BBT

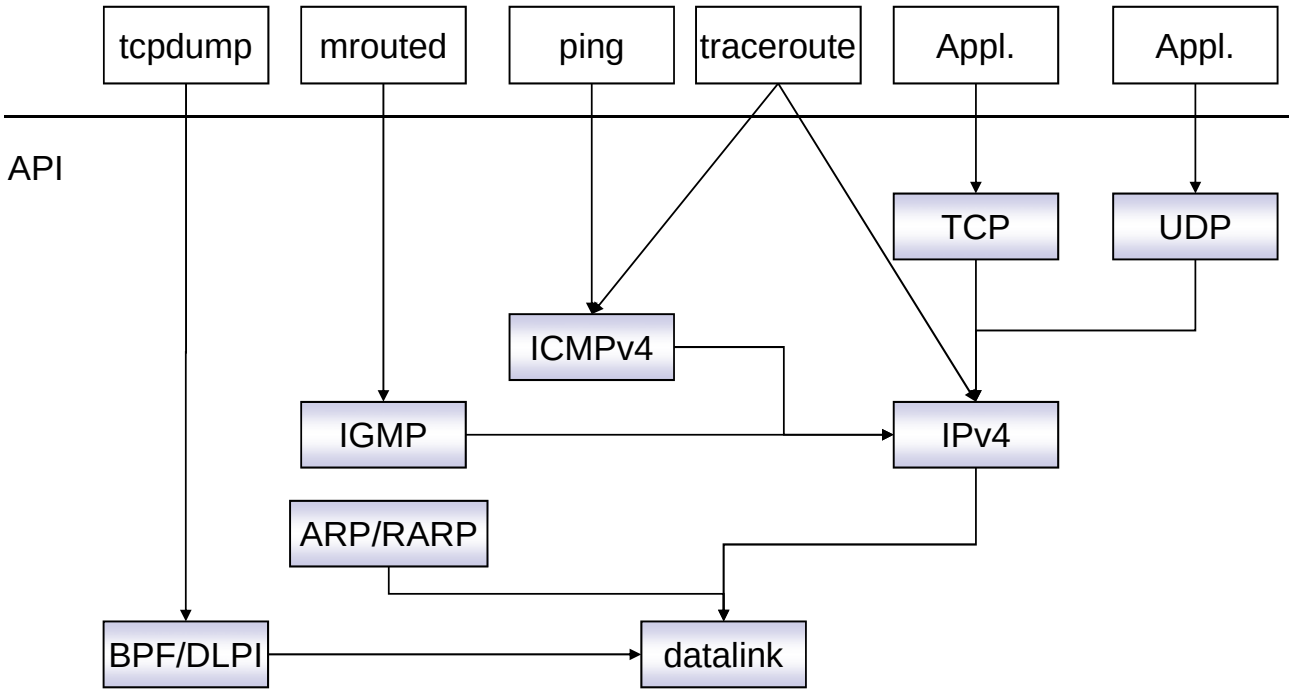
CPP

TCP/IP protocols

Socket

BBT

Overview of TCP/IP protocol



Protocols

- IPv4
  - Internet Protocol, version4. It uses 32-bit addresses. IPv4 provides the packet delivery service for TCP, UDP, ICMP, and IGMP.
- TCP
  - transmission Control Protocol. TCP is a connection-oriented protocol that provides a reliable, full-duplex, byte stream for the user process. TCP socket are an example of stream sockets. TCP takes care of details such as acknowledgements, timeouts, retransmissions, and the like. Most internet application programs use TCP.

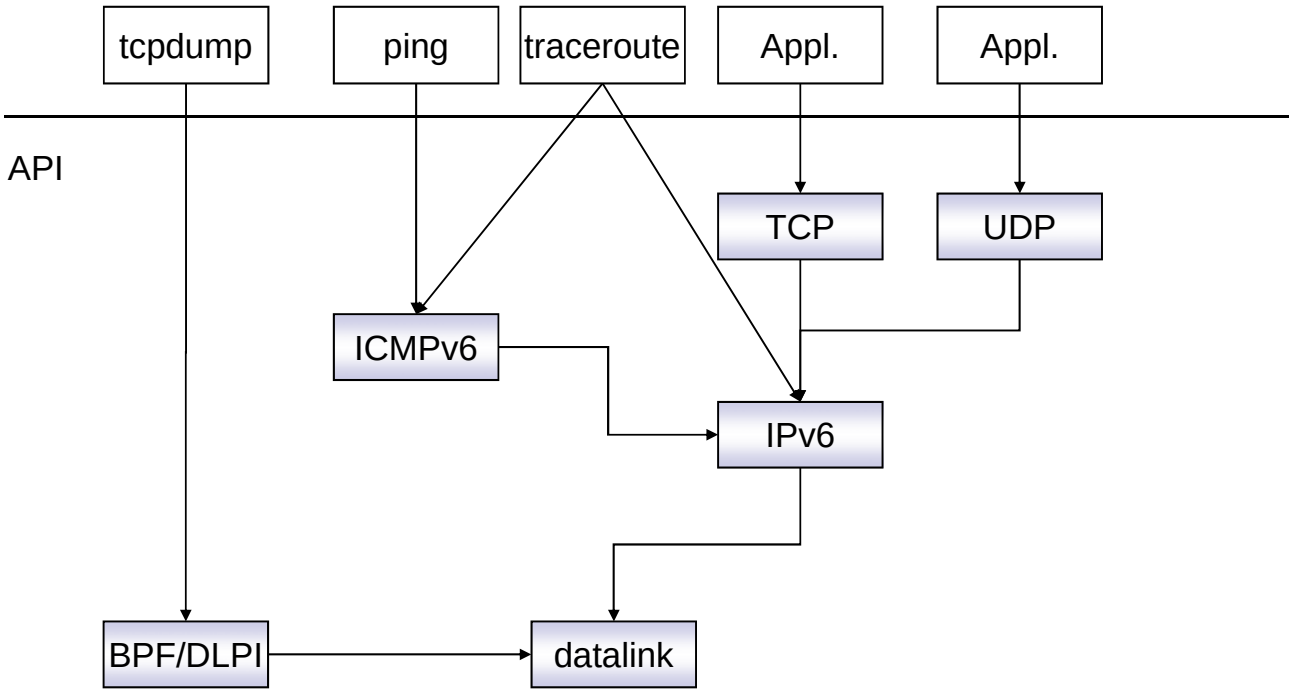
Protocols –cont.

- UDP
  - User Datagram Protocol. UDP is a connectionless protocol and UDP sockets are an example of datagram sockets. there is no guarantee that UDP datagrams ever reach their intended destination.
- ICMP
  - internet Control Message Protocol. ICMP handles error and control information between routers and hosts. These messages are normal generated by and processed by the TCP/IP networking software itself, not user processes.

Protocols –cont.

- IGMP
  - internet group Management Protocol. IGMP is used with multicasting, which is optional with IP.
- ARP
  - Address Resolution Protocol, ARP maps an IPv4 address into a hardware address (such as Ethernet address).
- RARP
  - reverse Address Resolution Protocol. RARP maps a hardware address into an IPv4 address.

IPV6 replacements



protocols

- IPv6
  - Internet Protocol, version 6. IPv6 was designed in the mid-1990s as a replacement for IPv4. The major change is a larger address comprising 128bits, to deal with the explosive growth of the Internet in the 1990s. IPv6 provides the packet delivery service for TCP, UDP, and ICMPv6
- ICMPv6
  - Internet Control Message Protocol, version 6. ICMPv6 combines the functionality of ICMPv4, IGMP, and ARP.

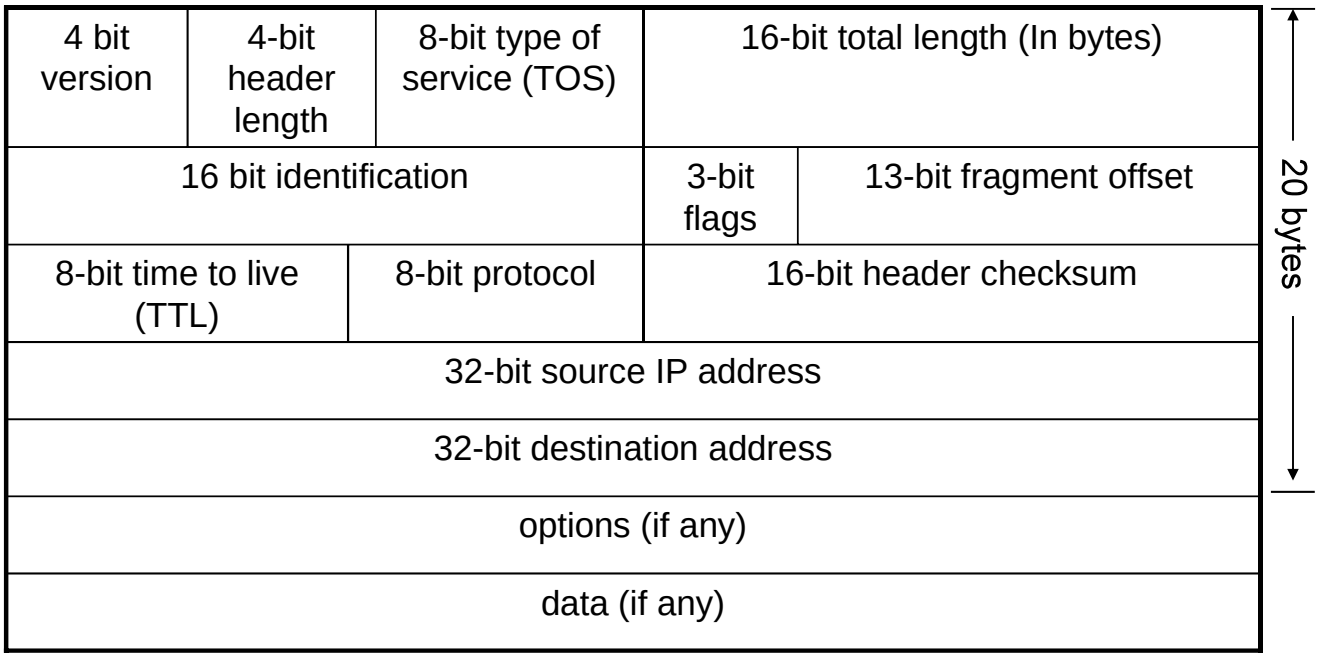
BPF & DLPI

- BPF
  - BSD Packet Filter which is the method of collecting data from this network interface running into promiscuous mode. BPF receives copies from the driver of sent packets and received packets. Before traveling through the kernel all the way up to the user process the user can set a filter so only interesting packets go through the Kernel.
- DLPI
  - data link provider interface. This interface provides access to the datalink and is normally provided with SVR4

IP

- IP provides an unreliable, connectionless datagram delivery service.
  - unreliable - there are no guarantees that an IP datagram successfully gets to its destination.
  - connectionless - IP does not maintain any state information about successive datagrams.

IP header



time-to-live (TTL)

- sets an upper limit on the number of routers through which a datagram can pass. It limits the lifetime of the datagram. It is initialized by the sender to some value (often 32 or 64) and decremented by one by every router that handles the datagram. When this field reaches 0, the datagram is thrown away, and the sender is notified with an ICMP message. (IP_TTL)

Fragmentation

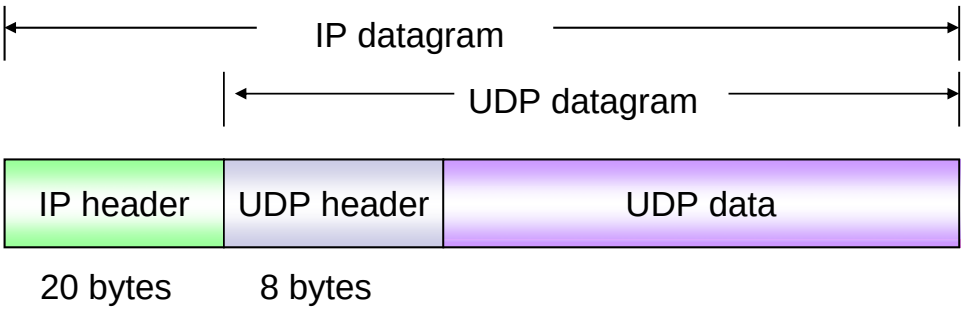
- IP layer receives an IP datagram to send, and queries the interface to obtain its MTU. IP compares the MTU with the datagram size. If the datagram size is greater than MTU size, it performs fragmentation . Fragmentation can take place either at the original sending host or at an intermediate router.
  - **identification** - contains a unique value for each IP datagram that the sender transmits. This number is copied into each fragment of a particular datagram.
  - **flags** - uses one bit as the "more fragments" bit. This bit is turned on for each fragment comprising a datagram except the final fragment.
  - **offset -** contains the offset of this fragment from the beginning of the original datagram

UDP

- UDP is a simple, datagram-oriented, transport layer protocol: each output operation by a process produces exactly one UDP datagram, which causes one IP datagram to be sent.

BBT

UDP data in an IP datagram



UDP Header

16 bit source port number	16-bit destination port number
16-bit UDP length	16-bit UDP checksum
data (if any)	

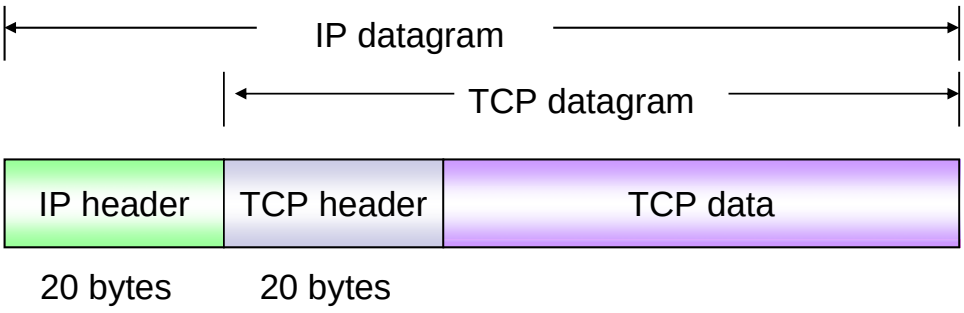
TCP protocol

- TCP provides connections between clients and servers.
  - A TCP client establishes a connection with a given server, exchanges data with that server across the connection, and then terminates the connection.
- TCP provides reliability.
  - When TCP sends data to the other end, it requires an acknowledgement in return.
  - Maintain a checksum on its header and data. If a segment arrives with a invalid checksum, TCP discards it.
  - TCP sequences the data by associating a sequence number with very byte that it sends.

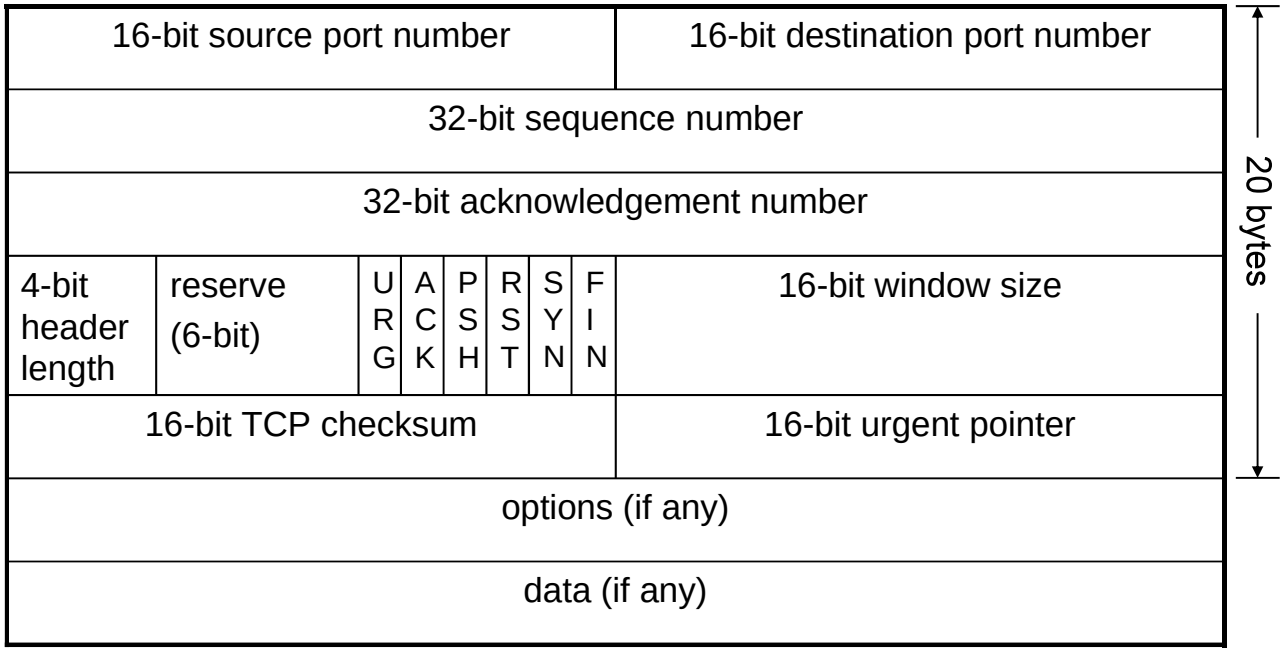
TCP protocol –cont.

- TCP provide flow control. TCP always tells its peer exactly how many bytes of data it is willing to accept from the peer. This is called the advertised window.
  - This window is the amount of room currently available in the receive buffer, guaranteeing that the sender cannot overflow the receiver's buffer.
- TCP connection is also full-duplex.

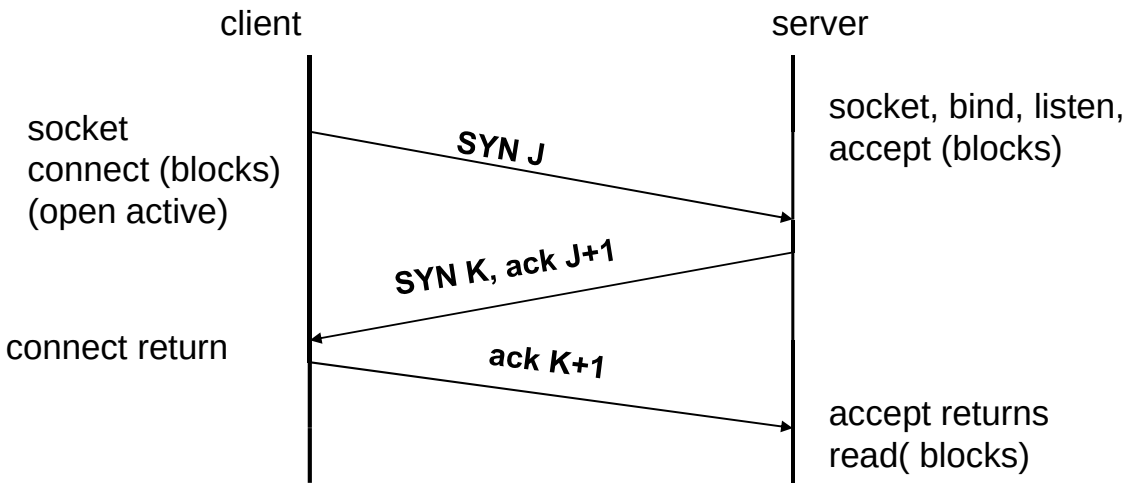
TCP data in an IP datagram



TCP header



TCP connection establishment



- Three way handshake
- SYN is a SYN segment (synchronization), it just contain IP header, TCP header and TCP options.

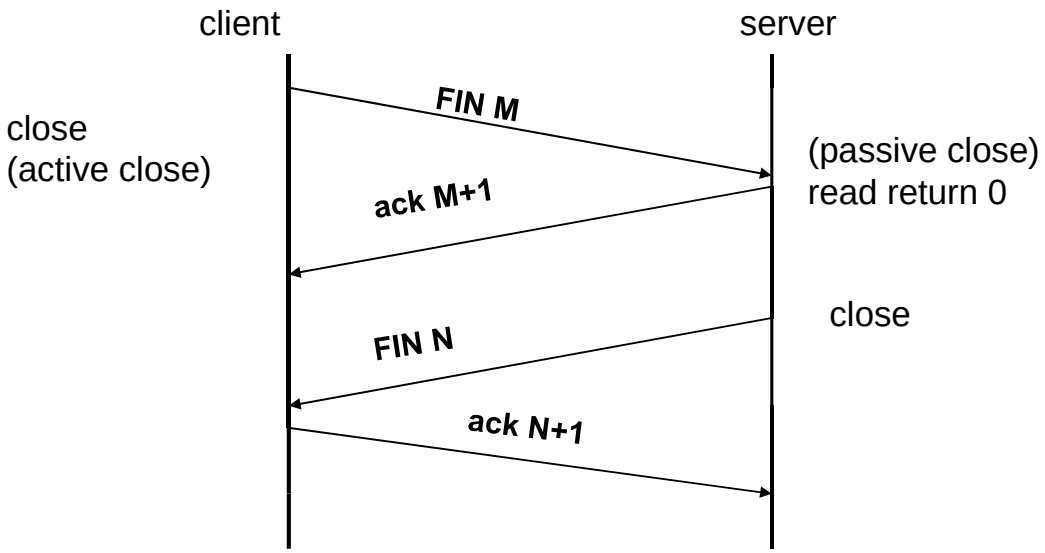
SYN options

SYN can contain TCP options

- MSS
  - with this option the TCP sending the SYN announces its maximum segment size, the maximum amount of data that it is willing to accept in each TCP segment. (TCP_MAXSEG)
- Window scale option (advertised windows size)
  - The maximum window that either TCP can advertise to the other TCP is 65535, because the corresponding field in the TCP header occupies 16 bit. This option specifies that the window in the TCP header must be scaled (left-shifted) by 0-14 bit, providing maximum window size (65536* 2^{14}). (SO_RCVBUF)

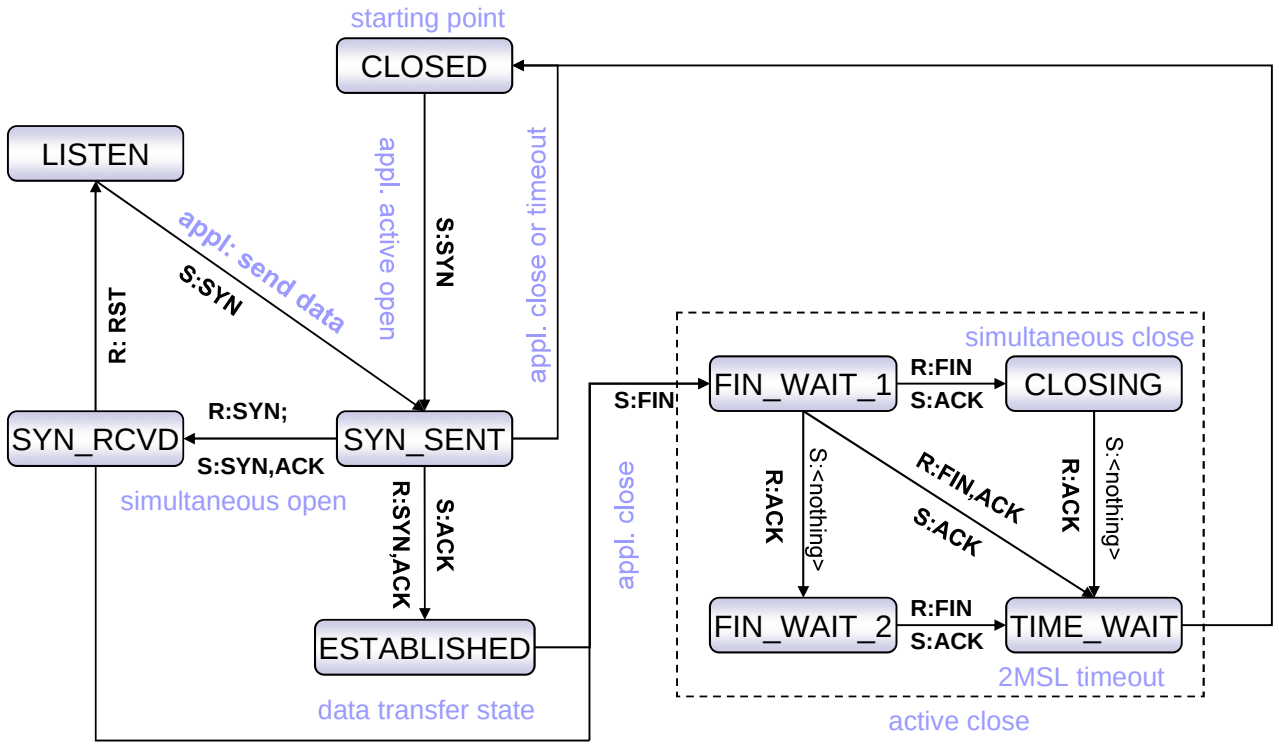
BBT

TCP connection close



TCP state transaction

R: receive S: send
appl. application

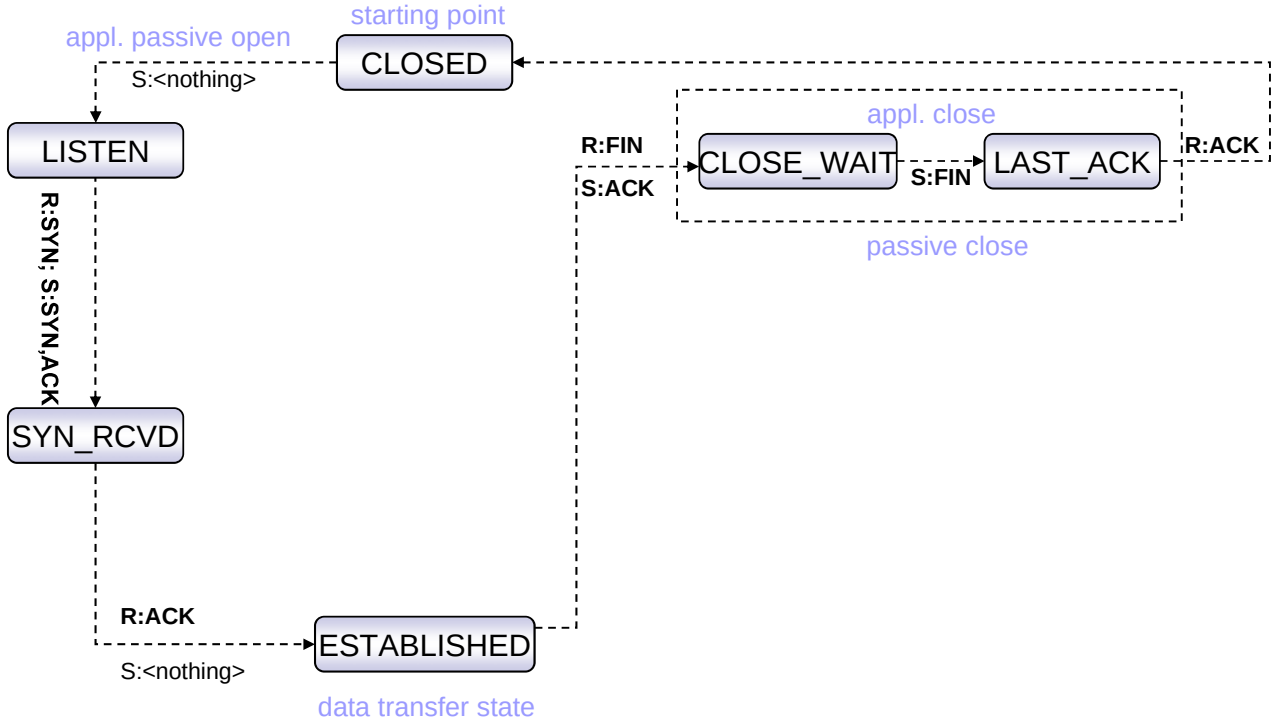


Normal transitions for client Normal transitions for server

BBT

TCP state transaction

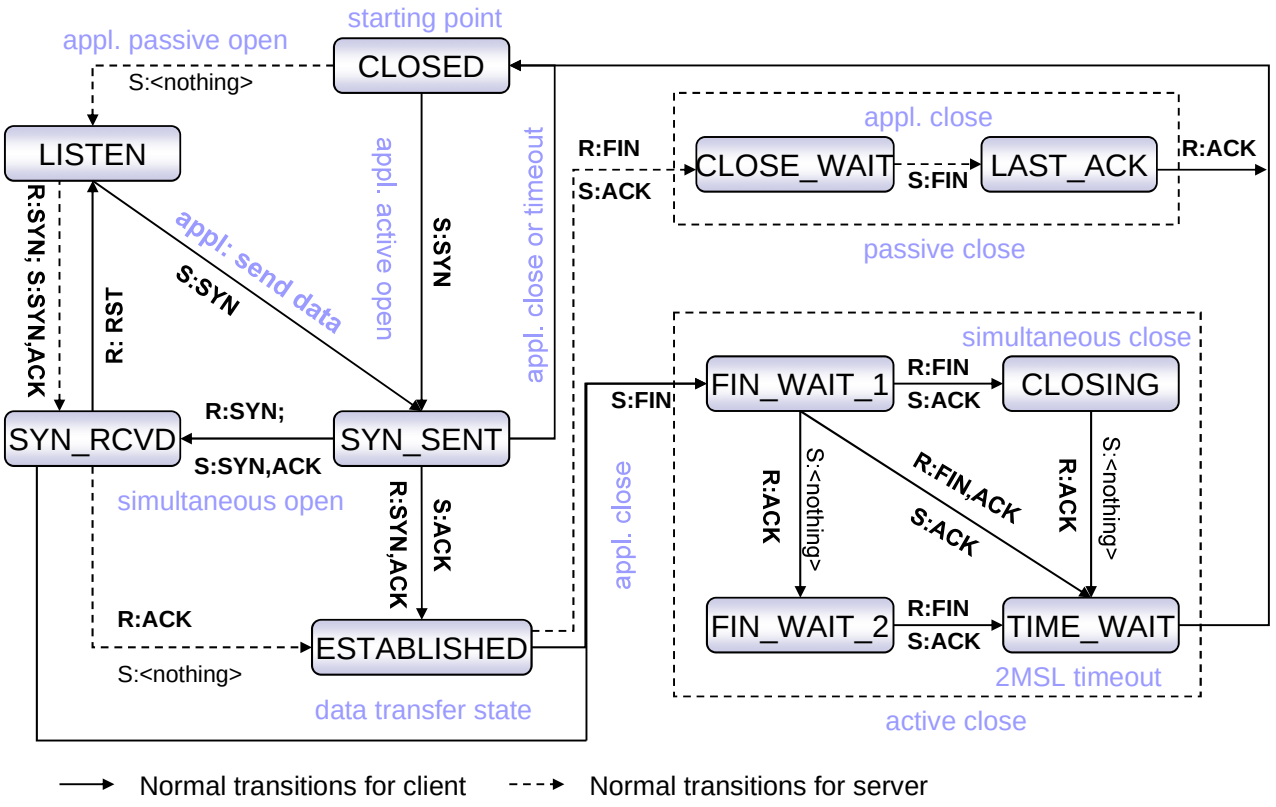
R: receive S: send
appl. application



Normal transitions for client Normal transitions for server

TCP state transaction

R: receive S: send
appl. application



BBT

TIME_WAIT state

- The end that performs the active close goes through TIME_WAIT state.
- The duration that this endpoint remains in this state is twice the MSL (maximum segment lifetime), or 2MSL.
- There are two reasons fot the TIME_WAIT state
  - Implement TCP's full-duplex connection termination reliably
  - Allow old duplicate segments to expire in the network

Buffer size and limitations

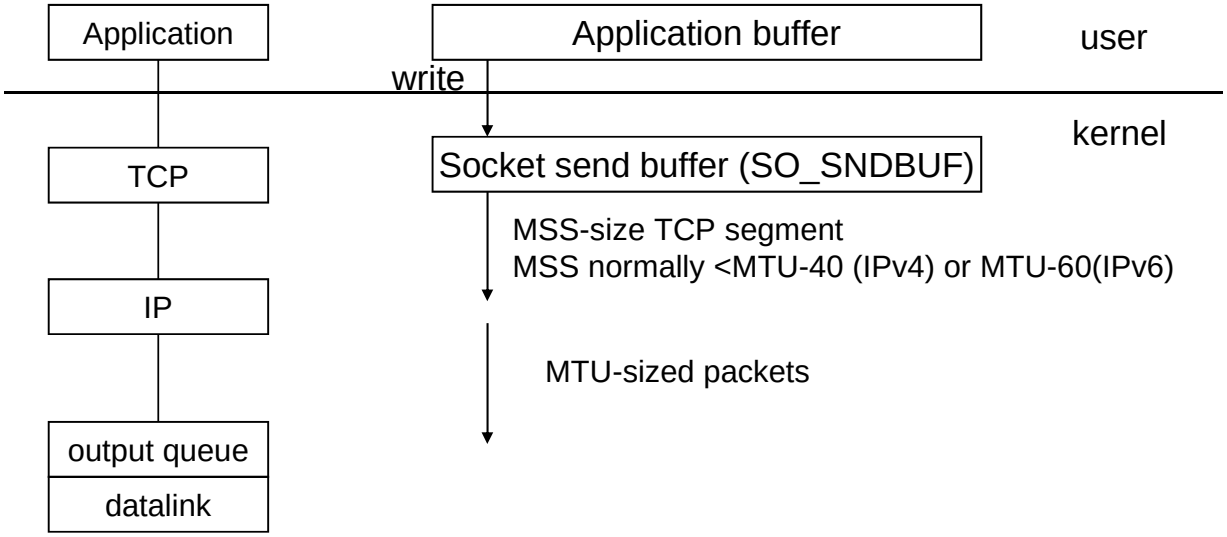
- The maximum size of an IPv4 datagram is 65535 bytes, include IPv4 header.
- The maximum size of an IPv6 datagram is 65575 bytes, including the 40-byte IPv6 header.
- MTU (maximum transmission unit) Ethernet MTU is 1500. the minimum MTU for IPv4 is 68 bytes, for IPv6 is 576 bytes.
- The smallest MTU in the path between two hosts is called the path MTU.
- When an IP datagram is to be send out an interface, if the size of the datagram exceeds the link MTU, fragmentation is performed. The fragments are never reassembled until they reach the final destination.

Limitations –cont.

- IPv4 and IPv6 define a minimum reassembly buffer size: the minimum datagram size that we are guaranteed any implementation must support. For IPv4 this is 576 bytes. 1500 bytes for IPv6.
- TCP has an MSS, maximum segment size, that announces to the peer TCP the maximum amount of TCP data that the peer can send per segment. The goal of the MSS is to tell the peer the actual value of the reassembly buffer size and try to avoid fragmentation.

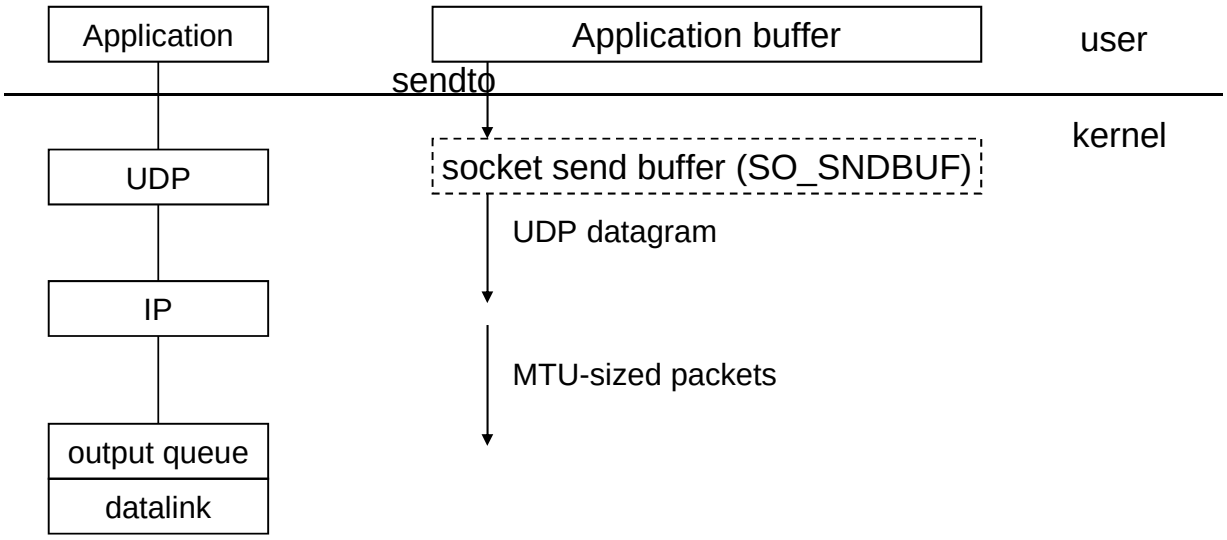
BBT

TCP output



BBT

UDP output



TCPDUMP

- collect data from this network interface running into promiscuous mode. Using BPF or DLPI.

tcpdump format

<tcp-segment> starting sequence number: ending sequence number (data bytes)

- Example:
  - 1412042008:1412042008(0) is the sequence number of the packet and the number of data sent.

TCP segment

- TCP segment organized like this :
  - timestamp source -> destination : flags
- Flags can be any of the list
  - S -> SYN (Synchronize sequence numbers - Connection establishment)
  - F -> FIN (Ending of sending by sender - Connection termination)
  - R -> RST (Reset connection)
  - P -> PSH (Push data)
  - . (No flag is set)
  - We can also find ACK (Acknowledgement) and URG (Urgent) flags following the ones above.

Example:

1.	0.0	svr4.1037 > bsdi.discard: S 1415531521:1415531521(0) win 4096 <mss 1024>
2.	0.002402 (0.0024)	bsdi.discard > svr4.1037: S 1823083521:1823083521(0) ack 1415531522 win 4096 <mss 1024>
3.	0.007224 (0.0048)	svr4.1037 > bsdi.discard: ack 1823083522 win 4096
4.	4.155441 (4.1482)	svr4.1037 > bsdi.discard: F 1415531522:1415531522(0) ack 1823083522 win 4096
5.	4.156747 (0.0013)	bsdi.discard > svr4.1037: . ack 1415531523 win 4096
6.	4.158144 (0.0014)	bsdi.discard > svr4.1037: F 1823083522:1823083522(0) ack 1415531523 win 4096
7.	4.180662 (0.0225)	svr4.1037 > bsdi.discard: . ack 1823083523 win 4096

BBT

Socket programming

- TCP
- UDP

BBT

Socket concept

- Style communication
  - Units of data transmission
  - Can data be lost
  - Is communication entirely
  - Namespace
- Type of protocol

BBT

Communication Styles

- SOCK_STREAM
- SOCK_DGRAM
- SOCK_RAW

BBT

Namespace

- sockaddr:

```
struct sockaddr{
  short int sa_family
  char sa_data[14]
}
```

BBT

Address format

- AF_LOCAL
- AF_UNIX
- AF_FILE
- AF_INET
- AF_INET6
- AF_UNSPEC

BBT

create socket

- **int socket (int namespace,int style, int protocol)**

“PF_INET” “SOCK_STREAM”/ “SOCK_DGRAM” “0”

BBT

Set socket address

int bind(int socket, ← socket descriptor

const struct sockaddr *addr,

socklen_t length)

IP addres+port

length of socket

BBT

Get socket address

int getsockname (int socket, struct sockaddr *addr,socklen_t *length-ptr)

BBT

Local address

include <sys/un.h>
sockaddr_un {
    short int sun_family
    char sun_path[108]
}

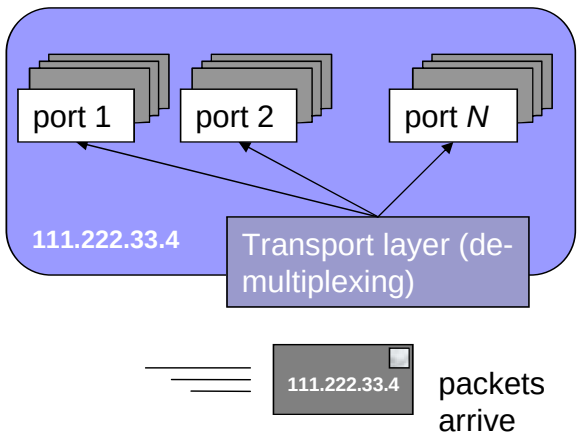
int SUN_LEN(struct sockaddr_un *ptr)

BBT

Internet address

- include <netinet/in.h>

struct sockaddr_in {
    sa_family_t sin_family;
    in_port_t   sin_port;
    struct in_addr sin_addr;
}



in_addr

- in_addr_t s_addr
  - 32bit ipv4 address, network byte ordered.
- System defined address, host byte ordered.
  - uint32_t INADDR_LOOPBACK
  - uint32_t INADDR_ANY
  - uint32_t INADDR_BROADCAST
  - uint32_t INADDR_NONE

POSIX datatypes

- include <sys/types.h>
- int8_t signed 8-bit integer
- uint8_t unsigned 8-bit integer
- int16_t signed 16 bit integer
- uint16_t unsigned 16-bit integer
- int32_t signed 32-bit integer
- uint32_t unsigned 32-bit integer

POSIX datatypes –cont.

- include <sys/socket.h>
- socklen_t length of socket address structure, uint32_t

- include <netinet/in.h>
- in_addr_t uint32_t
- in_port_t TCP or UDP port, normally uint16_t
  - network byte ordered.

Byte order conversion

- include <netinet/in.h>
- uint16_t htons(uint16_t hostshort)
- uint16_t ntohs(uint16_t netshort)
- uint32_t htonl(uint32_t hostlong)
- uint32_t ntohl(uint32_t netlong)

Host address function

- int inet_aton(const char *cp, struct in_addr *pin)
- int_addr_t inet_addr(const char *strptr) (deprecated)
- char * inte_ntoa(struct in_addr addr)
- struct in_addr inet_makeaddr(uint32_t net,uint32_t local)
  - Convert an IPv4 address between a dotted-decimal string (e.g. “192.168.212.120”) and its 32-bit network byte ordered binary value.
- int inet_pton(int af,const char *cp,void *buf)
- const char * inet_ntop(int af,const void *cp,char *buf, size_t len)
  - Handle both IPv4 and IPv6 address.

Host name

- /etc/hosts
- struct hostent
- netdb.h

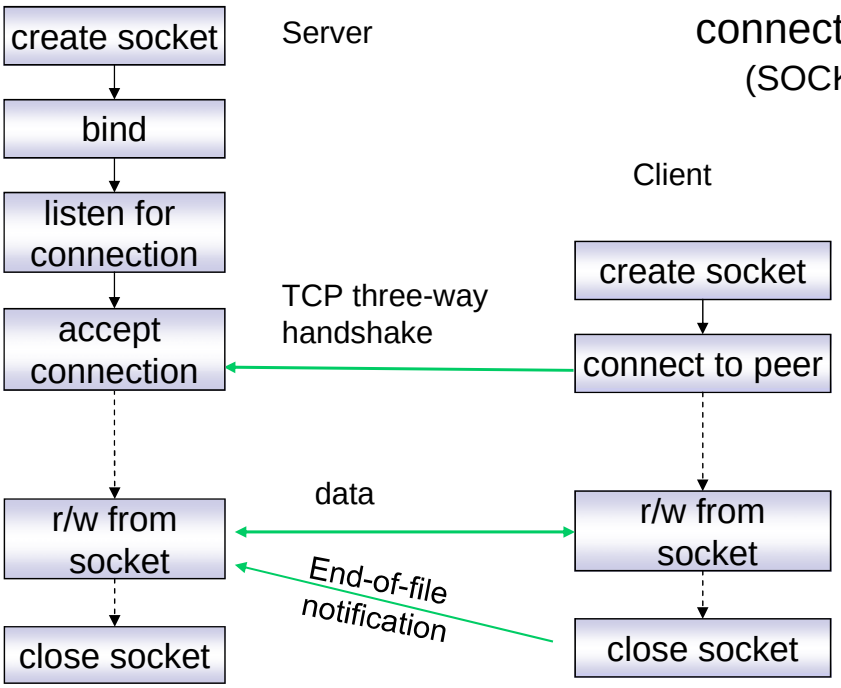
Inet port

- int IPPORT_RESERVED
- int IPPORT_USERRESERVED (netinet/in.h)
- Services database
- /etc/services
- struct servent

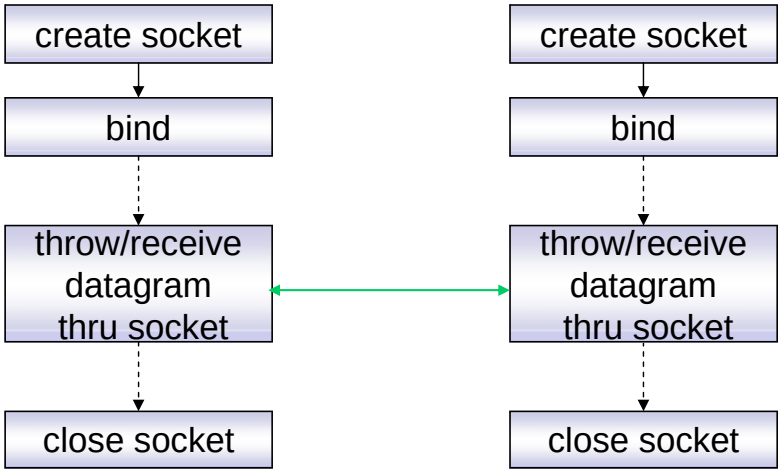
Protocols database

- /etc/protocols
- netdb.h
- struct protoent

Step in socket program



Datagram communication



connectionless
(SOCK_DGRAM)

Connect

Remote side

int connect(int socket ,struct sockaddr *addr,
Socklen_t length)

like bind!

BBT

Socket

- **waiting for connection (server)**
  int listen(int socket, unsigned int n);

Blocking wait

socket descriptor

Length of the queue

- **accepting connection (server)**
  int accept (int socket, struct sockaddr *addr,
  socklen_t *length_ptr);

return a new socket descriptor for this connection

Length of the structure, remember the *

set to NULL if you don't want it

BBT

getsockname and getpeername

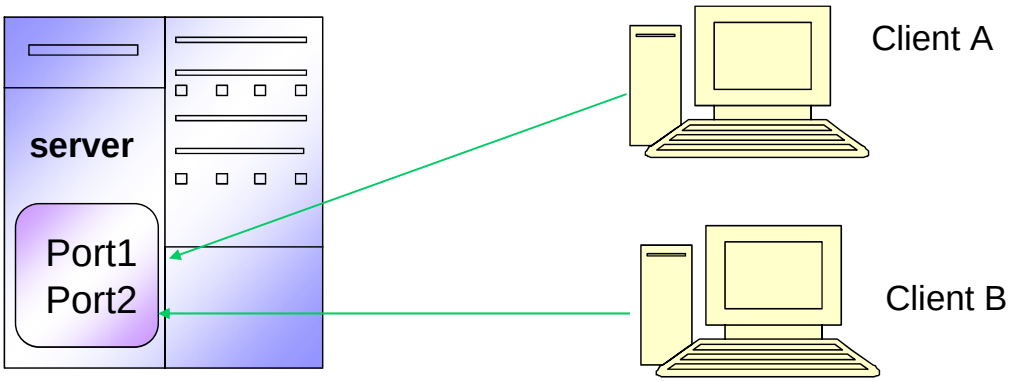
- **int getsockname(int s , struct sockaddr * name , socklen_t * namelen)**
  - returns the current name for the specified socket
- **int getpeername(int s, struct sockaddr *name, socklen_t *namelen)**
  - returns the name of the peer connected to socket s

BBT

Socket

- R/W with socket
  - recv/send
  - read/write
  - recvfrom/sendto
- close a socket
  - int close(int socket);
  - int shutdown(int s, int how);
    - how is 0, further receives will be disallowed.
    - how is 1, further sends will be disallowed.
    - how is 2, further sends and receives will be disallowed.

I/O Multiplexing—select()



- How to listen to two ports at the same time in a process??
- One of the solution: Use Non-blocking I/O
  - very complicated
- Another solution:Use select() in blocking I/O

BBT

select()

- Include sys/select.h and sys/time.h
- Use in standard I/O or socket I/O
- Type
  - fd_set mask
  - struct timeval {
    long tv_sec;
    long tv_usec;
    }

Useful Functions

- Use to check the readability of the descriptor, return while one of fd is ready to read or timeout
  - **int select(int max_fdp, fd_set * rmask, fd_set *, fd_set *, struct timeval *timeout);**
- Clear all bits in fdset
  - **FD_ZERO(fd_set *mask);**
- Turn on the bit for fd in fdset
  - **FD_SET(int fd , fd_set *mask);**
- Check if the bit for fd on in fdset
  - **int FD_ISSET(int fd, fd_set *mask);**

Socket options

- **int getsockopt(int s, int level, int optname, void *opt-val, socklen_t *optlen);**

- **int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);**

level

- SOL_SOCKETS
- IPPROTO_IP
- IPPROTO_TCP

BBT

Socket options (SOL_SOCKETS)

- **SO_DEBUG**
  - enables recording of debugging information
- **SO_REUSEADDR**
  - enables local address reuse
- **SO_KEEPALIVE**
  - enables keep connections alive
- **SO_DONTROUTE**
  - enables routing bypass for outgoing messages
- **SO_LINGER**
  - linger on close if data present

Socket options –cont.

- **SO_BROADCAST**
  - enables permission to transmit broadcast messages
- **SO_OOBINLINE**
  - enables reception of out-of-band data in band
- **SO_SNDBUF**
  - set buffer size for output
- **SO_RCVBUF**
  - set buffer size for input
- **SO_SNDLOWAT**
  - set minimum count for output
- **SO_RCVLOWAT**
  - set minimum count for input

BBT

Socket options –cont.

- **SO_SNDTIMEO**
  - get timeout value for output (get only)
- **SO_RCVTIMEO**
  - get timeout value for input (get only)
- **SO_TYPE**
  - get the type of the socket (get only)
- **SO_ERROR**
  - get and clear error on the socket (get only)

SO_LINGER

```
struct linger{
    int l_noff;
    int l_linger;  /* linger time as seconds.
};
```

- if l_onoff is 0, the option is turned off.
- if l_onoff is nonzero and l_linger is 0, TCP aborts the connection when it is closed. That is TCP discards any data still remaining in the socket send buffer and sends an RST to the peer, not the normal four-packet connection termination sequence.

SO_LINGER –cont.

- If l_onoff is nonzero and l_linger is nonzero, then the kernel will linger when the socket is closed.
  - That is if there is any data still remaining in the socket send buffer, the process is put to sleep until either all the data is send and acknowledged by the peer TCP, or the linger time expires.
  - It is important for the application to check the return value from close, because if the linger time expires before the remaining data is sent and acknowledged, close returns EWOULDBLOCK and remaining data in the send buffer is discarded.

TCP options (IPPROTO_TCP)

- **TCP_NODELAY**
  - Turn the Nagle algorithm off. This means that packets are always sent as soon as possible and no unnecessary delays are introduced, at the cost of more packets in the network.
- **TCP_MAXSEG**
  - Set or receive the maximum segment size for outgoing TCP packets. Values greater than the interface MTU are ignored and have no effect.
- **TCP_KEEPALIVE**
  - It specifies the idle time in seconds for the connection before TCP starts sending keepalive probes. The default value must be at least 7200 seconds. This option is effective only when the SO_KEPALIVE socket option is enabled.