Transmission Control Protocol

Claudio Cicconetti <c.cicconetti@iet.unipi.it>
Table of Contents

Transmission Control Protocol
The window principle
TCP segment format
Acknowledgment and retransmissions
Establishing a TCP connection
Application Programming Interface
TCP congestion control algorithms
TCP provides considerably more facilities for applications than UDP:

– error recovery
– flow control
– reliability.

TCP is a connection-oriented protocol.

Most of user applications use TCP.
Transmission Control Protocol

process 1

port m

TCP

IP

host A

------------

port n

TCP

IP

host B

TCP connection

reliable

unreliable

IP datagram
Stream Data Transfer

From the application's viewpoint, TCP transfers a contiguous stream of bytes through the network.

The application does not have to bother with chopping the data into basic blocks or datagrams.

TCP does this by grouping the bytes in TCP segments, which are passed to IP.
Transmission Control Protocol

TCP itself decides how to segment the data, which it forwards at its own convenience.

Sometimes, an application needs to be sure that all the data passed to TCP has actually been transmitted to the destination. In this case, the *push* function is used, which pushes all remaining TCP segments to the destination host.

When a connection is closed, data are automatically pushed to the destination.
Transmission Control Protocol

Reliability

TCP assigns a sequence number to each byte transmitted and expects a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted.

The receiving TCP uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments.
Flow Control

The receiving TCP, when sending an ACK back to the sender, also indicates the number of bytes that it can receive without causing overflow in its internal buffer.

Full Duplex

TCP allows concurrent data streams in both directions.
Transmission Control Protocol

Logical Connections

The reliability and flow control mechanisms require that TCP initializes and maintains certain status information for each data stream.

The combination of this status, including sockets, sequence numbers and window sizes, is called a logical connection. Each connection is uniquely identified by the pair of sockets used by the sending and receiving processes.
The window principle

A trivial transport protocol is:

– send a packet and then wait for an ACK from the receiver before sending the next packet;

– if the ACK is not received within a certain amount of time, retransmit the packet.

While this mechanism ensures reliability, it only uses a part of the available network bandwidth.
The window principle

utilization $\rho = \frac{t_{DATA}}{t_{DATA} + 2 \tau + t_{ACK}}$
The window principle

Now, consider a protocol where the sender groups its packets to be transmitted:

– the sender can send all packets within the window without receiving an ACK, but must start a timeout timer for each of them;

– the receiver must acknowledge each packet received, indicating the sequence number of the last well-received packet;

– the sender slides the window on each ACK received.
The window principle

\[
\text{utilization } \rho = \frac{N \cdot t_{\text{DATA}}}{N \cdot t_{\text{DATA}} + 2 \tau + t_{\text{ACK}}}
\]
The window principle

If packet 2 is lost, the receiver does not acknowledge the reception of subsequent data messages.

The sender re-transmits unacknowledged messages after a timeout expires.
The window principle is used in TCP, but:

- the window principle is used at the byte level, that is, the segments sent and ACKs received carry byte-sequence numbers and the window size is expressed as a number of bytes;
- the window size is determined by the receiver when the connection is established and is variable during the data transfer.
The window principle

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

- **1-8**:(bytes that have been transmitted and acknowledged)
- **9-11**:(bytes that have been transmitted but not acknowledged)
- **12-15**: (bytes that have been transmitted without waiting for acknowledgements)
- **16**: (bytes that cannot be transmitted)
TCP segment format

- src port
- dst port
- sequence number
- acknowledgment number
- window
- checksum
- urgent pointer
- options
- padding
- data
TCP segment format

**Source/destination port** is the 16-bit source/destination TCP port number.

**Sequence number** is the sequence number of the first data byte in this segment.

**Acknowledgment number** (used in ACK segments) contains the value of the next sequence number that the receiver is expecting to receive.

**Data offset** is the number of 32-bit words in the TCP header.
TCP segment format

**Window** (used in ACK segments) specifies the number of data bytes that the receiver is willing to accept.

**Checksum** is a 16-bit checksum of pseudo-header, TCP header, and TCP data.

**Urgent pointer** (with URG control bit set) points to the first data octet following the urgent data.
TCP segment format

Flags

- URG: the urgent pointer is significant.
- ACK: the acknowledgment is significant.
- PSH: push data to the receiver.
- RST: reset the connection.
- SYN: synchronizes sequence numbers.
- FIN: no more data from sender.
TCP segment format

Options

Just as in the case of IPv4, a number of options can be specified using a TLV encoding.

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Value…</th>
</tr>
</thead>
</table>

## TCP segment format

<table>
<thead>
<tr>
<th>Option</th>
<th>Length</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>End of option list</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>No-Operation</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Maximum Segment Size</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Window Scale</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Sack-Permitted</td>
</tr>
<tr>
<td>5</td>
<td>variable</td>
<td>Sack</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>Timestamps</td>
</tr>
</tbody>
</table>
TCP segment format

The **Maximum Segment Size** option is only used during the establishment of the connection (SYN control bit set) and is sent by the receiver to indicate the maximum segment length it can handle.

If this option is not used, any segment size is allowed.
TCP segment format

The **Window Scale** option is used to expand the definition of the TCP window to 32 bits. This option is determined while handshaking and there is no way to change it after the connection has been established.

It defines the 32-bit window size by using scale factor in the SYN segment over standard 16-bit window size. The receiver rebuild the 32-bit window size by using the 16-bit window size and scale factor.
Acknowledgment and retransmissions

TCP sends data in variable length segments.

Sequence numbers are based on a byte count.

Acknowledgments specify the sequence number of the next byte that the receiver expects to receive.
Acknowledgment and retransmissions

If a segment gets lost, the receiver will acknowledge all further well-received segments with an acknowledgment referring to the first byte of the missing packet.

The sender will stop transmitting when it has sent all the bytes in the window. Eventually, a timeout will occur and the missing segment will be retransmitted.
In order to set the timeout values to an appropriate value, TCP implements an algorithm that estimates the round trip time (RTT) of the segments.

To do this, TCP records the time at which a segment was sent, and the time at which the ACK is received. Then, it applies a weighted average:

\[
RTT(t+1) = \alpha \cdot RTT(t) + \beta \cdot \text{Sample}(t)
\]

Where \( \alpha + \beta = 1 \).
Establishing a TCP connection

Before any data can be transferred, a connection has to be established between the two processes.

One of the processes (usually the server) issues a passive OPEN call, the other an active OPEN call.

The passive OPEN call remains dormant until another process tries to connect to it by an active OPEN.
Establishing a TCP connection

Client

Active OPEN

Receive SYN+ACK
Connection open

Server

Passive OPEN
(socket listening)

Receive SYN

SYN, seq = y, ack = x+1

Receive ACK
Connection open

SYN, seq = x

ack = y+1
Establishing a TCP connection

This whole process is known as a three-way handshake.

Note that the exchanged TCP segments include the initial sequence numbers from both sides, to be used on subsequent data transfers.
Establishing a TCP connection

Open connections are closed by means of the segments with the FIN flag set.

Since the connection is full-duplex, the FIN segment only closes the data transfer in one direction.

The other process will now send the remaining data it still has to transmit and also ends with a TCP segment where the FIN bit is set.
Establishing a TCP connection

Host A

Receive FIN

Send n bytes

Close the connection

Socket status is deleted

Host B

Close the connection

B cannot send data to A

Receive DATA

Receive FIN

Socket status is deleted

time

FIN, seq = w

ack = w+1

DATA, seq = z

ack = z+n+1

FIN, seq = z+n+1

ack = z+n+2
Establishing a TCP connection
TCP congestion control algorithms

Unlike UDP, TCP implements congestion control, which prevents a sender from overrunning the capacity of the network.

TCP adapts the sender’s rate to the network capacity and attempt to either avoid or detect congestion situations.

Several congestion control enhancements have been added and suggested to TCP over the years.
TCP congestion control algorithms

One of the most common implementations of TCP is called *Reno*, and combines four different mechanisms:

- Slow start
- Congestion avoidance
- Fast retransmit
- Fast recovery
TCP congestion control algorithms

Slow start adds another window to the sender’s TCP: the congestion window (\(cwnd\)), in bytes.

The initial value of \(cwnd\) for new TCP connections is equal to the Maximum Segment Size (MSS) advertised.

\(cwnd\) is incremented of MSS each time an ACK is received.
TCP congestion control algorithms

The *effective window* is used by TCP to decide how many bytes can be on fly, i.e., unacknowledged.

The effective window is computed as the minimum between the congestion window (sender side) and the advertised window (receiver side).
TCP congestion control algorithms

cwnd (in units of MSSs)

1

2

3

4

5

6

7

8

Sender

Receiver

D1

A1

D2

D3

A2

A3

D4

D5

D6

D4

D5

D6

D7

D7
TCP congestion control algorithms

Slow start allows the $cwnd$ size of a TCP connection to grow exponentially in its initial phase.

The sender starts a timer at each TCP segment transmission, whose value is based on the estimated RTT.
TCP congestion control algorithms

If the acknowledgment of a segment is not received when the timer expires the segment is assumed to be lost, which is considered to be due to congestion.

In this case:

- the value sshthresh is set to half the current value of cwnd;
- cwnd is reset to its initial value;
- missing segments are retransmitted.
TCP congestion control algorithms

Sender

Receiver

cwnd (in units of MSSs)

D7 is buffered even though out-of-order
The value of \textit{sshthresh} is used to enable the congestion avoidance mechanism, as a substitute of the slow start.

When \textit{cwnd} becomes equal to \textit{sshthresh}, \textit{cwnd} is further incremented of one MSS at each RTT, thus yielding a linear increase of the congestion window.
TCP congestion control algorithms

Diagram showing the slow start and congestion avoidance phases with key points labeled as `timeout` and `cwnd`. The diagram also indicates `sshthresh` as a critical threshold.
TCP congestion control algorithms

Fast retransmit is a further optimization, which avoids having TCP wait for a timeout to resend lost segments.

In fact, TCP may generate an immediate acknowledgment when an out-of-order segment is received, called duplicate acknowledgment (DUP-ACK).

However, DUP-ACK are also caused by TCP segments being re-ordered by the network, in which case no retransmission should occur.
TCP congestion control algorithms

cwnd (in units of MSSs)

Sender

Receiver

duplicate acknowledgments are sent

timeout

7
8

1

2
TCP congestion control algorithms

With fast recovery, after fast retransmit sends what appears to be the missing segment, congestion avoidance is performed instead of slow start.

This is because, when three DUP-ACK segments are generated by the receiver, this means that three TCP segments have left the network.