Computer Networks Practical Documentation

Jeffrey Klardie
VU University, Amsterdam

July 4, 2013

1 Architecture overview

The main architecture for this TCP stack is implemented in the classes TCP, SEGMENTHANDLER and TIMEOUTHANDLER. These classes comprise the global architecture of the TCP stack. The TCP class exposes a socket for the user to write to and read from, used to communicate with the other end. A background thread is continuously waiting for new packets, and passes them to the SEGMENTHANDLER. This class handles these packets based on the current state the TCP stack is in. The TIMEOUTHANDLER handles all timeouts that might occur inside the system. What follows is a detailed overview of these classes and their workings.

1.1 TCP

This class contains the main methods for handling the TCP stack. It exposes one public constructor, which creates the required underlying IP stack. Also, it exposes three methods:

- **public Socket socket()** creates a new TRANSMISSIONCONTROLBLOCK object, and returns a new client socket.
- **public Socket socket(int port)** creates a new TRANSMISSIONCONTROLBLOCK object, and returns a new server socket which will later use the given port to listen for incoming requests.
- **protected TransmissionControlBlock.State getState()** returns the current state the TCP stack is in. This method is only used for testing.

The TCP class also has two private methods for sending and receiving data:

- **private int send(byte[] buf, int offset, int len)**: This method tries to send *len* bytes from the given buffer *buf*, starting at *offset*. The number of bytes written and acknowledged is returned, or -1 in case of any error. The behavior follows the strict state machine as defined in RFC 793. The following can happen depending on the state the stack is in:

  - ESTABLISHED or CLOSE-WAIT: the data is segmentized (to make sure the maximum packet size is not overridden), the segments are sent, and the variable SND.NXT (next sequence number to use) is incremented. Now the method blocks, until all data has been acknowledged. If all data has been acknowledged, the amount of written data is returned, and -1 otherwise.
  - CLOSED: error, because the connection does not exist.
  - SYN-SENT or SYN-RECEIVED: error, because the connection is not established.
  - LISTEN: error, because switching from passive to active open is not supported.
  - other states: error, because the connection is closing.

- **private int receive(byte[] buf, int offset, int maxlen)**: This method tries to receive *maxlen* bytes from the other end, and writes them into the given buffer *buf*, starting at *offset*. The number of bytes read is returned, or -1 in case of any error. Just like send(), this method follows the strict TCP state machine. The following can happen depending on the state the stack is in:

  - ESTABLISHED, FIN-WAIT-1 or FIN-WAIT-2: If there is no data to process in the queue, block until there is data. When data is available, write this data to the given buffer, and return the number of bytes read.
  - CLOSE-WAIT: The remote side has already sent a FIN message, so the other end is done sending. This means that either data that was still in a queue is returned, or an error is thrown.
  - CLOSED: error, because the connection does not exist.
  - LISTEN, SYN-SENT or SYN-RECEIVED: error, because the connection is not established. In a full TCP implementation the receive request would be queued until the connection is established. In the current version this is not implemented though.
  - other states: error, because the connection is closing.

1.2 TCP.Socket

This is a subclass of TCP, and the only one that is exposed to the end user. It exposes a total of five methods used to open a connection, read or write data, and to close a connection.

- **public boolean connect(IpAddress dst, int port)**: This method tries to create a connection by sending the initial synchronize packet to start the three-way handshake. The method blocks until the connection is established. It returns true if the connection is established, or false if an error occurs. In detail, this method performs the following steps (if the state was closed):

  - Store the foreign socket information in the TRANSMISSIONCONTROLBLOCK object.
  - Initialize and starts the segment receiver. This is a thread that waits for incoming segments, and forwards them to the SEGMENTHANDLER object.
  - Generate the initial sequence number.

  - Construct an SYN packet and send it to the foreign socket.
  - Set the SND.NXT (next sequence number to use) and SND.UNA (last sequence number not acknowledged yet) variables.
  - Enter the state SYN-SENT.
  - Wait until the state changes to either ESTABLISHED (successfully connected) or CLOSED (error). This is possible because the SEGMENTHANDLER takes care of incoming messages, and therefore also the messages used in the three-way handshake.

- **public void accept()**: This method starts the segment receiver thread, and enters the LISTEN state. After that, it simply blocks until the state changes to ESTABLISHED. The SEGMENTHANDLER that runs in a background thread will handle incoming SYN packets, and respond to them depending on the state the TCP stack is in at that moment. It will correctly perform the three-way handshake if the state is LISTEN, and a SYN packet arrives.
- **public int read(byte[] buf, int offset, int maxlen)**: This method simply calls TCP.RECEIVE(BUF, OFFSET, MAXLEN).
- **public int write(byte[] buf, int offset, int len)**: This method simply calls TCP.SEND(BUF, OFFSET, LEN).
- **public boolean close()**: This method closes a connection and only returns true if the close succeeded. The execution is highly dependent on the current state the stack is in (remember that the SEGMENTHANDLER runs in the background to receive packets. Therefore it is possible to wait until a certain state is entered):
  - CLOSED: Error, because there is no active connection.
  - LISTEN: The state CLOSED is entered and any outstanding receives are stopped.
  - SYN-SENT: The state CLOSED is entered. Any queued sends and receives should be stopped. This implementation currently only stops outstanding receives.
  - SYN-RECEIVED: If we did not send anything, and there is also no data in the transmission queue, send the FIN packet and enter the state FIN-WAIT-1. Otherwise, wait until the connection state becomes ESTABLISHED, and execute the close() call again.
  - ESTABLISHED: Enter the state FIN-WAIT-1, wait until all data that has been sent is acknowledged, and send the FIN packet. Then wait until the connection state becomes CLOSED.
  - CLOSE-WAIT: Wait until all data that has been sent is acknowledged , send the FIN packet and enter the state LAST-ACK. Then wait until the connection state becomes CLOSED.
  - any other state: True is returned immediately, as the connection is already closing.

1.3 SegmentHandler

The SEGMENTHANDLER class is called by the SEGMENTRECEIVER class. The latter runs in the background, and blocks until a new IP packet arrives. The data part of the IP packet is decoded,

and the resulting segment is passed to the segment handler. It is started when a connection is opened (either `connect()` or `listen()` inside SOCKET are called).

The SEGMENTHANDLER responds to all incoming packets according to the current state the TCP stack is in. When a new segment arrives, first of all the expected checksum is calculated. If the received checksum is not matching the expected checksum, the packet is dropped. If the checksum is valid, the following happens based on the state:

- CLOSED: The packet is dropped, because the connection is closed.
- LISTEN: A SYN packet is expected, and if the packet contains any other control bit, the packet is dropped. When the packet is valid, the foreign socket info is stored, the variables RCV.NXT (next expected sequence number) and IRS (initial receive sequence number) are all set in the TRANSMISSIONCONTROLBLOCK object. Next, a SYN,ACK packet is sent in reply, the variables SND.NXT (next sequence number to send) and SND.UNA (last unacknowledged sequence number) are set, and the SYN-RECEIVED state is entered.
- SYN-SENT: The initial SYN packet is sent, so a matching SYN,ACK packet is expected. The incoming packet is dropped if it does not contain the SYN and ACK control bits, and also if the ACK number is incorrect. If this is all correct, the variables RCV.NXT and IRS are set. Also, the SND.UNA is updated (because we received a new ACK), and all packets that are hereby acknowledged are removed from the transmission queue. If this ACK actually acknowledges our SYN packet, then the ESTABLISHED state is entered and an ACK is sent in reply. Note that we do drop any data that was send along the SYN,ACK message.
- any other state: first the sequence number of the packet is checked. A valid sequence number falls within the window of expected packets (greater than the previous sequence number, and smaller than the previous sequence number + the maximum packet size). Packets with invalid sequence numbers are dropped. After that, the following happens:
  - If the packet contains the SYN control bit, it is dropped, because this is considered an error (we already synchronized sequence numbers before).
  - If the packet does not contain an ACK number, it is dropped. Besides the initial SYN packet, all packets require an ACK value. Otherwise, based on the state:
    - SYN-SENT: If this ACK acknowledges our SYN, we enter state ESTABLISHED.
    - LAST-ACK: Only thing that can arrive here is an acknowledgement to the sent FIN, so state CLOSED is entered.
    - TIME-WAIT: Only thing that can arrive here is a retransmission of the sent FIN. Here we restart the time-wait timer.
    - any other state: if the ACK is inside the expected window, the SND.UNA value is updated, and any packets that are now acknowledged are removed from the retransmission queue. Duplicate ACKs are ignored. If the state is FIN-WAIT-1 and our FIN is acked, we enter state FIN-WAIT-2. If the state is CLOSING and our FIN is acknowledged, enter the TIME-WAIT state, and start the time-wait timer.
  - If the packet contains data, and in the state ESTABLISHED, FIN-WAIT-1 or FIN-WAIT-2 the data is added to the processing queue. The RCV.NXT variable (next expected sequence number) is updated and an ACK packet is sent in reply. In any other state the data is unexpected, and ignored.

  - If the packet contains the FIN flag, and in a state other than CLOSED, LISTEN and SYN-SENT the RCV.NXT variable is updated, and an ACK packet is sent. Next, depending on the state, the following happens:
    * SYN-RECEIVED or ESTABLISHED: Enter CLOSE-WAIT state.
    * FIN-WAIT-1: If this acknowledges our FIN, enter TIME-WAIT state, and start the time-wait timer.
    * FIN-WAIT-2: Enter TIME-WAIT state, and start the time-wait timer.
    * TIME-WAIT: Restart the time-wait timer.
    * CLOSE-WAIT, CLOSING or LAST-ACK: Do nothing. Already closing connection.

1.4 TimeoutHandler

The TIMEOUTHANDLER class handles all timeouts that can occur inside the system. These timeouts can either be a user timeout, a retransmission timeout, or a time-wait timeout, handled in `onUserTimeout()`, `onRetransmissionTimeout()` and `onTimeWaitTimeout()` respectively.

- **onUserTimeout()** is called if a packet is not acknowledged within a specific timeframe. In that situation all queues are flushed, the user gets an error for any outstanding calls, and the CLOSED state is entered. Note that currently this method is not implemented.
- **onRetransmissionTimeout()** is called if the retransmission timeout of a segment in the retransmission queue expires. When that happens, this method checks that the maximum number of retransmits (10 in this implementation) have not been exceeded. If not, the segment is sent again, and its retransmission timer is restarted. Note that when a segment does receive an acknowledgement, the entry is removed from the retransmission queue. If the maximum number of retransmits is reached, three things can happen:
  - State is SYN-SENT: Three-way handshake was not completed, so move to CLOSED again.
  - State is SYN-RECEIVED: Three-way handshake was not completed, so move to LISTEN again.
  - Otherwise: state remains the same, packet is dropped.
- **onTimeWaitTimeout()** is called when the time-wait timeout expires. The time-wait timer is started when the TCP stack enters the TIME-WAIT state. This method sets the state to CLOSED.

2 Other classes

2.1 Segment

The SEGMENT class represents a segment to be sent over the network. It contains all information needed by the TCP protocol like the source- and destination port, the sequence- and acknowledgement number, the window size, control bits, data, etc. Besides storing this information, it also takes care of encoding and decoding packets.

2.2 TransmissionControlBlock

This class keeps track of all information needed by an active TCP connection. Examples are the socket information, the next sequence number to use when sending, the next sequence number that is expected from the other side, the last unacknowledged sequence number, the current state, etc.

This class also provides methods to block until a certain state is reached (`public void waitForStates(Transmi states)`), until a certain segment has been acknowledged (`public boolean waitForAck(Segment segment)`), or to wait until all packets that have been sent are acknowledged (`public void waitUntilAllAcknowled`

Other methods provide functionality to queue received data that has to be processed, and also to wait for this data to arrive (`public void waitForDataToProcess()`). Also, this class takes care of the retransmission queue, and provides methods to add or remove segments from this queue.

2.3 SegmentUtil

The SEGMENTUTIL provides four static methods to construct SYN-, SYN,ACK-, FIN-, or data-packets. Providing the sequence number and the acknowledgement number (where needed), the correct SEGMENT object is returned.

Also, this class contains methods used for sequence number arithmetic. It is important to notice that sequence numbers have a maximum value, and will wrap around at the point they reach this value. It is therefore impossible to compare two sequence numbers with normal arithmetic, to see if one was before the other. The following methods help to perform such checks in a safe manner:

- **public static boolean inWindow(long left, long seq, long right)**: This method does a wraparound-safe check to see if *seq* is inside the half open window [*left*, *right*). If *left* $\leq$ *right*, then it is assumed that the sequence numbers have wrapped around (i.e. from a high number, near the maximum, to a low, near 0).
- **public static boolean overlap(long left1, long right1, long left2, long right2)**: This method does a wraparound-safe check to see if the windows bounded by [*left1*, *right1*) and [*left2*, *right2*) overlap each other in at least one point.
- **public static boolean isAcked(Segment segment, long ack)**: This method checks if the given *segment* is completely acknowledged by the given *ack*.

2.4 ChecksumUtil

This class provides one public method `public static short calculateChecksum(ByteBuffer tcpPacketBuffer, IP.IpAddress srcAddr, IP.IpAddress destAddr, int tcpLength)`, which calculates the checksum for a given packet (already encoded in the *tcpPacketBuffer*).