Efficient data transmission with TCP

In this lesson, we'll study the main data transfer mechanisms used by TCP.

We'll cover the following

Segment Transmission Strategies

In a transport protocol such as TCP that offers a byte stream, a practical issue that was left as an implementation choice in RFC 793 was *when* a new TCP segment should be sent.

There are two simple and extreme implementation choices:

1. Send a TCP segment as soon as the application has requested the transmission of some data.
   - **Advantage**: This allows TCP to provide a **low delay service**.
   - **Disadvantage**: If the application is writing data one byte at a time, TCP would place each byte in a segment containing 20 bytes of the TCP header. This is a **huge overhead** that is not acceptable in wide area networks.
2. Transmit a new TCP segment once the application has produced MSS bytes of data. Recall MSS from this lesson on TCP Headers.
   - **Advantage: Reduced overhead**
   - **Disadvantage:** Potentially at the cost of a **very high delay**, which may be unacceptable for interactive applications.

Nagle's Algorithm

An elegant solution to this problem was proposed by John Nagle in RFC 896 called **Nagle’s Algorithm**.

In essence, as long as there are unacknowledged packets, Nagle’s algorithm keeps collecting application layer data up to maximum segment size, to be sent together in a single packet. This helps reduce the packetization overhead by reducing small packets.

Algorithm

```
if window size >= MSS and available data >= MSS:
    send one MSS-sized segment
else:
    if there is unacknowledged data:
        place data in buffer until acknowledgement is received
    else:
        send one TCP segment containing all buffered data
```

Nagle's Algorithm

The algorithm takes up 2-3 lines in a TCP implementation. Nagle’s algorithm ‘executes’ every time new data comes in from the remote host. Here’s how it works: it sends data if it is at least the size of one MSS and the window size is appropriate. Otherwise, it checks if any unacknowledged segments exist. If so, it **buffers the data** and doesn’t send it. There is no timer on this condition, and it will keep buffering data until previous segments are acknowledged. If an *ACK* segment comes in, it sends the data.

Limitations

Nagle’s has a few limitations:

1. Nagle’s algorithm is only supported by TCP and no other protocols like UDP.

2. TCP applications that require **low latency** and **fast response times** such as internet phone calls or real-time online video games, do not work well when Nagle’s is enabled. The delay caused by the algorithm triggers a noticeable lag. These applications usually *disable* Nagle’s with an interface called the `TCP_NODELAY` option.

3. The algorithm was originally developed at a time when computer networks supported much less bandwidth than they do today. It saved bandwidth and made a lot of sense at the time, however, the algorithm is much less frequently used today.

4. The algorithm also works poorly with **delayed ACKS**, a TCP feature that is used now. With both algorithms enabled, applications experience a consistent delay because Nagle's algorithm doesn't send data until an ACK is received and delayed ACKs feature doesn't send an ACK until after a certain delay.

Quick Quiz!

1

Consider the following scenario:

- The last segment received by a TCP entity had an ACK value of 10.
- The MSS value is 536.
- There are 50 bytes in the buffer waiting for transmission.
- The window size is 500.
- The application sends a 500 byte message.
- All the data sent so far has been acknowledged.

Will the data be buffered or sent on if Nagle's was active?

A) It will be buffered

B) One TCP segment with all the buffered data will be sent

C) One MSS sized segment will be sent

COMPLETED 0%

For now, that's all on Nagle's. Let's move on to TCP window-scaling mechanisms.