ISO Defined Types of Network Service:

Type A:

Network connection with acceptable residual error rate and acceptable rate of signaled failures.

- Reliable, sequencing network service with arbitrary message size.

- Reliable, nonsequencing network service with arbitrary message size.

- Reliable, nonsequencing network service with maximum message size.

Type B:

Network connections with acceptable residual error rate but unacceptable rate of signaled failures.

Type C:

Network connections with residual error rate not acceptable to the transport service user.
Transport Layer Issues

Reliable Sequencing Network Service

Addressing:
How to identify the target user.

Multiplexing:
Mapping transport connections onto network connections.

Flow Control:
Receiver Mechanisms:
- Discard TPDUs that overflow the buffer.
- Refuse to accept further TPDUs from the network.
- Use a sliding window protocol.
- Use a credit allocation mechanism.

Connection Establishment and Termination:
- Allows each end to assure that the other exists.
- Allows negotiation of optional parameters.
- Triggers allocation of transport entity resources.
Credit Allocation Mechanism

Example of Credit Allocation Mechanism
Simple Connection State Diagram
Reliable Nonsequencing Network Service

TPDUs may arrive out of order!

- Sequence numbers are required for connection oriented service.
- Credit allocations must be numbered sequentially.
- Data TPDUs may arrive before connection is fully opened. (Data TPDUs should be queued until connection is fully established.)
- TPDUs may arrive after a connection has been closed. (Solution is to have the connection termination message contain the number of the last data TPDU sent on this connection.)

Example: Flow Control with a Nonsequenced Network Service
Reliable Nonsequencing Network Service
with Maximum TPDU Size

Fragmentation and reassembly required at the transport layer.
Failure Prone Networks (Type B)

The transport entity must cope with the problem of recovering from known loss of data and/or network connections.

Note: the sequence numbering is an effective tool for dealing with network failures.

Examples:

X.25 Reset is Received:

- Issue a control TPDU to the other end that acknowledges a reset condition and gives the number of the last data TPDU received.

- Refrain from issuing new data TPDUs until a corresponding reset control TPDU is received from the other end.

X.25 Restart is Received:

- Issue a request to the network service for a new network connection.

- Issue a control TPDU to the other end to identify the new network connection for the ongoing transport connection.

- Resynchronize with the use of reset control TPDUs.
Unreliable Network Service (Type C)

A TPDU may fail to arrive at the destination and go undetected!

Retransmission Strategy:
- Must have a positive ACK scheme.
- Must have a Retransmission Timer.

Duplicate Detection:
- Duplicate Received Prior to the Close of a Connection:
  a) The receiver must assume that its ACK was lost and send another ACK.
  b) The sequence number space must be long enough so as not to cycle in less than the maximum possible TPDU lifetime.
- Duplicate Received After the Close of a Connection:
  a) Sequence numbers should extend across connection lifetimes (TCP).
  b) Provide a separate transport connection identifier and use a new identifier for each new connection.
  c) If a system crash occurs, the above two schemes can fail. One could use a Reconnection Timer to insure a minimum time between closing one connection and opening another with the same destination.
Example of Incorrect Duplicate Detection

A times out and retransmits DT 0
A times out and retransmits DT 1

Example of Incorrect Duplicate Detection

Obsolete DT arrives
Flow Control:
- The credit allocation mechanism works well.
- Add a Window Timer to provide a maximum time between ACK/CREDIT TPDU's.

Connection Establishment:
- Since Connection Requests can get lost, retransmissions can cause delayed duplicates.
- Must use a three-way-handshake to open connections.

Connection Termination:
- Each side must explicitly ACK each other's close message.
- The close message must contain the sequence number of the last data TPDU sent.

Crash Recovery:
- Connections can become half-open.
- Use a Give-Up Timer to abort nonresponsive connections.
Revised Connection State Diagram
Examples of a Three-Way Handshake

(a) Normal operation

- RPC X
- RPC Y, ACK X
- DATA Y, ACK Y

(b) Delayed RPC

- RPC X
- RPC Y, ACK X
- RST, ACK Y
- DATA X, ACK Y
- RPC Z, ACK W

(c) Delayed RPC/ACK

- RPC X
- RPC Y, ACK X
- RST, ACK W
- DATA X, ACK Y
- RPC Z, ACK W

COMMENT

- A initiates a connection
- B accepts and acknowledges
- A acknowledges and begins transmission
- Old RPC arrives
- B accepts and acknowledges
- A rejects B's connection
- A initiates a connection
- B accepts and acknowledges
- Old RPC/ACK arrives, A rejects
- A acknowledges and begins transmission
Example: A Simple Transport Protocol

Primitives for a Simple Transport Service:

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Packet sent</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISTEN</td>
<td>(none)</td>
<td>Block until some process tries to connect</td>
</tr>
<tr>
<td>CONNECT</td>
<td>CONNECTION REQ.</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>SEND</td>
<td>DATA</td>
<td>Send information</td>
</tr>
<tr>
<td>RECEIVE</td>
<td>(none)</td>
<td>Block until a DATA packet arrives</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>DISCONNECTION REQ.</td>
<td>This side wants to release the connection</td>
</tr>
</tbody>
</table>

Network Layer Packets Used in the Example:

<table>
<thead>
<tr>
<th>Network packet</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL REQUEST</td>
<td>Sent to establish a connection</td>
</tr>
<tr>
<td>CALL ACCEPTED</td>
<td>Response to CALL REQUEST</td>
</tr>
<tr>
<td>CLEAR REQUEST</td>
<td>Sent to release a connection</td>
</tr>
<tr>
<td>CLEAR CONFIRMATION</td>
<td>Response to CLEAR REQUEST</td>
</tr>
<tr>
<td>DATA</td>
<td>Used to transport data</td>
</tr>
<tr>
<td>CREDIT</td>
<td>Control packet for managing the window</td>
</tr>
</tbody>
</table>
Finite State Machine Representation

<table>
<thead>
<tr>
<th>Primitives</th>
<th>State</th>
<th>Idle</th>
<th>Waiting</th>
<th>Queued</th>
<th>Established</th>
<th>Sending</th>
<th>Receiving</th>
<th>Disconnect</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISTEN</td>
<td>P1: -/Idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P2: A1/Estab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P2: A2/Idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONNECT</td>
<td>P1: -/Idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P1: A3/Wait</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>P4: A5/Idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4: A6/Disc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEND</td>
<td>P5: A7/Estab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P5: A8/Send</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECEIVE</td>
<td>A9/Receiving</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Call_req</td>
<td>P3: A1/Estab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3: A4/Que'd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Call_acc</td>
<td>-/Estab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear_req</td>
<td>-/Idle</td>
<td></td>
<td></td>
<td></td>
<td>A10/Estab</td>
<td>A10/Estab</td>
<td>A10/Estab</td>
<td>-/Idle</td>
</tr>
<tr>
<td>Clear_conf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataPkt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A12/Estab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Credit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A11/Estab</td>
<td>A7/Estab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timeout</td>
<td>-/Idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Predicates
- P1: Connection table full
- P2: Call_req pending
- P3: LISTEN pending
- P4: Clear_req pending
- P5: Credit available

Actions
- A1: Send Call_acc
- A2: Wait for Call_req
- A3: Send Call_req
- A4: Start timer
- A5: Send Clear_conf
- A6: Send Clear_req
- A7: Send message
- A8: Wait for credit
- A9: Send credit
- A10: SetClr_req_received_flag
- A11: Record credit
- A12: Accept message
A Simple Transport Protocol

```c
#define MAX_CONN 32    /* max number of simultaneous connections */
#define MAX_MSG_SIZE 8192 /* largest message in bytes */
#define MAX_PKT_SIZE 512  /* largest packet in bytes */
#define TIMEOUT 20
#define CRED 1
#define OK 0

#define ERR_FULL -1
#define ERR_REJECT -2
#define ERR_CLOSED -3
#define LOW_ERR -3

typedef int transport_address;
typedef enum {CALL_REQ,CALL_ACC,CLEAR_REQ,CLEAR_CONF,DATA_PKT,CREDIT} pkt_type;
typedef enum {IDLE,WAITING,QUEUED,ESTABLISHED,SENDING,RECEIVING,DISCONN} cstate;

/* Global variables. */
transport_address listen_address; /* local address being listened to */
int listen_conn;                  /* connection identifier for listen */
unsigned char data[MAX_PKT_SIZE];/* scratch area for packet data */

struct conn {
    transport_address local_address, remote_address;
    cstate state;          /* state of this connection */
    unsigned char *user_buf_addr; /* pointer to receive buffer */
    int byte_count;        /* send/receive count */
    int clr_req_received; /* set when CLEAR_REQ packet received */
    int timer;             /* used to time out CALL_REQ packets */
    int credits;           /* number of messages that may be sent */
} conn[MAX_CONN + 1];       /* slot 0 is not used */

void sleep(void);            /* prototypes */
void wakeup(void);
void to_net(int cid, int q, int m, pkt_type pt, unsigned char *p, int bytes);
void from_net(int *cid, int *q, int *m, pkt_type *pt, unsigned char *p, int *bytes);

int listen(transport_address t)
{ /* User wants to listen for a connection. See if CALL_REQ has already arrived. */
    int i, found = 0;

    for (i = 1; i <= MAX_CONN; i++) /* search the table for CALL_REQ */
        if (conn[i].state == QUEUED && conn[i].local_address == t) {
            found = i;
            break;
        }

    if (found == 0) {
        /* No CALL_REQ is waiting. Go to sleep until arrival or timeout. */
        listen_address = t; sleep(); i = listen_conn ;
    }
    conn[i].state = ESTABLISHED; /* connection is ESTABLISHED */
    conn[i].timer = 0;            /* timer is not used */
```
A Simple Transport Protocol cont.

```c
listen_conn = 0; /* 0 is assumed to be an invalid address */
to_net(i, 0, 0, CALL_ACC, data, 0); /* tell net to accept connection */
return(i); /* return connection identifier */
}

int connect(transport_address l, transport_address r)
{ /* User wants to connect to a remote process; send CALL_REQ packet. */
    int i;
    struct conn *cptr;

data[0] = r; data[1] = l; /* CALL_REQ packet needs these */
i = MAX_CONN; /* search table backward */
while (conn[i].state != IDLE && i > 1) i = i - 1;
if (conn[i].state == IDLE) {
    /* Make a table entry that CALL_REQ has been sent. */
cptr = &conn[i];
cptr->local_address = l; cptr->remote_address = r;
cptr->state = WAITING; cptr->clr_req_received = 0;
cptr->credits = 0; cptr->timer = 0;
to_net(i, 0, 0, CALL_REQ, data, 2);
sleep(); /* wait for CALL_ACC or CLEAR_REQ */
if (cptr->state == ESTABLISHED) return(i);
if (cptr->clr_req_received) {
    /* Other side refused call. */
cptr->state = IDLE; /* back to IDLE state */
to_net(i, 0, 0, CLEAR_CONF, data, 0);
return(ErrRejected);
}
} else return(ErrFull); /* reject CONNECT: no table space */
}

int send(int cid, unsigned char buft[ ], int bytes)
{ /* User wants to send a message. */
    int i, count, m;
    struct conn *cptr = &conn[cid];

    /* Enter SENDING state. */
cptr->state = SENDING;
cptr->byte_count = 0; /* # bytes sent so far this message */
if (cptr->clr_req_received == 0 && cptr->credits == 0) sleep();
if (cptr->clr_req_received == 0) {
    /* Credit available; split message into packets if need be. */
do {
        if (bytes - cptr->byte_count > MAX_PKT_SIZE) {/* multipacket message */
            count = MAX_PKT_SIZE; m = 1; /* more packets later */
        } else {/* single packet message */
            count = bytes - cptr->byte_count; m = 0; /* last pkt of this message */
        }
for (i = 0; i < count; i++) data[i] = buft[cptr->byte_count + i];
to_net(cid, 0, 0, DATA_PKT, data, count); /* send 1 packet */
cptr->byte_count = cptr->byte_count + count; /* increment bytes sent so far */
} while (cptr->byte_count < bytes); /* loop until whole message sent */
```
A Simple Transport Protocol cont.

cptr->credits--;            /* each message uses up one credit */
cptr->state = ESTABLISHED;
    return(OK);
} else {
    cptr->state = ESTABLISHED;
    return(ERR_CLOSED);         /* send failed: peer wants to disconnect */
}

int receive(int cid, unsigned char bufptr[], int *bytes)
{    /* User is prepared to receive a message. */
    struct conn *cptr = &conn[cid];

    if (cptr->clr_req_received == 0) {
        /* Connection still established; try to receive. */
        cptr->state = RECEIVING;
        cptr->user_buf_addr = bufptr;
        cptr->byte_count = 0;
        data[0] = CRED;
        data[1] = 1;
        to_net(cid, 1, 0, CREDIT, data, 2);    /* send credit */
        sleep();                               /* block awaiting data */
        *bytes = cptr->byte_count;
    }
    cptr->state = ESTABLISHED;
    return(cptr->clr_req_received ? ERR_CLOSED : OK);
}

int disconnect(int cid)
{    /* User wants to release a connection. */
    struct conn *cptr = &conn[cid];

    if (cptr->clr_req_received) {        /* other side initiated termination */
        cptr->state = IDLE;             /* connection is now released */
        to_net(cid, 0, 0, CLEAR_CONF, data, 0);
    } else {                           /* we initiated termination */
        cptr->state = DISCONN;         /* not released until other side agrees */
        to_net(cid, 0, 0, CLEAR_REQ, data, 0);
    }
    return(OK);
}

void packet_arrival(void)
{    /* A packet has arrived, get and process it. */
    int cid;
    /* connection on which packet arrived */
    int count, i, q, m;
    pkt_type ptype;        /* CALL_REQ, CALL_ACC, CLEAR_REQ, CLEAR_CONF, DATA_PKT, CREDIT */
    unsigned char data[MAX_PKT_SIZE];    /* data portion of the incoming packet */
    struct conn *cptr;

    from_net(&cid, &q, &m, &ptype, data, &count);    /* go get it */
    cptr = &conn[cid];
A Simple Transport Protocol cont.

```
switch (ptype) {
    case CALL_REQ: /* remote user wants to establish connection */
        cptr->local_address = data[0]; cptr->remote_address = data[1];
        if (cptr->local_address == listen_address) {
            listen_conn = cid; cptr->state = ESTABLISHED; wakeup();
        } else {
            cptr->state = QUEUED; cptr->timer = TIMEOUT;
        }
        cptr->clr_req_received = 0; cptr->credits = 0;
        break;
    case CALL_ACC: /* remote user has accepted our CALL_REQ */
        cptr->state = ESTABLISHED;
        wakeup();
        break;
    case CLEARREQ: /* remote user wants to disconnect or reject call */
        cptr->clr_req_received = 1;
        if (cptr->state == DISCONN) cptr->state = IDLE; /* clear collision */
        if (cptr->state == WAITING || cptr->state == RECEIVING || cptr->state == SENDING) wakeup();
        break;
    case CLEARCONF: /* remote user agrees to disconnect */
        cptr->state = IDLE;
        break;
    case CREDIT: /* remote user is waiting for data */
        cptr->credits += data[1];
        if (cptr->state == SENDING) wakeup();
        break;
    case DATA_PKT: /* remote user has sent data */
        for (i = 0; i < count; i++) cptr->user_buf_addr[cptr->byte_count + i] = data[i];
        cptr->byte_count += count;
        if (m == 0) wakeup();
}
}

void clock(void)
{ /* The clock has ticked, check for timeouts of queued connect requests. */
    int i;
    struct conn *cptr;
    for (i = 1; i <= MAX_CONN; i++) {
        cptr = &conn[i];
        if (cptr->timer > 0) { /* timer was running */
            cptr->timer--;
            if (cptr->timer == 0) { /* timer has now expired */
                cptr->state = IDLE;
                to_net(i, 0, 0, CLEAR_REQ, data, 0);
            }
        }
    }
}
```
Transport Protocols

! Transmission Control Protocol (TCP)
   Connection-Oriented

! User Datagram Protocol (UDP)
   Connectionless

! TCP Services:
   - Provides reliable communications across reliable and unreliable networks and internets.
   - Stream oriented.
   - Supports security and precedence labeling:
     - Data Stream Push
     - Urgent Data Signaling
   - Designed specifically and exclusively to work with IP.

! UDP Services:
   - Provides a transport-level, unreliable, datagram service.
   - Delivery and duplicate detection are not guaranteed.
TCP Service Request Primitives

UNSPECIFIED-PASSIVE-OPEN:
Listen for connection attempts at specified security and precedence levels from any remote user.

FULL-PASSIVE-OPEN:
Listen for connection attempts at specified security and precedence levels from specified user.

ACTIVE-OPEN:
Request connection at particular security and precedence levels.

ACTIVE-OPEN-WITH-DATA:
Request connection at particular security and precedence levels and transmit data with request.

SEND:
Transfer data across named connection.

ALLOCATE:
Issue incremental allocation for receive data to TCP.

CLOSE:
Close connection gracefully.

ABORT:
Close connection abruptly.

STATUS:
Report connection status.
TCP Service Response Primitives

OPEN-ID:
Informs user of connection name assigned to pending connection requested in an OPEN primitive.

OPEN-FAILURE:
Reports failure of an ACTIVE-OPEN or request.

OPEN-SUCCESS:
Reports completion of an ACTIVE-OPEN request.

DELIVER:
Reports arrival of data.

CLOSING:
Reports that remote TCP user has issued a CLOSE.

TERMINATE:
Reports that connections has been terminated and no longer exists.

STATUS-RESPONSE:
Reports current status of connection.
<table>
<thead>
<tr>
<th>TCP Service Request Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNSPECIFIED-PASSIVE-OPEN (source-port, [timeout], [timeout-action], [precedence], [security])</td>
</tr>
<tr>
<td>FULL-PASSIVE-OPEN (source-port, destination-port, destination-address, [timeout], [timeout-action], [precedence], [security])</td>
</tr>
<tr>
<td>ACTIVE-OPEN (source-port, destination-port, destination-address, [timeout], [timeout-action], [precedence], [security])</td>
</tr>
<tr>
<td>ACTIVE-OPEN-WITH-DATA (source-port, destination-port, destination-address, [timeout], [timeout-action], [precedence], [security], data, data-length, push-flag, urgent-flag)</td>
</tr>
<tr>
<td>SEND (local-connection-name, data, data-length, push-flag, urgent-flag, [timeout], [timeout-action])</td>
</tr>
<tr>
<td>ALLOCATE (local-connection-name, data-length)</td>
</tr>
<tr>
<td>CLOSE (local-connection-name)</td>
</tr>
<tr>
<td>ABORT (local-connection-name)</td>
</tr>
<tr>
<td>STATUS (local-connection-name)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCP Service Response Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN-ID (local-connection-name, source-port, destination-port, destination-address)</td>
</tr>
<tr>
<td>OPEN-FAILURE (local-connection-name)</td>
</tr>
<tr>
<td>OPEN-SUCCESS (local-connection-name)</td>
</tr>
<tr>
<td>DELIVER (local-connection-name, data, data-length, urgent-flag)</td>
</tr>
<tr>
<td>CLOSING (local-connection-name)</td>
</tr>
<tr>
<td>TERMINATE (local-connection-name, description)</td>
</tr>
<tr>
<td>STATUS-RESPONSE (local-connection-name, source-port, source-address, destination-port, destination-address, connection-state, receive-window, send-window, amount-waiting-ack, amount-waiting-receipt, urgent-mode, timeout, timeout-action)</td>
</tr>
</tbody>
</table>
# TCP TPDU Format

<table>
<thead>
<tr>
<th>0</th>
<th>15</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acknowledge Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Offset</td>
<td>reserved</td>
<td>Flags</td>
<td>Window</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent Pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options and Padding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TCP Header Parameters

Source Port (16 bits):
- Source SAP.

Destination Port (16 bits):
- Destination SAP.

Sequence Number (32 bits):
- Seq. Number of the first data octet in this TPDU.
- When SYN is present, it is the initial sequence number (ISN) (first data octet is ISN+ 1).

Acknowledgement Number (32 bits):
- Next octet expected.

Data Offset (4 bits):
- Number of 32-bit words in header.

Reserved (6 bits): (for future use)

Flags:
- URG: Urgent pointer field significant
- ACK: Acknowledgement field significant
- PSH: Push function
- RST: Reset the connection
- SYN: Synchronize the sequence numbers
- FIN: No more data from sender

Window (16 bits):
- Flow control credit allocation in octets.
- Number of octets, beginning with the one specified in the ack field that the sender will accept.

Checksum (16 bits):
- 1's complement of the sum of all the 16-bit words.

Urgent Pointer (16 bits):
- Points to the octet following the urgent data.

Options (variable):
- At present only option is maximum size of TPDU.
Parameters passed to IP:

! Precedence: a 3-bit field
! Normal delay / low delay
! Normal throughput / high throughput
! Normal reliability / high reliability
! Security: an 11 bit field
TCP Mechanisms

Connection Establishment:
! Always uses a Three-Way Handshake:
- SYN
- SYN + ACK
- ACK

Data Transfer:
! Viewed as a stream of octets.
! Each octet is numbered.
! Flow control using a credit allocation scheme (in octets).
! TCP normally exercises its own discretion when to construct a TPDU - unless PUSH flag is used).
! RST flag when apparently incorrect data for current connection is received. (e.g. delayed duplicate SYN, acks for data not yet sent, etc.).

Connection Termination:
! Graceful Close:
- Transport user issues CLOSE primitive.
- Transport entity sets the FIN bit on last TPDU sent.

! Abrupt Termination:
- User issues an ABORT primitive.
- Entity abandons all efforts to send or receive data.
- Entity discards data in its buffers.
- Entity sends out a RST TPDU.
The TCP Entity State Diagram

TCP entity state diagram.
Example of TCP Operation

CLIENT SIDE

- ACTIVE_OPEN
- OPEN_SUCC.
- SEND (N bytes) (Push)

SERVER SIDE

- PASSIVE_OPEN
- OPEN_RCVD
- DELIVER

SYN = 1 Seq = X
SYN = 1 ACK = 1 Seq = Y Ack = X+1
ACK = 1 Seq = X+1 Ack = Y+1
ACK = 1 PUSH = 1 Seq = X+1 Ack = Y+1
ACK = 1 Seq = Y+1 Ack = X+N+1 Window = N
Example of TCP Operation cont.

CLIENT SIDE

DELIVER

ACK = 1 Seq = Y+1 Ack = X+N+1

ACK = 1 Seq = Y+W+1 Ack = X+N+1

Window = W

DELIVER

ACK = 1 Seq = Y+2W+1 Ack = X+N+1

ACK = 1 Ack = Y+2W+1

Window = W

DELIVER

ACK = 1 Seq = X+N+1 Ack = Y+3W+1 Window = W

SERVER SIDE

SEND (3W byte)
Example of TCP Operation cont.

CLIENT SIDE

CLOSE

FIN = 1 Seq = X+N+1

ACK = 1 Seq = Y+3W+1 Ack = X+N+1

ACK = 1 FIN = 1 Seq = Y+3W+1

ACK = 1 Ack = Y+4W+1

TERMINATE

SERVER SIDE

SEND (W bytes)

CLOSING

CLOSE (Sends W bytes)

TERMINATE
### Currently Assigned TCP Port Numbers

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Keyword</th>
<th>UNIX Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TCPPMUX</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>RJE</td>
<td>-</td>
<td>TCP Multiplexor</td>
</tr>
<tr>
<td>5</td>
<td>ECHO</td>
<td>echo</td>
<td>Remote Job Entry</td>
</tr>
<tr>
<td>7</td>
<td>DISCARD</td>
<td>discard</td>
<td>Echo</td>
</tr>
<tr>
<td>9</td>
<td>USERS</td>
<td>systat</td>
<td>Active Users</td>
</tr>
<tr>
<td>11</td>
<td>DAYTIME</td>
<td>daytime</td>
<td>Daytime</td>
</tr>
<tr>
<td>13</td>
<td>-</td>
<td>netstat</td>
<td>Network status program</td>
</tr>
<tr>
<td>15</td>
<td>QUOTE</td>
<td>qotd</td>
<td>Quote of the Day</td>
</tr>
<tr>
<td>17</td>
<td>CHARGEN</td>
<td>chargen</td>
<td>Character Generator</td>
</tr>
<tr>
<td>19</td>
<td>FTP-DATA</td>
<td>ftp-data</td>
<td>File Transfer Protocol (data)</td>
</tr>
<tr>
<td>20</td>
<td>FTP</td>
<td>ftp</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>21</td>
<td>TELNET</td>
<td>telnet</td>
<td>Terminal Connection</td>
</tr>
<tr>
<td>23</td>
<td>SMTP</td>
<td>smtp</td>
<td>Simple Mail Transport Protocol</td>
</tr>
<tr>
<td>25</td>
<td>TIME</td>
<td>time</td>
<td>Time</td>
</tr>
<tr>
<td>37</td>
<td>NAMERSERVER</td>
<td>name</td>
<td>Host Name Server</td>
</tr>
<tr>
<td>42</td>
<td>NICNAME</td>
<td>whois</td>
<td>Who Is</td>
</tr>
<tr>
<td>43</td>
<td>DOMAIN</td>
<td>nameserver</td>
<td>Domain Name Server</td>
</tr>
<tr>
<td>53</td>
<td>-</td>
<td>rje</td>
<td>any private RJE service</td>
</tr>
<tr>
<td>77</td>
<td>FINGER</td>
<td>finger</td>
<td>Finger</td>
</tr>
<tr>
<td>93</td>
<td>DCP</td>
<td>supdup</td>
<td>Device Control Protocol</td>
</tr>
<tr>
<td>95</td>
<td>SUPDUP</td>
<td>supdup</td>
<td>SUPDUP Protocol</td>
</tr>
<tr>
<td>101</td>
<td>HOSTNAME</td>
<td>hostnames</td>
<td>NIC Host Name Server</td>
</tr>
<tr>
<td>102</td>
<td>ISO-TSAP</td>
<td>iso-tsap</td>
<td>ISO-TSAP</td>
</tr>
<tr>
<td>103</td>
<td>X400</td>
<td>x400</td>
<td>X.400 Mail Service</td>
</tr>
<tr>
<td>104</td>
<td>X400-SND</td>
<td>x400-snd</td>
<td>X.400 Mail Sending</td>
</tr>
<tr>
<td>111</td>
<td>SUNRPC</td>
<td>sunrpc</td>
<td>SUN Remote Procedure Call</td>
</tr>
<tr>
<td>113</td>
<td>AUTH</td>
<td>auth</td>
<td>Authentication Service</td>
</tr>
<tr>
<td>117</td>
<td>UUCP-PATH</td>
<td>uucp-path</td>
<td>UUCP Path Service</td>
</tr>
<tr>
<td>119</td>
<td>NNTP</td>
<td>nntp</td>
<td>USENET News Transfer Protocol</td>
</tr>
<tr>
<td>129</td>
<td>PWDGEN</td>
<td>-</td>
<td>Password Generator Protocol</td>
</tr>
<tr>
<td>139</td>
<td>NETBIOS-SSN</td>
<td>-</td>
<td>NETBIOS Session Service</td>
</tr>
<tr>
<td>160-223</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples of currently assigned TCP port numbers. To the extent possible, protocols like UDP use the same numbers.
# UDP TPDU Format

<table>
<thead>
<tr>
<th>0</th>
<th>15</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td>Checksum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Currently Assigned UDP Port Numbers

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Keyword</th>
<th>UNIX Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>ECHO</td>
<td>echo</td>
<td>Echo</td>
</tr>
<tr>
<td>9</td>
<td>DISCARD</td>
<td>discard</td>
<td>Discard</td>
</tr>
<tr>
<td>11</td>
<td>USERS</td>
<td>systat</td>
<td>Active Users</td>
</tr>
<tr>
<td>13</td>
<td>DAYTIME</td>
<td>daytime</td>
<td>Daytime</td>
</tr>
<tr>
<td>15</td>
<td>-</td>
<td>netstat</td>
<td>Who is up or NETSTAT</td>
</tr>
<tr>
<td>17</td>
<td>QUOTE</td>
<td>qotd</td>
<td>Quote of the Day</td>
</tr>
<tr>
<td>19</td>
<td>CHARGEN</td>
<td>chargen</td>
<td>Character Generator</td>
</tr>
<tr>
<td>37</td>
<td>TIME</td>
<td>time</td>
<td>Time</td>
</tr>
<tr>
<td>42</td>
<td>NAMESERVER</td>
<td>name</td>
<td>Host Name Server</td>
</tr>
<tr>
<td>43</td>
<td>NICNAME</td>
<td>whois</td>
<td>Who Is</td>
</tr>
<tr>
<td>53</td>
<td>DOMAIN</td>
<td>nameserver</td>
<td>Domain Name Server</td>
</tr>
<tr>
<td>67</td>
<td>BOOTPS</td>
<td>bootps</td>
<td>Bootstrap Protocol Server</td>
</tr>
<tr>
<td>68</td>
<td>BOOTPC</td>
<td>bootpc</td>
<td>Bootstrap Protocol Client</td>
</tr>
<tr>
<td>69</td>
<td>TFTP</td>
<td>tftp</td>
<td>Trivial File Transfer</td>
</tr>
<tr>
<td>111</td>
<td>SUNRPC</td>
<td>sunrpc</td>
<td>Sun Microsystems RPC</td>
</tr>
<tr>
<td>123</td>
<td>NTP</td>
<td>ntp</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>161</td>
<td>-</td>
<td>snmp</td>
<td>SNMP net monitor</td>
</tr>
<tr>
<td>162</td>
<td>-</td>
<td>snmp-trap</td>
<td>SNMP traps</td>
</tr>
<tr>
<td>512</td>
<td>-</td>
<td>biff</td>
<td>UNIX comsat</td>
</tr>
<tr>
<td>513</td>
<td>-</td>
<td>who</td>
<td>UNIX rwho daemon</td>
</tr>
<tr>
<td>514</td>
<td>-</td>
<td>syslog</td>
<td>system log</td>
</tr>
<tr>
<td>525</td>
<td>-</td>
<td>timed</td>
<td>Time daemon</td>
</tr>
</tbody>
</table>

An illustrative sample of currently assigned UDP ports showing the standard keyword and the UNIX equivalent; the list is not exhaustive. To the extent possible, other transport protocols that offer identical services use the same port numbers as UDP.