Computer Networks Report

Tcp –protocol

This protocol suites to LINUX operating system. INET is implemented using the Bsd socket interface as the means of communication with the user level. So in this protocol I explained about

Files like tcp ,tcp\_input and vegas.

Coming to first point tcp header is the one of the most important part in the protocol i.e tcp header.

Inorder to implement this header we require some functions ie.

**1)Struct tcphdr** and it is defined in tcp.h file.

This defines the complete architecture of the tcp header in a structure which forms the basis for all tcp implementations.

While establishing the connection it follows the 3-way hand shake algorithm and there will be in change of states as defined in the tcp state diagram. The state change is followed from tcp\_listen to tcp\_connection\_established.

2) **tcp\_rcv\_state\_process**

Used for changing the state of TCP\_LISTEN in the socket option to TCP\_CONNECTION\_ESTABLISHED

The tcp is followed through following states to establish a connection

i) TCP\_LISTEN(listening on some port)

ii) TCP\_SYN\_SENT(Syn flag sent)

iii)TCP\_SYN\_RECV(Syn+ack recieved)

iv)TCP\_CONNECTION\_ESTABLISHED

3) **tcp\_rcv\_synsent\_state\_process**

The flag bits in the tcpheader checks by this function and throws the state transition accordingly

If SEG.ACK =< ISS, or SEG.ACK > SND.NXT

then reset

else

send the acknowledgment

It also checks if the tcp\_header.RST flag is set then it has to reset the connection.

**4) tcp\_rcv\_fastopen\_synack**

If we want to connect the connection between them in fast way we can use this function and it is able to send data before the 3-way ack has been established based on the tcp-fastopen

Cryptographic token that authenticates that the client as an old client for the server. The token is saved on the client side and is used to lower the latency and skip a round trip time lowering cwnd size.

Here SYN and SYN-ACK are piggybacked with data to save some rtt.

The MSS advertised by the server is stored in the cache to determine

The maximum amount of data that can be supported in the SYN packet achieved by mss\_clamp by the user.

**5)tcp\_rcv\_rtt\_measure**

It contains the algorithm to calculate the rtt based on the difference between tcp timestamping and tcp rtt estimate.

**6)** **tcp\_ack\_update\_window**

This function defines the need to update window for window scaling in order to achieve flow control.

The ack is piggybacked with the update\_window and this function is called when the protocol thinks that it has to update the window

the functoion is followed from tcp\_ack\_may\_update\_window.

**7) static void tcp\_grow\_window**

The function is used to grow the congestion window size and the function just uses the principle that whenever there is more unacknowledged data on the network then increasing the window size may counter balance the rtt latency and hence is used to grow window size (follows the slow start threshold )

**9) tcp\_init\_buffer\_space**

This function is used to initialize the buffer space. It is called just after the TCP eneters TCP\_CONNECTION\_ESTABLISHED state.

**10) tcp\_init\_cwnd**

Used to initialize congestion window that needs to control the multiplicative increase and additive increase in congestion avoidance and hence to control the window factor for congestion.

**11) tcp\_init\_sock**

This function is used to initialize the socket to be used for the tcp\_sock

SOCK\_STREAM Provides sequenced, reliable, two-way, connection-based byte streams.

An out-of-band data transmission mechanism may be supported.

Family bits are used to tell the scope of communication.

AF\_UNIX, Local communication

AF\_INET IPv4 Internet protocols

AF\_INET6 IPv6 Internet protocols

**12) tcp\_mark\_urg**

This function is used to mark a segment if it is urgent or not. For this we need to set the URG pointer in the tcp header as tp->URG=1; and hence the packet is marked urgent

13)**tcp\_send\_mss**

This utility is used to send mss value whenever required. Especially required during caching mss in fastopen mode, fragmenting data over a router or when a router reclamps the mss.

14)**tcp\_sendmsg**

Tcp\_sendmsg is used to send message over a socket when a connection is well established. It is same function that we do on socket level to send a serialized data over a network.

**15) tcp\_recvmsg**

This routine copies from a sock struct into the user buffer and thus copies data into buffer. Only works after three way handshake or fastopen mode has been received.

16)**tcp\_getsockopt**

getsockopt() and setsockopt() manipulate options for the socket referred to by the file descriptor sockfd. Options may exist at multiple protocol levels; they are always present at the uppermost socket level.

17) **tcp\_cong\_avoid**

This function forms primary basis for the congestion-avoidance-algorithm using aspects such as **multiplicative decrease and additive increase** that form the most important aspect for controlling congestion on the internet.

**TCP VEGAS**

TCP Vegas is a TCP congestion avoidance algorithm that emphasizes packet delay, rather than packet loss, as a signal to help determine the rate at which to send packets. It was developed at the University of Arizona by Lawrence Brakmo and Larry L. Peterson.

**SUMMARY**

TCP Vegas adopts a more sophisticated bandwidth estimation scheme. It uses the difference between expected and actual flow rates to estimate the available bandwidth in the network. The idea is that when the network is not congested, the actual flow rate will be close to the expected flow rate. Otherwise, the actual flow rate will be smaller than the expected flow rate. TCP Vegas, using this difference in flow rates, estimates the congestion level in the network and updates the window size accordingly. This difference in the flow rates can be easily translated into the difference between the window size and the number of acknowledged packets during the round trip time, using the equation,

Diff = (Expected – Actual) BaseRTT,

Where Expected is the expected rate, Actual is the actual rate, and BaseRTT is the minimum round trip time. The details of the algorithm are as follow:

1. First, the sender computes the expected flow rate Expected = CWND/BaseRTT,

Where CWND is the current window size and BaseRTT is the minimum round trip time.

1. Second, the sender estimates the current flow rate by using the actual round trip time according to Actual = CWND/RTT, where RTT is the actual round trip time of a packet.
2. The sender, using the expected and actual flow rates, computes the estimated backlog in the queue from diff = (Expected-Actual) BaseRTT.
3. Based on diff, the sender updates its window size as follows:

CWND = CWND+1 if diff < α

CWND = CWND-1 if diff > β

CWND otherwise

TCP Vegas tries to keep at least α packets but no more than β packets in the queues. The reason behind this is that TCP Vegas attempts to detect and utilize the extra bandwidth whenever it becomes available without congesting the network. This mechanism is fundamentally different from that used by TCP Reno. TCP Reno always updates its window size to guarantee full utilization of available bandwidth, leading to constant packet losses, whereas TCP Vegas does not cause any oscillation in window size once it converges to an equilibrium point.

TCP implements RFC 3782

FUNCTIONS USED WITH DESCRITPTION:

1. **static void vegas\_enable**

This function enables the VEGAS algorithm.

1. **static inline void vegas\_disable**

This function disables the VEGAS.

If the connection is idle and we are restarting then we’ll not do any vegas calculations until we get any fresh RTT samples. So when we restart, we reset our Vegas state to a clean state. After we get acks for this flight of packets, then we can make Vegas calculations again.

1. **void tcp\_vegas\_cwnd\_event**(struct sock \*sk, enum tcp\_ca\_event event)

This function initialises the Vegas when we want to start or restart.

1. **static inline u32 tcp\_vegas\_ssthresh**(struct tcp\_sock \*tp)

This function returns the threshold value.

1. **static void tcp\_vegas\_cong\_avoid**(struct sock \*sk, u32 ack, u32 acked)

This function applies the main congestion avoidance algorithm of TCP Vegas.

If we do not apply Vegas, then we’ll implement TCP Reno.

If we are using Vegas,

we do the Vegas calculations only if we get enough RTT samples that we can reasonably sure that we got at least one RTT sample that wasn’t from delayed ACK.

If we had 2 samples total, which means we are getting only 1 ACK per RTT, which means they are almost delayed ACK’s.

If we have 3 samples we are OK.