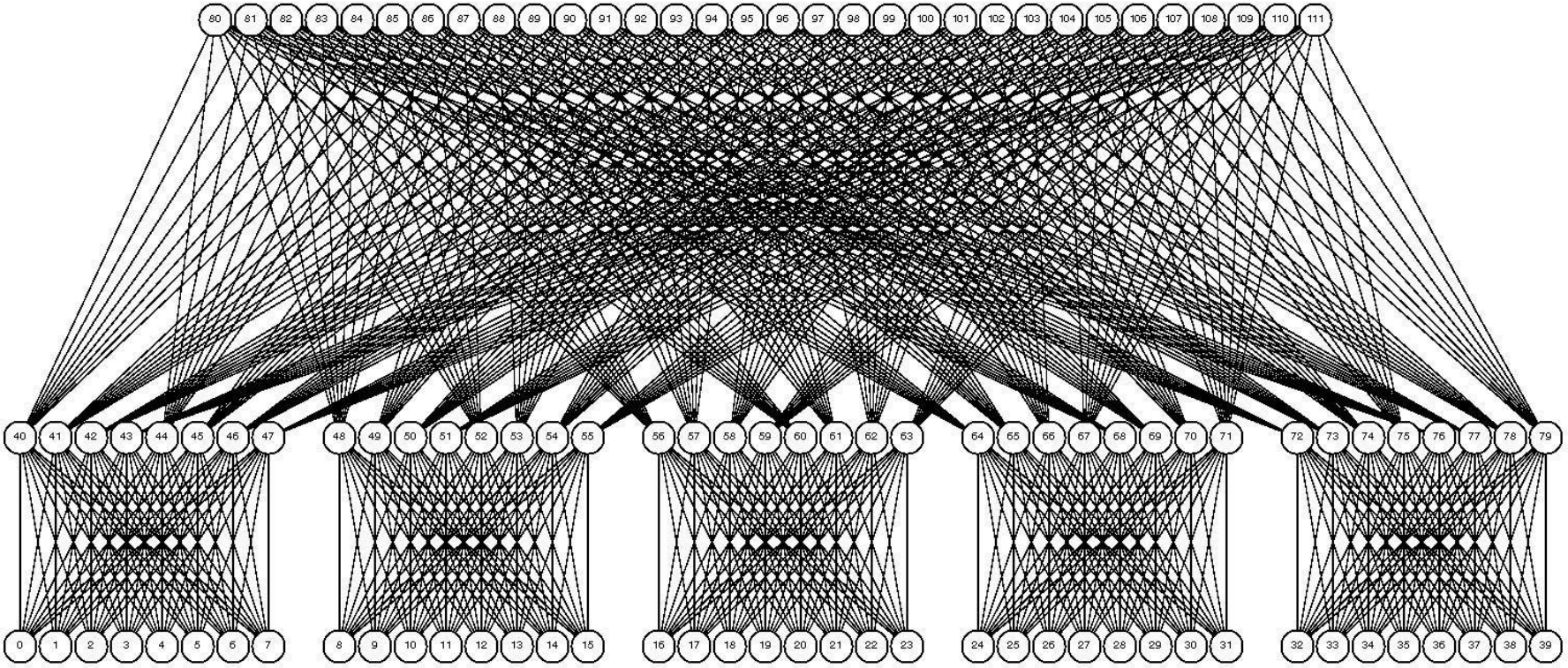
192 Hosts, 64 SW16 Switches



256 Hosts, 80 SW16 Switches



320 Hosts, 112 SW16 Switches



练习

- 用尽可能少的4端口交换机搭建8端口全互联网络的连接图。