Monitoring TCP Connections

- TCP does not provide an application with **immediate** notification when a connection is lost due to network outages or system crashes.

- Under these circumstances an application that is sending data to its peer may not discover the loss of connectivity until TCP has exhausted all of its retries. This can take considerable time-approximately 9 minutes for BSD-derived systems.

- Thus it is up to the application programmer to implement code to detect loss of connectivity with minimal delay.

- They accomplish this using a dedicated point-to-point link and a polling protocol to test continually for the presence of its peer.

- This may take the form of explicit data query messages such as those used in poll-select protocols, or they may take the form of background packets that continually monitor the presence of the virtual circuit.

- The price that is paid for these techniques is in network bandwidth. Each of these polling messages consume network resources that could otherwise be used to carry user data.

- The cost in available network bandwidth is one reason for TCP not to provide immediate notification of connectivity loss.

- Most applications have no need for immediate notification, and so they should not have to pay for it with decreased bandwidth. Applications that do need to know in a timely manner when their peers become unreachable, can implement their own mechanism for ascertaining connectivity.

- One of the fundamental TCP/IP design principles is that all of the network intelligence should be as close as possible to the end points of a connection and that the network itself should be relatively dumb.

- When applied to monitoring connectivity between peer applications, this principle means that the applications should provide the ability when as the need arises, rather than having it available and active for all applications, regardless of need.

Keep-Alives

- TCP does in fact have a mechanism called a keep-alive timer for detecting dead connections. However it is not useful to most applications.
- This feature is also a controversial topic in the TCP/IP community (read "religious debate") because many are of the opinion that end-to-end polling is application layer issue and not a transport protocol issue.
- When an application enables the keep-alive mechanism, TCP sends a special segment to its peer when the connection has been idle for 2 hours.
- If the peer host is reachable and the peer application is still running, the peer TCP responds with an ACK. In this case, the sender's TCP will reset the keep-alive timer for 2 hours into the future. This exchange is transparent to the application.
- If the peer host is reachable but the peer application is not running, the peer TCP stack responds with an RST and the TCP stack sending the keep-alive drops the connection and returns an ECONNRESET error to the application.
- This is normally the result of the peer host crashing and being rebooted, because if the peer application merely terminated or crashed, the peer TCP would have sent a FIN.
- If the peer host does not respond with an ACK or an RST, the TCP sending the keep-alive will timeout after 75 seconds.
- The sending stack then sends a total of 10 keep-alive probes, 75 seconds apart and if it still does not receive a response it decides that its peer is unreachable or has crashed.
- At that point it drops the connection and informs the application with an ETIMEDOUT or, if a router has returned an ICMP host or network unreachable error, with an EHOSTUNREACH or ENETUNREACH error.
- The main problem with keep-alives for applications that need immediate notification of a connectivity loss is the default timeout values.
- For BSD-derived implementations using the default values take 2 hours, 11 minutes, 15 seconds to discover that connectivity has been lost.
- This value makes more sense when we realize that keep-alives are intended to release the resources held by defunct connections.
- Such connections can come about, for example, when a client connects to a server and the client's host crashes. Without the keep-alive mechanism, the server waits forever for the client's next request, because it never receives a FIN.
- This situation is increasingly common due to users of PC-based systems merely turning off the computer or its modem instead of shutting down applications correctly.

- Some implementations allow one or both time intervals to be changed, but this is almost always on a system wide basis. The change affects all TCP connections on the system.

- The default time periods are too long, and if the default is changed, they are no longer useful for their original purpose of cleaning up long-dead connections.

- All these issues are the main reason that keep-alives are not really useful as a connection monitoring mechanism.

- There is a new POSIX socket option, TCP-KEEPALIVE, which does allow the timeout interval to be specified on a per-connection basis, but it is not widely implemented.

Heartbeats

- The problems with using keep-alives to monitor connectivity are best solved by implementing a mechanism within the application.

- The implementation would depend on the application. Consider the following two cases:

    1. A client and server that exchange several different types of messages, each having a packet header that identifies the message type.

    2. An application that provides data to its peer as a byte stream with no inherent boundaries or structure to the message.

- The first case is relatively easy. We can implement a new message type, MSG_HEARTBEAT, that one side can send to another.

- Upon receipt of the MSG_HEARBEAT message, the application merely returns the message to its peer.

- In this way one or both sides of the connection can monitor the channel for connectivity, with only one side actually sending the heartbeat.

- The following is the header file that is used by both client and server. The constants define the various message types that the client and server exchange.

```
#define MSG_DATA            1           // application specific msg
#define MSG_ACK             2           // ACK
#define MSG_HEARTBEAT       3           // heartbeat message
#define SERVER_TCP_PORT     7000        // Default port
#define BUFLEN              256         //Buffer length
#define TRUE                1

typedef struct                          /* message structure */
{
    u_int32_t type;                     /* MSG_TYPE1, ... */
    char data[ BUFLEN ];
} msg_t;

#define T0                      30      /* idle time before heartbeat */
#define T1                      5       /* time to wait for response */
```

- The **msg_t** data structure defines the packet structure used to exchange information. The timeout values selected here are arbitrary and in practice would be selected based on the application and type of network.
- The client performs the usual initialization and connects to the server at the host and port address specified on the command line.
- The select mask is established for the connected socket and time T1 is initialized.
- If the client does not receive a message within Tl seconds, select returns with a timer expiration.
- The read select mask is then established and then the program blocks in the call to select until data is received on the socket or until the timer expires.
- If more than three consecutive heartbeats have been transmitted without a response, the connection is assumed to be dead.
- In this example, we merely quit, but a real application could take whatever steps it deemed appropriate.
- If the heartbeat probes have not been exhausted yet, a heartbeat is sent to the peer and the timer is set to T2 seconds.
- If we do not receive a message from our peer within this amount of time, we either send another heartbeat or declare the connection dead depending on the value of heartbeats.
- A ***recv*** is executed to read up to one message. Since the client has just received a message from its peer the heartbeat counter is reset to zero, and the timer reset for TI seconds.

- The server performs the usual initialization and then starts accepting connections from clients.
- The initial timer value is set to Tl + T2 seconds. Because the peer sends a heartbeat after Tl seconds of inactivity, the server allows for some extra time by increasing the timeout value by T2 seconds.
- Next, the select mask is initialized for readability on the socket connected to the peer host.
- If the server misses more than three consecutive heartbeats, the connection is declared dead and the application exits.
- The timer is reset to T2 seconds. By now, the client should be sending its heartbeat every T2 seconds, and we time out after that amount of time so that we can count the missed heartbeats.
- As with the client, recv is called to read a message. Because a message has just been received from the client, the connection is still alive the missed heartbeat count is reset to zero and the timer value to Tl + T2 seconds.
- If the received packet is a heartbeat message, it is echoed back to the client. When the client receives this message, both sides know that the connection is still alive.
- The model that we have just implemented will not work very well when one side sends the other a stream of data that has no inherent message or record boundaries.
- The problem is that the heartbeat will just appear as part of the character stream and therefore other measure will be required to explicitly check for special messages such as heartbeats.
- To avoid a complicated design a completely different model can be implemented. This method uses a separate connection for the heartbeats.
- It may seem redundant to use one connection to monitor another, but the objective here is to detect either the crash of our peer's host or the unavailability of the network.
- If one of these events occurs, either both connections are affected or neither is. There are several out ways of approaching this problem.
- One natural way is to use a separate thread to control the heartbeat function.
- The previous code can be modified quite easily for this design. The major difference is the select logic, which must now handle two sockets.
- Additional logic to set up the extra connection will also be required. After the client connects to the server, it sends the server a port number on which it listens for the heartbeat connection from the server.
- This arrangement is similar to what the FTP server does when it establishes a data connection back to the client.