# Congestion Control

TCP congestion window and congestion policy are used to avoid network congestion. As the network is the main component in wireless communication, any congestion in a network must be avoided. If the data sent by the sender is not delivered by the network, it must inform the sender about it. The network, other than the receiver also helps in determining the sender’s window size.

## What is TCP Congestion Control?

Before understanding what is TCP congestion control, let’s first understand what you mean by congestion in the TCP network.

Congestion is an important factor in packet switched network. It refers to the state of a network where the message traffic becomes so heavy that the network response time slows down leading to the failure of the packet. It leads to packet loss.  
Due to this, it is necessary to control the congestion in the network, however, it cannot be avoided.

TCP congestion control refers to the mechanism that prevents congestion from happening or removes it after congestion takes place.  
When congestion takes place in the network, TCP handles it by reducing the size of the sender’s window. The window size of the sender is determined by the following two factors:

* Receiver window size
* Congestion window size

**Receiver Window Size**

It shows how much data can a receiver receive in bytes without giving any acknowledgment.

Things to remember for receiver window size:

1. The sender should not send data greater than that of the size of receiver window.
2. If the data sent is greater than that of the size of the receiver’s window, then it causes retransmission of TCP due to the dropping of TCP segment.
3. Hence sender should always send data that is less than or equal to the size of the receiver’s window.
4. TCP header is used for sending the window size of the receiver to the sender.

**Congestion Window**

It is the state of TCP that limits the amount of data to be sent by the sender into the network even before receiving the acknowledgment.

Following are the things to remember for the congestion window:

1. To calculate the size of the congestion window, different variants of TCP and methods are used.
2. Only the sender knows the congestion window and its size and it is not sent over the link or network.  
   The **formula** for determining the sender’s window size is:

Sender window size = Minimum (Receiver window size, Congestion window size)

**TCP Congestion Control**

* TCP congestion control is a method used by the TCP protocol to manage data flow over a network and prevent congestion.
* TCP uses a congestion window and congestion policy that avoids congestion.
* Previously, we assumed that only the receiver could dictate the sender’s window size. We ignored another entity here, the network.
* If the network cannot deliver the data as fast as it is created by the sender, it must tell the sender to slow down.
* In other words, in addition to the receiver, the network is a second entity that determines the size of the sender’s window.

## **Congestion Policy in TCP**

* **Slow Start Phase:**Starts slow increment is exponential to the threshold.
* **Congestion Avoidance Phase:**After reaching the threshold increment is by 1.
* **Congestion Detection Phase:**The sender goes back to the Slow start phase or the Congestion avoidance phase.

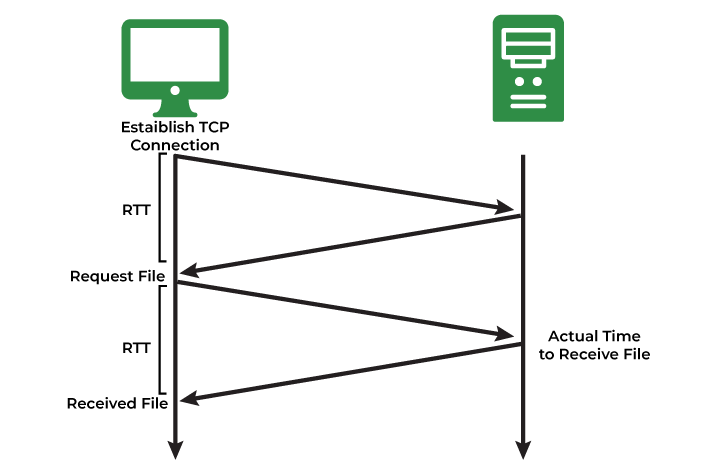
### 1 .**Slow Start Phase**

**Exponential Increment**: In this phase after every [RTT](https://www.geeksforgeeks.org/what-is-rttround-trip-time/)the congestion window size increments exponentially.

# What is RTT(Round Trip Time)?

RTT (Round Trip Time) also called round-trip delay is a crucial tool in determining the health of a network. It is the **time** **between a request for data and the display of that data**. It is the **duration measured in milliseconds.**

RTT can be analyzed and determined by pinging a certain address. It refers to the time taken by a network request to reach a destination and to revert back to the original source. In this scenario, the source is the computer and the destination is a system that captures the arriving signal and reverts it back.



**Example:** If the initial congestion window size is 1 segment, and the first segment is successfully acknowledged, the congestion window size becomes 2 segments. If the next transmission is also acknowledged, the congestion window size doubles to 4 segments. This exponential growth continues as long as all segments are successfully acknowledged.

### **2. Congestion Avoidance Phase**

**Additive Increment:** This phase starts after the threshold value also denoted as **ssthresh.** The size of CWND (Congestion Window) increases additive. After each RTT cwnd = cwnd + 1.

**For example:** if the congestion window size is 20 [segments](https://www.geeksforgeeks.org/difference-between-segments-packets-and-frames/)and all 20 segments are successfully acknowledged within an RTT, the congestion window size would be increased to 21 segments in the next RTT. If all 21 segments are again successfully acknowledged, the congestion window size will be increased to 22 segments, and so on.

### **3**. **Congestion Detection Phase**

**Multiplicative Decrement:** If congestion occurs, the congestion window size is decreased. The only way a sender can guess that congestion has happened is the need to retransmit a segment. Retransmission is needed to recover a missing packet that is assumed to have been dropped by a [router](https://www.geeksforgeeks.org/introduction-of-a-router/)due to congestion. Retransmission can occur in one of two cases: when the RTO timer times out or when three duplicate ACKs are received.

**Case 1:** Retransmission due to Timeout – In this case, the congestion possibility is high.

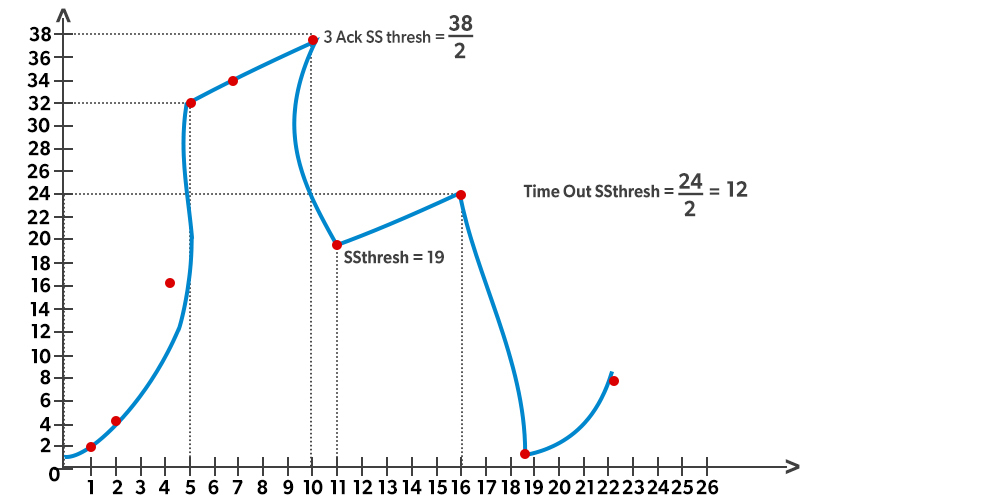
(a) ssthresh is reduced to half of the current window size.  
(b) set cwnd = 1  
(c) start with the slow start phase again.

**Case 2:**Retransmission due to 3 Acknowledgement Duplicates – The congestion possibility is less.

(a) ssthresh value reduces to half of the current window size.  
(b) set cwnd= ssthresh  
(c) start with congestion avoidance phase

#### Example

Assume a [TCP protocol](https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/)experiencing the behavior of slow start. At the 5th transmission round with a threshold (ssthresh) value of 32 goes into the congestion avoidance phase and continues till the 10th transmission. At the 10th transmission round, 3 duplicate ACKs are received by the receiver and entered into additive increase mode. Timeout occurs at the 16th transmission round. Plot the transmission round (time) vs congestion window size of TCP segments.



Congestion in TCP is handled by using these **three phases:**

1. Slow Start
2. Congestion Avoidance
3. Congestion Detection

**Slow Start Phase**

In the slow start phase, the sender sets congestion window size = maximum segment size (1 MSS) at the initial stage. The sender increases the size of the congestion window by 1 MSS after receiving the ACK (acknowledgment).  
The size of the congestion window increases exponentially in this phase.  
The **formula** for determining the size of the congestion window is  
Congestion window size = Congestion window size + Maximum segment size

| **Round trip time** | **Congestion window size** | **result** |
| --- | --- | --- |
| After a round trip of 1 | (2)1 | 2 MSS |
| After a round trip of 2 | (2)2 | 4 MSS |
| After a round trip of 3 | (2)3 | 8 MSS |

This is how you calculate the size of the congestion window and it goes on for n number of values.  
The general formula for determining the size of the congestion window is **(2)round trip time**

This phase continues until window size reaches its slow start threshold.

The **formula** for determining the threshold is given:

Threshold = Maximum number of TCP segments that the receiver window can accommodate / 2

= (Receiver window size / Maximum Segment Size) / 2

1. **Congestion Avoidance Phase**

In this phase, after the threshold is reached, the size of the congestion window is increased by the sender linearly in order to avoid congestion. Each time an acknowledgment is received, the sender increments the size of the congestion window by 1.

The **formula** for determining the size of the congestion window in this phase is  
Congestion window size = Congestion window size + 1 This phase continues until the size of the window becomes equal to that of the receiver window size.

1. **Congestion Detection Phase**

In this phase, the sender identifies the segment loss and gives acknowledgment depending on the type of loss detected.

**Case-01: Detection On Time Out**

1. In this, the timer time-out expires even before receiving acknowledgment for a segment.
2. It suggests a stronger possibility of congestion in a network
3. In this, there are chances that a segment has been dropped in the network

**Reaction in response to Detection on time out:**

* Setting the threshold to start at half of the current size of the window
* Decreasing the size of the congestion window to MSS
* Slow start phase is resumed

**Case-02: Detection Of Receiving 3 Duplicate Acknowledgements**

This case suggests the weaker possibility of congestion in the network. In this, the sender receives three duplicate acknowledgments for a network segment. The chances are that fewer segments have dropped while the one sent later might have reached.

**Reaction on receiving 3 duplicate acknowledgments:**

* Setting the threshold to start at half of the current size of the window
* Decreasing the size of the congestion window to that of the slow start threshold
* The congestion avoidance phase is resumed

TCP Timers-

Timers used by TCP to **avoid excessive delays during communication are called as TCP Timers.**

The 4 important timers used by a TCP implementation are-

**Time Out Timer**

**Time Wait Timer**

**Keep Alive Timer**

**Persistent Timer**

**Time Out Timer**- TCP uses a time out timer for retransmission of lost segments.

* Sender starts a time out timer after transmitting a TCP segment to the receiver.
* If sender receives an acknowledgement before the timer goes off, it stops the timer.
* If sender does not receives any acknowledgement and the timer goes off, then TCP Retransmission occurs.
* Sender retransmits the same segment and resets the timer.
* The value of time out timer is dynamic and changes with the amount of traffic in the network.
* Time out timer is also called as Retransmission Timer.

**Time Wait Timer**- TCP uses a time wait timer during connection termination.

* Sender starts the time wait timer after sending the ACK for the second FIN segment.
* It allows to resend the final acknowledgement if it gets lost.
* It prevents the just closed port from reopening again quickly to some other application.
* It ensures that all the segments heading towards the just closed port are discarded.
* The value of time wait timer is usually set to twice the lifetime of a TCP segment.

**Keep Alive Timer**- TCP uses a keep alive timer to prevent long idle TCP connections.

* Each time server hears from the client, it resets the keep alive timer to 2 hours.
* If server does not hear from the client for 2 hours, it sends 10 probe segments to the client.
* These probe segments are sent at a gap of 75 seconds.
* If server receives no response after sending 10 probe segments, it assumes that the client is down.
* Then, server terminates the connection automatically.

**Persistent Timer-**

TCP uses a persistent timer to deal with a zero-widow-size deadlock situation.

It keeps the window size information flowing even if the other end closes its receiver window.

# Leaky bucket algorithm

In computer networks, congestion occurs when data traffic exceeds the available bandwidth and leads to packet loss, delays, and reduced performance.

Traffic shaping can prevent and reduce congestion in a network.

It is a technique used to regulate data flow by controlling the rate at which packets are sent into the network.

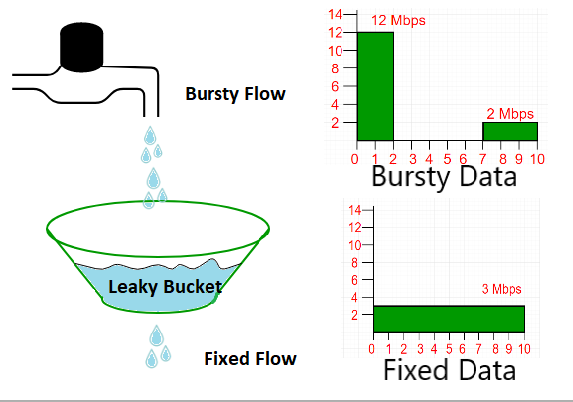
 There are 2 types of traffic shaping algorithms:

1. **Leaky Bucket**
2. **Token Bucket**

## Leaky bucket algorithm

Suppose we have a bucket in which we are pouring water at random points in time but we have to get water at a fixed rate to achieve this we will make a hole at the bottom of the bucket. This will ensure that the water coming out is at some fixed rate. If the bucket gets full, then we will stop pouring water into it.

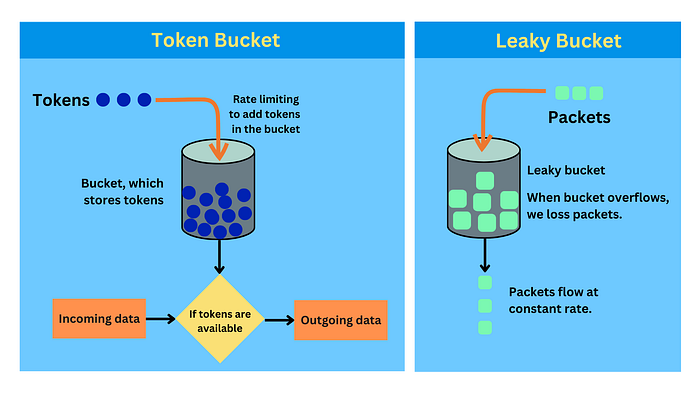
The **input rate can vary but the output rate remains constant**. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate.



In the above figure, we assume that the network has committed a **bandwidth of 3 Mbps** for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment.

In the above figure, the host sends a burst of data at a rate of 12 Mbps for 2s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in 10 s. The leaky bucket smooths out the traffic by sending out data at a rate of 3 Mbps during the same 10 s.

Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion.



# Token Bucket Algorithm

* Mechanism: The token bucket algorithm is based on tokens being added to a bucket at a fixed rate. Each token represents permission to send a certain amount of data. When a packet (data) needs to be sent, it can only be transmitted if there is a token available, which is then removed from the bucket.

Characteristics:

* Burst Allowance: Can handle bursty traffic because the bucket can store tokens, allowing for temporary bursts of data as long as there are tokens in the bucket.
* Flexibility: The rate of token addition and the size of the bucket can be adjusted to control the data rate.
* Example: Think of a video streaming service. The service allows data bursts for fast initial streaming (buffering) as long as tokens are available in the bucket. Once the tokens are used up, the streaming rate is limited to the rate of token replenishment.

Pros:

* Allows for flexibility in handling bursts of traffic.
* Useful for applications where occasional bursts are acceptable.

Cons:

* Requires monitoring the number of available tokens, which might add complexity.

# Leaky Bucket Algorithm

* Mechanism: In the leaky bucket algorithm, packets are added to a queue (bucket), and they are released at a steady, constant rate. If the bucket (buffer) is full, incoming packets are discarded or queued for later transmission.

Characteristics:

* Smooth Traffic: Ensures a steady, uniform output rate regardless of the input burstiness.
* Overflow: Can result in packet loss if the bucket overflows.
* Example: Imagine an ISP limiting internet speed. The ISP uses a leaky bucket to smooth out the internet traffic. Regardless of how bursty the incoming traffic is, the data flow to the user is at a consistent, predetermined rate. If the data comes in too fast and the bucket fills up, excess packets are dropped.

Pros:

* Simple to implement and understand.
* Ensures a steady, consistent flow of traffic.

Cons:

* Does not allow for much flexibility in handling traffic bursts.
* Can lead to packet loss if incoming rate exceeds the bucket’s capacity.

# Key Differences

* Traffic Burst Handling: Token bucket allows for bursts of data until the bucket’s tokens are exhausted, making it suitable for applications where such bursts are common. In contrast, the leaky bucket smooths out the data flow, releasing packets at a steady, constant rate.
* Use Cases: Token bucket is ideal for applications that require flexibility and can tolerate bursts, like video streaming. Leaky bucket is suited for scenarios where a steady, continuous data flow is required, like voice over IP (VoIP) or real-time streaming.