Today

- ipv6 api
- atm
- via

ATM

- ATM physical layer
- IP over ATM
- AAL programming

ATM

asynchronous transfer mode

- developed by phone guys
- both WAN and LAN support
- provides QoS
- good for voice, video, data
- IP over ATM
- or direct AAL programming

ATM architecture

- fiber-based usually (OC3, OC12, OC48)
- connection oriented, cell switching
- based on 53-byte cells – 5 byte header, 48 byte payload
- header: VPI,VCI,payload type, CLP, HEC
  - VPI is routing field through ATM switches
  - VCI is end user designator
  - CLP – cell loss priority
  - HEC – header error control
- cells alway in order, but can be dropped
- services
  - real time – CBR, VBR
  - non-real time – ABR, UBR

VPI/VCI

- VC can provide QoS – cell loss ratio, cell delay variation, max cell rate, minimum cell rate
- traffic shaping/policing to enforce QoS
- cell sequence preserved
- switch can discard
- VC setup is handled by "control plane"
- setup proceeds through each switch to destination
- VPI can be re-routed transparently, all its VCI go with it
- appearance of "direct links" between your sites

LAN and IP

- ATM interface cards for your computer
- no broadcast
- can do multicast– point-to-multipoint
- LAN emulation server resolves ARP's to ATM address
- IP over ATM defined (MTU 9180)

transparent to your TCP/IP apps

AAL programming

ATM Adaptation Layer

- AAL services
  - handle transmission errors
  - segmentation and reassembly
  - flow control
- AAL0 – single cells
- AAL5 – 2**16 payload, CRC, used for IP
- can use native ATM; avoid IP overhead, specify QoS requirements
- early API, atm_open, atm_read, atm_write
- native AAL can coexist with IP-over-ATM
- need to set up VPI through switches (net mgr)

ATM socket

- address: vpi.vci
- setsockopt for QoS

ATM sockaddr struct

```
struct sockaddr_atmsvc {
    unsigned short      sas_family;     /* address family, AF_ATMSVC
    struct {                            /* SVC address */
        unsigned char   prv[ATM_ESA_LEN];/* private ATM address */
        char            pub[ATM_E164_LEN+1]; /* public address (E.16
                                        /* unused addresses must be
        char            lij_type;       /* role in LIJ call; one of
        uint32_t        lij_id;         /* LIJ call identifier */
    }                   sas_addr;       /* SVC address */
};
```

atm reader

Linux ATM api

```
...
    if ((s = socket(PF_ATMPVC,SOCK_DGRAM,0)) < 0)
        err_sys("socket");
    memset(&addr,0,sizeof(addr));
    if (text2atm(argv[1],(struct sockaddr *) &addr,sizeof(addr),
      T2A_PVC | T2A_UNSPEC | T2A_WILDCARD) < 0) usage(name);
    memset(&qos,0,sizeof(qos));
    qos.aal = ATM_AAL5;
    qos.rxtp.traffic_class = ATM_UBR;
    qos.rxtp.max_sdu = BSIZE;
    if (setsockopt(s,SOL_ATM,SO_ATMQOS,&qos,sizeof(qos)) < 0)
        err_sys("setsockopt SO_ATMQOS");
    if (bind(s,(struct sockaddr *) &addr,sizeof(addr)) < 0)
        err_sys("bind");
...
   n = read(s,buff,BSIZE);
```


ATM writer

```
...
    if ((s = socket(PF_ATMPVC,SOCK_DGRAM,0)) < 0)
        err_sys("socket");
    memset(&addr,0,sizeof(addr));
    if (text2atm(argv[1],(struct sockaddr *) &addr,sizeof(addr),T2A_
        usage(argv[0]);
    memset(&qos,0,sizeof(qos));
    qos.aal = ATM_AAL5;
    qos.txtp.traffic_class = ATM_UBR;
    qos.txtp.max_sdu = strlen(argv[2])-offset;
    if (setsockopt(s,SOL_ATM,SO_ATMQOS,&qos,sizeof(qos)) < 0)
        err_sys("setsockopt SO_ATMQOS");
    if (connect(s,(struct sockaddr *) &addr,sizeof(addr)) < 0)
        err_sys("connect");
    size = write(s,buff,lth);
...
```


low latency

- reduce latency for RPC, etc. (LANs)
- higher speed net (gig-e)
- UDP instead of TCP
- new protocols (XTP, T/TCP)
- kernel mods (myrinet, Duke's gigabit/sec TCP)
- CRAY SHMEM (put's get's)
- AAL
- avoid memory copies and context switches
- active messages (U-Net)
- VIA (virtual interface architecture)

VIA

- low-latency message passing
- zero copy: interface card to/from user buffer
- eliminate context switches
- used by MPI/PVM
- hardware implentation or software emulation
- some interface cards VIA ready (giganet)
- kernel hacks for some net cards
- support by intel, compaq, microsoft and others

VIA features

- protected user-level access to net interface
- send/receive
- RDMA write (remote direct memory access)
- data transfer without OS
- to/from anywhere in address space
- scatter/gather
- polling/blocking for completion
- message can be dropped if recv not posted
- options for blocking send til recv'd

memory mgt

- data buffers can reside anywhere in user space
- register buffer with VI
  - kernel locks pages
  - kernel provides virtual-physical translation for net card
  - page tables provide address translations and enforce access right

VIA standards

- API (VIPL) only lossely specified
- kernel-hardware spec is loose implementations on same box from different vendors probably won't work
- end-point naming is unspecified

Performance: latency: 20 us (200 for ether/UDP)

next time

XTI

performance

futures