## Today

- long fat pipes
- assignments 4/5/6
- review



## long fat pipes

bandwidth-delay product

- bandwidth and delay of path dictates buff size

```
T1 (1.5 Mbs) and RTT 60 ms:  need 11.5 KBytes buffer or pipe
  will not stay full
gigabit/sec and RTT 60 ms: need 7.5 MBytes
```

- faster links or slower RTT (satellite) requires bigger buffs
- may exceed max window size (65535)



## new TCP options

RFC 1323 (both sides must comply)

- needed for long fat pipes
- window scale option (1 GByte)
- that much data in flight, can't afford to resend: need selective ACK (SACK)
- RTT calculated only once every roundtrip, with time-stamp option, can do on every send-ack
- sequence numbers may wrap on high speed link

```
packets should be discarded after 2MSL
on Ethernet, takes 60 minutes to send 4GB and wrap sequence number
on FDDI, takes 5 minutes
giganet, 34 seconds

PAWS -- protection against wrapped sequence numbers
```

- use time-stamp option to discard wrapped sequence numbers



## UDP vs TCP

- must use UDP for multicast/broadcast (would need to send N copies with TCP)
- UDP for simple request-reply apps
  - TCP connection set up expensive (packets, RTT)
  - may not need flow control
  - might still need timeout/retransmission/sequencing
- active flows to lots (100s) of hosts from a server may exceed file descriptors for OS (TCP), then use UDP
- bulk transfer, use TCP
- can emulate datagram with readn() in TCP
- multimedia app's use UDP

## midterm review

where we've been...

- network architecture (layers)
- BSD sockets, DNS
- TCP clients and servers
- UDP clients and servers
- concurrency (fork/signal)
- BSD servers (inetd)
- IP protocols
- tools



## Objectives

- understand BSD socket calls
- understand IP protocols
- examine client/servers
  *"how do they do that"*
- develop client/servers
  design
  implement
  diagnose
- understand design tradeoffs
- develop portable library



## Concepts

- **client/servers**
  - concurrent/iterative
  - statefull/stateless
- **concurrency**
  - multiple processes/threads
  - asynchronous events (signal)
  - nonblocking I/O
- **communications**
  - stream/datagram (circuits/packets)
  - reliable/unreliable
  - performance



## philosophy

*I hear, and I forget.*
*I see, and I remember.*
*I do, and I understand.*

```
        Assignments
1  telnet and ping
2  Ethernet and traceroute
3  TCP client,
4  UDP server, TCP client/server
5  concurrent i/o (select)
6  concurrent server (fork)
```

## BSD sockets

- transport layer interface
- API, subroutine library
- no standards (BSD is de facto)
- supports multiple protocol families (TCP/IP, XNS, UNIX, IPv6, ATM)
- flexibility is paid for in complexity
- mixture of filling data structures and function calls
- supports I/O abstraction
- **full duplex**, can't read what you write
- readn/writen for streams

```
socket bind connect listen accept read write sendto recvfrom
setsockopt getsockname gethostbyname gethostbyaddr inet_ntoa
getservbyname htons

signal fork wait select alarm
```



## IP architecture

- layers and encapsulation
- IP: addressing, routing, fragmentation, TTL
- ICMP: echo and error feedback
- UDP: ports, checksums, unreliable
- TCP: timers, sequence numbers, buffers, retransmission, ACKs, flow control, congestion control, 3-way handshake, close
- performance: big buffers, concurrency



## tools

- perror()
- trace/truss
- netstat -r -a -s -i
- lsof
- ifconfig
- ping, telnet
- traceroute
- nslookup, arp
- ttcp, netperf, sock, nnstat
- tcpdump



## next time

inclass midterm

what's ahead:

- BSD servers
- multicast
- RPC/XDR
- BPF
- IP security
- IPv6
- winsock, java, perl