International Journal of Computing and Applications | **ISSN: 0973-5704**
Volume 13, Number 2, (July-December 2018), pp 371-379

A DESIGN AND IMPLEMENTATION FRAMEWORK OF CONGESTION CONTROL ALGORITHM MODULE IN THE NETWORK SIMULATOR 3 (NS-3)

Girish Paliwal[1], Swapnesh Taterh[2] and N.S. Yadav[3]

[1,2] Amity University Rajasthan, Jaipur. Email: [1]gpaliwal@jpr.amity.edu, [2]staterh@jpr.amity.edu
[3]JECRC university Jaipur. Email: narensingyadav@yahoo.com

Abstract: The wireless sensor networking is very widely used in recent time because of its mobility. The traffic is increasing very fast so that traffic controlling protocols are required for both directions. The incoming and outgoing packets are controlling for reducing the congestion of mobile access point. One of the most important components of the Traffic Control is the congestion prevention and congestion control algorithms. If congestion occurs than the congestion control algorithms resolved it to smooth transmission of data. The congestion control algorithms role is to control the flow of packets transmitted by the Sanders it also resolves the congestion by dropping some packets and next packet to pass through the network interface. The Linux congestion Control algorithms are also performs scheduling of packets, congestion prevention, and congestion control. In this paper, we present the design and implementation congestion control algorithms as an additional module in ns-3 as well as existing module modification. We present the design and implementation algorithm framework of the class introduced to model a congestion control.

Keywords: NS-3, Network simulation, Congestion Control framework, Congestion control algorithms, TCP, wireless sensor network.

1. INTRODUCTION

Most of the devices in this world working on the Wi-Fi or wireless Technology because of it's wire free, not have limited geography reason, freely mobility accessing etc. but it also having some limitations one of the major limitations the wireless router devices having a Limited number of devices connectivity because of it's not having congestion control algorithm and its hardware limitation

The Traffic Control infrastructure of the Linux kernel enables to perform a number of actions on both outgoing packets before they are handed to the network devices for transmission, and incoming packets, before they are processed by the network layer protocols for the congestion control[1]. We focus on the transmission path taken by packets. Once the output interface and the next hop for an outgoing packet have been selected, the packet is en-queued into a queuing discipline (queue disc) in ns 3 number of the queue handling algorithm is implemented, which determines how the packet will be handled [2].

Today the majority of the researcher's field area is a simulation of different kind of protocols and finding out the better protocol those provide the better facility in the networking for the specific criteria the simulator play an important role in analyzing the number of metrics is and their values. ns3 can provide a number of congestion control algorithms those are already implemented and tested by numerous researchers weather at a different kind of networks scenario. moreover, simulation allows to easily reproducing measurements. This permits researchers to easily evolute and Debug protocols. Which type of technologies and algorithms are implemented, to compare them against other approaches. Finally, make a decision to apply an enhancement into existing protocols.

The ns3 network simulators tool used by the researchers all around the World to simulate different variety of the communication Technologies and network topologies, different kind of protocols and algorithm it also can be used in real time connected to the system does that provide a wide range of opportunity for the researchers to find out the actual problem behind the specific gap.

Till now ns3 has suffered TCP performance when we use to simulate a wireless network with large bandwidth delay product channel this is due to the absence of the TCP options and particular the window scaling timestamp options which one born in 1992 to improve TCP performance over such path. Another problem with the ns3 infrastructure is very less used for TCP congestion control algorithms to till date. The ns3 Simulator includes the implementation of different kind of congestion control algorithms as TCP new Reno and TCP Westwood, another handheld network simulator ns2 provide tested implementation of additional TCP variants, unfortunately, these were never put it into the ns3.because of less number of volunteer developer are freely share their code with ns3 developer team[3].

This paper related to the implementation Framework that presents in the implementation of Window scaling and timestamp option into the ns3. The TCP congestion control algorithm various properties of the protocol have been analyzed and reported by the researchers to validate. The ns3 having Congestion control implementation over different propagation delay throughput and error rates are changed [4] .

The senders Sands the packets at a specific transmission rate. If senders speed transmission is higher than the bottleneck bandwidth path than en-queue the packets. When a queue is requested to de-queue a packet depends on the implemented congestion control mechanisms. Basically, en-queuing a packet into a queue triggers a number of consecutive requests of de-queuing a packet. This process can be halted by the net device driver. The net device driver congestion control mechanism usually stops its transmission queue when it is full or its occupancy is above a given threshold. When the net device is able to receive packets again, the driver congestion control can start its transmission again. Additionally, the net device congestion control mechanism can wake a queue disc, i.e., request it to de-queue a packet, when its transmission queue is empty or its occupancy is below a given threshold. Currently, as it is lacking an equivalent of the Linux Traffic congestion Control infrastructure. No of congestion control mechanism is implemented and packets are only stored in the net device transmission queues. Consequently, cubic, Reno, Westwood congestion control algorithms can only manages the packets stored in the net device queues, which is not what happens in Linux. This paper presents the work done to introduce the ns3 equivalent of the Linux congestion Control algorithm infrastructure into ns-3. We believe that our work will allow researchers to carry out more realistic simulations and to evaluate congestion control algorithms more precisely. The remaining of this paper is organized as follows. In section 2 we provide an overview of the Linux congestion Control and of the current status of ns-3. Section 3 describes the model and design of the proposed congestion Control module for ns-3. Section 4 presents the congestion Control helper and some usage examples. Section 5 describes the experiments we performed with the new architecture and the results we obtained. In section 6 we conclude our work [2].

2. RELATED WORK

In this section, we first summarize the background of the NS3 network simulator implemented number of TCP congestion control algorithms and the extension of Window scaling and timestamps. the present is a survey of related work in this section.

Using the RFC 793 that was define a TCP window header field of 16 bit that suggests the largest represent able window is 65535 bytes, this limitation can be a problem over chain house with a high bandwidth delay product because it limits the exploitable bandwidth in such links.NFC 1323 introduced the window skin extension it expands the definition of the TCP window to 32 bit using a scaling factor to make this 32-bit value in the 16-bit window field this is connector is carried

out around in new TCP option Windows scale that is only set in sync segment[2].

The window scaling factor is 6 in each direction when the connection is opened another extension control is the timestamp option it is defined in mechanism that allows every segment including retransmission to Gill Limited at very low computational cost this way round trip time RTT measurement and retransmission Timeout RTO calculation can be really accurate feature of an essential for optimum TCP performance. Timestamp also have other uses as like protection from wrap around of sequence numbers. Many researchers have also improved the original congestion control algorithms of TCP those are implemented in the ns3. The key Idea behind any TCP congestion control algorithm to calculate RTT of a connection and the same instantaneous transmission rate of a reference TCP connection with the lower RTT, it is shown that this target can be achieved by modifying the time is here in order from that throughput to be independent of the RTT. This can be obtained through the use of an equation *p* equals to RTT / RTO retransmission time out [4].

Currently, network simulator ns-3 includes limitations for TCP functionalities. TCP or not supported and missed model for widely used congestion control algorithms. Therefore simulation can be an earthquake for today is standard and unable to represent what happened inside into the dense network of Gigabit Ethernet or high-speed satellite network channels. His paper present how can we implement ns3 TCP infrastructure for congestion control algorithm through the addition as well as various models of TCP congestion control algorithms those already implemented into the ns3. These are widely used an algorithm such as TCP cubic, TCP hybla, TCP Westwood TCP Reno TCP new Reno bic etc. [4] Design and implementation of ns3 algorithms and technologies have been developed over the years. Respect to the TCP and UDP researchers works. The most important thing is about to implementation and validation of the TCP CP congestion control protocols. The researchers provide different TCP performance and comparison with the existing TCP congestion control algorithm in ns3 like as Tahoe, Reno, and TCP new Reno. There is numerous parameters that affect the various networks correct characteristics such as error control rates bottleneck bandwidth propagation delay throughput are used as a performance metrics for the protocols. Ns3 also provides the direct code execution platform to check your congestion control algorithm in the real world implementation. The resources required design new congestion control algorithm and check it with existing congestion control algorithms and find out the result of the efficiency of the new algorithm corresponding to the existing algorithm that's why a new researcher required implementing and performing a comparative analysis after implementation of the new protocol. In this paper, we provide a model to implement congestion control algorithm [4]. Congestion control algorithm framework researchers also improve the original congestion control algorithm of TCP those are already implemented by various resources it is one possibility to make a good congestion control algorithm. It is required to know how can new researchers I will to implement the new code for improving the existing congestion control algorithm. When employing high speed congestion one possibility is to use TCP the main Idea behind using the specific congestion control algorithm to obtain along RTT connection to the same instantaneous transmission rate of a particular TCP connection with lower RTT. researchers using the analytical steps to show The Bourne that can be achieved by modifying the specific congestion control algorithm in order to get the better throughput and the network is less congested or congestion free. Many researchers used to calculate the slow start threshold and the congestion window to improve the congestion control protocol how to avoid congestion [4]. The design and the implementation of high speed and high capacity congestion control algorithm is a major task for the researchers. Most of the researchers used to control the congestion and the congestion window when the congestion window is growing faster than a specific Point the condition window gradually reduced due to slow down the transmission speed. Where some researchers are used to control the

congestion on the basis of AIMD additive increase and multiplicative decrease algorithm to control the round trip time. The round trip time is a time that consumer a packet to receive acknowledgment after sending. To implementing a high-speed TCP congestion control algorithm is required to periodically calculate cwnd and RTT. According to the cwnd and RTT, the data transmission speed is control by the sender after the occurrence of congestion [4].

Most of the cases that are seen by the researcher the condition occur in a network when the different routers having different transmission speed. in that case, the channel work according to the minimum transmission speed if any one of the channels transmits data with the high speed then the bottleneck link show as a congested path. The congestion control problem is handled with a starting point of the current window value as cwnd minimum and the maximum of the cwnd value represent as a target or as an event that indicates the buffer is full it is required to handle transmission rate or slow down the transmission to transmit the packets without any congestion. This is indicating the transmission of data without any loss or no loss data transmission this approach known as congestion free transmission. But it is very difficult to achieve the congestion free transmission because there are numerous factors those affect the channel transmission speed and data transmission rate due to these reasons it is not practically possible to achieve congestion free transmission. But many researchers develop a different kind of congestion control algorithm that provides the maximum throughput of the network for a specific network structure of network topology. It is also having some research gap that required increasing the channel speed channel bandwidth to get maximum throughput that indicating the transmission of the packet are very less drop and achieve the maximum number of acknowledgments. This kind of data transmission is possible with the effective and efficient congestion control algorithm that is sure the sender the data delivered guaranteed without failure [4].

In this section describe the model of the traffic congestion control module its design having the several challenges that we encountered during the implementation we are following steps to designing and implementation of the congestion control algorithm.

3. MODEL DESCRIPTIONS

In order to support and features are described in the network simulator 3. We introduce each and everything that required to design and implementation of a new congestion control algorithm. The main consequence is that it required flow control between the nodes. Each node and the network transmitting packets having its own capacity to transmissions of data and router routing the received packet to the destination before routing the packet to the destination it is required to check the destination of the packet. In this process, the router takes sometimes to process the destination address during this time the router receives the continue data packets and maintain a queue. This queue having the limitations like a number of packets holds without congestion. For each load, it is necessary to keep a status of the packets that can be passing to the next load during the transmission. If the node stops the passing of packets when the resource becomes available in this condition known as congestion and required to resolve the condition as soon as possible. A packet received by the TCP layer of transmission can we pass to a queue to perform scheduling and polishing [2].

4. TCP MODELS IN NS3

Under this heading, we discuss the TCP models available in ns3. First of all, we learn the TCP when we implement a congestion control algorithm because TCP is the integration of the congestion control algorithm

4.1. TCP Support by ns3

The ns3 support different TCP implementations as well as the congestion control algorithm implementation. It is inherited from the common header class src/ network directory. So the user can show about implementation with minimum changes into the driven classes. We describe the two important base classes as the following[5].

TCP socket class: In this class TCP socket hosting attributes is described. These attributes can be reused across different implementations, for example, attribute initial and can be used for any of the implementation that drives from class TCP socket.

TCP socket factory class: This class describe the protocol instance to create TCP socket of the right type this is working at the layer 4 of the network.

In network simulator (ns3) TCP model supports a hold bi-directional TCP with connection setup and close logic. Different congestion control algorithms are also supported by ns3 as TCP new Reno is the default, TCP Westwood, TCP Hybla, High speed, BIC and scalable. When finally release congestion control yet another high-speed TCP vs. HTTP and low extra delete background transport also support the model. It also supports selective acknowledgment, multipath TCP is not yet supported in the ns3 release.

Models are used many kinds of the TCP. It is set as the application layer. You're telling the ns3 application which kind of socket factory is used. Using the helper function we can define and create a TCP receiver. similarly, we configure on of application traffic to use TCP. To configuring TCP that have specific type ID of an abstract base class known as TCP socket factory how does the script of ns3 wants the native ns3 TCP versus some another one whether Internet steaks are added to the load default TCP implementation that is aggregated to the load is ns3 TCP this is overwritten by new TCP.

Configure the behavior of the TCP. The number of parameters is exported through the ns3 attribute system these are documented in the class TCP socket. For example, the maximum segment size table attribute to set the default socket type before any Internet that related object are created when we put the statement[5].

4.2. TCP Socket Interaction Interface with Application Layer

There is an analysis on the public interface of the TCP socket. How it can be used interact with the socket at the shell. The analyses of the callback function by the socket. This is also carried out for the sake of clarity. We will use the terminology sender and receiver to clearly divide the functionality. TCP is two rolls can be applied at a time the load does not lose general it because of the following definition can be applied to the both of sockets in case of full duplex mode TCP state machine is as following that is commonly used.

4.3. TCP State Machine

The TCP transition state machine in Figure 1 shows the TCP transition States. This figure was taken from the ns3 nsnam.org official website that shows the TCP transition States. How the TCP can work if the acknowledgment is not received then wait for the acknowledgment and synchronize with the sender and receive acknowledgments. The description of each TCP transition state is described in the Table 1.

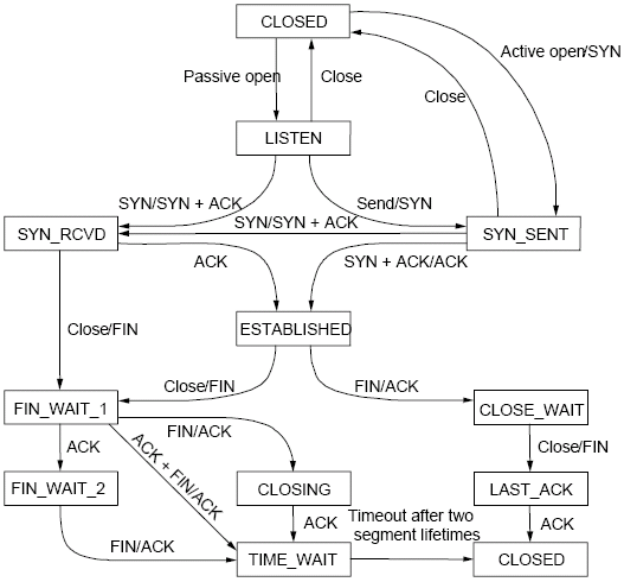

Figure 1: TCP transition state diagram

Table 1
Description of each TCP transition states

S. No.	*TCP States*	*Description of States*
1	CLOSED	The socket is finished.
2	LISTEN	Listening for a connection.
3	SYN_SENT	Send or sent a request for connection and wait for acknowledgment
4	SYN_RCVD	It is three-way handshaking received connection request send acknowledgment and wait for final acknowledgment

S. No.	TCP States	Description of States
5	ESTABLISHED	Establish a connection between sender and receiver
6	CLOSE_WAIT	Shutdown from far distance and wait for finish writing to close
7	LAST_ACK	After remote shut down the data in our before that have to finish sending
8	FIN_WAIT_1	After remote shutdown waiting is the complete transmission of buffer data.
9	FIN_WAIT_2	All data sent and wait for remote shutdown
10	CLOSING	When both sides shut down and having data for sending
11	TIME_WAIT	It is required because the other and may not have gotten our last acknowledgment then we transmit the data packet
12	LAST_STATE	It is used only in debugging the messages

4.4. Public Interface for Senders and Receivers

The public interface of the TCP is related to senders and receivers. The senders, as well as the receiver, are communicating to each other on the basis of TCP transition states. The TCP transition states are implemented are handled by some specific public interfaces those are implemented at sender and receiver side in ns3[5]. In this scope the table to describe the public accessible interfaces of the TCP base class.

Table 2
Senders and receivers public interface function and their description

S. No.	Public Receivers Interface	Description	Public Senders interface	Description
1	Bind()	To bind the socket with an address	Connect()	Try to connect with remote end point
2	Bind To Net Device()	Bind the socket with specified Net Device	Get Tx Available()	Return total amount of data stored in the TCP Tx buffer
3	Listen()	Listen from endpoint to incoming connection	Send()	Send the data for the TCP Tx buffer
4	Shutdown Send()	Terminate connection signal	Close()	Terminate connection, by sending a FIN msg
5	Get Rx Available()	Return total amount of data Recv or Recv From Socket.	Send To()	Same as *Send()*
6	Recv()	Collect data from TCP socket		
7	Recv From()	Collect data from TCP socket with source address		

4.5. Public Callbacks of TCP

These callbacks are called by the TCP socket to notify the application of interesting events. The given Table 3 shows public callbacks and their description as per the ns3.

Table 3
Public callbacks of TCP and their descriptions

S. No.	Public Callbacks	Description of Callbacks
1	Set Connect Callback	Notify Connection Succeeded first argument of callback, It is called in the SYN_SENT TCP state, before moving to the ESTABLISHED state.
2	Set Connect Callback	Notify Connection Failed second argument of callback, It is called after the SYN retransmission count goes to zero.
3	Set Close Callbacks	Notify Normal Close first argument of callback, A normal close is invoked.
4	Set Close Callbacks	Notify Error Close second argument of callback, Invoked when we send an RST segment (for whatever reason
5	Set Accept Callback	Notify Connection Request first argument of callback, Invoked in the LISTEN state of TCP when we receive an SYN.
6	Set Accept Callback	Notify New Connection Created second argument of callback, Invoked when from SYN_RCVD the socket passes to ESTABLISHED state.
7	Set Data Sent Callback	Notify Data Sent, Invoke the Socket notifies the application that some bytes have been transmitted on the IP level.

S. No.	Public Callbacks	Description of Callbacks
8	Set Send Callback	Notify-send, Invoked if there is some space in the tx buffer when entering the ESTABLISHED state
9	Set Recv Callback	Notify Data Recv, Called when in the receiver buffer there are in-order bytes

5. PROCESS TO IMPLEMENTING NEW CONGESTION CONTROL ALGORITHMS

In the First step to designing and writing a new congestion control algorithm from the scratch. New algorithm has a completely different process from changing of the existing TCP congestion control algorithm. It is starting of changing into TCP socket base class and TCP Congestion Ops.all the operation that you want to introduce into new congestion control algorithm are contained by the class TCP Congestion Ops. It is the similar implementation of TCP Congestion Ops of the Linux. It has the following congestion control operations are defined as follows[6]:

virtual std::string Get Name () const;

virtual uint32_t Get Ss Thresh (Ptr < const Tcp Socket State> tcb, uint32_t bytes In Flight);

virtual void Increase Window (Ptr < Tcp Socket State > tcb, uint32_t segments Acked);

virtual void Pkts Acked (Ptr < Tcp Socket State> tcb, uint32_t segments Acked, const Time& rtt);

virtual Ptr < Tcp Congestion Ops > Fork ();

In the TCP Congestion Ops most interesting method is to write or develop the Get Ss Thresh() and Increase Window(). These two functions are most important for congestion control in further the TCP socket base class decides the time to increase decrease congestion control window. Most of the information is provided to transmission control block. Here is also specify the method should be increased cwnd and SS thresh based on the number of segment acknowledged. When the TCP congestion control slow start then called the function Get Ss Thresh(). The congestion control algorithm is called when any packet is lost or it is about to lost. The packet acknowledgment (PktsAcked) is used when the congestion control algorithm having the information of RTT (round trip time) and it is required the time of ACK received periodically.

Next step of implementation required for checking the behavior of congestion control algorithm. It is required to check the behavior of the congestion control algorithm using the following tests. Those tests are already implemented and required to check your congestion control algorithm need some modification[7].

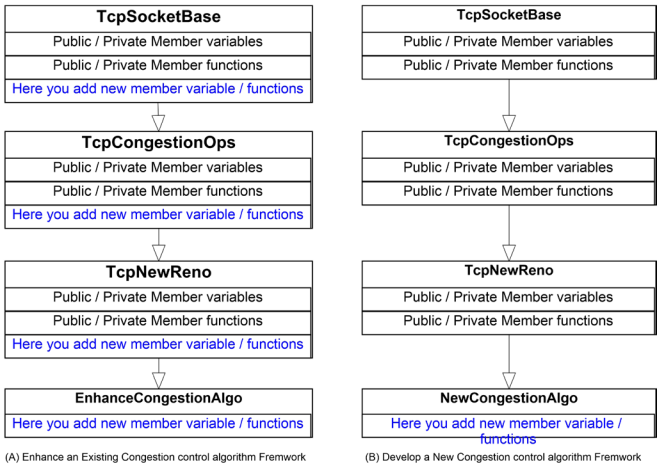

Figure 2: Framework for congestion control algorithm implementation and enhancement

- Check behavior of slow start using **tcp-slow-start-test**
- Check Unit test on the timestamp option by the **tcp-timestamp**
- Check Unit test on the window scaling option by the **tcp-wscaling**
- Check Unit test persist behavior for zero window conditions using **tcp-zero-window-test**

"Several tests have dependencies outside of the internet module, so they are located in a system test directory called src/test/ns3tcp".

- Check ns3 TCP congestion control algorithm works against liblinux2.6.26 using **ns3-tcp-cwnd**
- Check ns-3 TCP interoperates with liblinux2.6.26 by the **ns3-tcp-interoperability**

- Check the behavior of ns-3 TCP upon packet losses by the **ns3-tcp-loss**
- Check that ns-3 TCP Nagles algorithm works correctly and that it can be disabled using **ns3-tcp-no-delay**
- Check that ns-3 TCP successfully transfers an application data write of various sizes using **ns3-tcp-socket**
- Check the operation of the TCP state machine for several cases using **ns3-tcp-state**

6. INTEGRATION OF COMPONENTS INTO EXISTING CONGESTION CONTROL ALGORITHM OF NS3

First of all, we need to find out which variable integrated or Incorporated into existing congestion control algorithm in ns3. The ns3 having the number of congestion control algorithms you must find out suitable congestion control algorithm in which requirement of integration of the variable. By default, the packet Meta data is disabled in ns3 for the reasons performance. This approach work to increase the performance of ns3.we can easily explain the work packet metadata being enabled in ns3. The contents of the packet are stored in bytes buffer content the serialized version of the packets header footer as well as packets. Ns3 the content of buffer would be matched with a real world network packet data. I get meta data and what's the problem that providing information about the content of the byte buffer[8].

The implementation is not dependent on packet metadata that provided some knowledge of expected header structure within the packet being enqueued. In ns3 there is a number of queue management system only use as one output queue within a net device. All traffic coming into the queue has gone through the IP layer then you know where that queue resides. Packets and to the network either come from the application layer on the road that contains the queue or comes from another device went up to IP to be rooted[9].

Therefore the first order will be the link layer harder for the link layer protocol used by the net device follow by the IP header, however, the byte offset of the IP header depends on which link layer protocol is used. To handle this we have added an additional attribute in ns3. Ns3 also allow the user to specify the strategy used by the other to find out the IP header. You can apply the changes that we have made the diagram of classes to show in the figure 2. To enable a TCP socket the user set the required attribute in the TCP socket base it's true. As mentioned earlier you must be done for both and points of the TCP connection before used the protocol. The logic has been added to the connection setup phase [10].

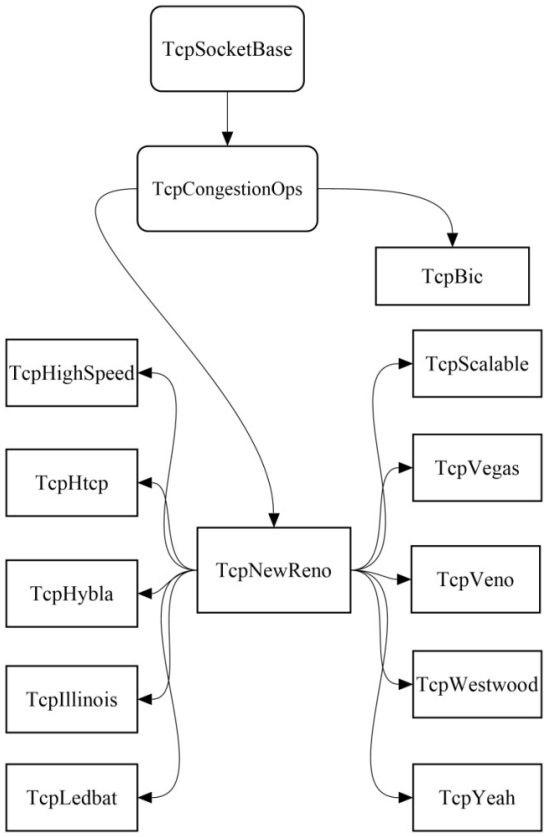

Figure 3: Hierarchical structure of congestion control algorithms

7. CONCLUSION

In this research paper, we try to elaborate how can you implement a new congestion control algorithm as well as enhance an existing congestion control algorithm. I would like to conclude my research related work on ns3 what is the framework of congestion control algorithm implementation. The conclusion of this work I find out a way to implement new congestion control algorithm inheriting socket base class or TCP congestion ops.

Underwriting and controlling the variable values that we include in our proposed congestion control algorithm. Further, we need to allies after simulation of the existing work as well as proposed work within a scenario. If you want to develop a new congestion control algorithm then you must follow the framework of congestion control algorithm. If you want to enhance the existing congestion control algorithm then you also follow this framework.

References

[1] G.K. Walia, O.P. Gupta, and S. Kumar, "Oriental Journal of Congestion Avoidance in Packet Networks Using Network Simulator-3 (NS-3)," 2016.

[2] P. Imputato, S. Avallone, N. Federico, and V. Claudio, "Design and Implementation of the Traffic Control Module in ns-3," pp. 1–8, 2016.

[3] P.S. Katkar, "Comparative Study of Network Simulator : NS2 and NS3," Vol. 6, No. 3, pp. 608–612, 2016.

[4] M. Casoni, C. A. Grazia, M. Klapez, and N. Patriciello, "Implementation and validation of TCP options and congestion control algorithms for ns-3," in *Proceedings of the 2015 Workshop on ns-3*, 2015, pp. 112–119.

[5] "NS3-TCP." [Online]. Available: https://www.nsnam.org/doxygen/group__tcp.html. [Accessed: 04-Jul-2017].

[6] "TCP-CongestionOps." [Online]. Available: https://www.nsnam.org/doxygen/group__congestion_ops.html. [Accessed: 04-Jul-2017].

[7] B. Levasseur, M. Claypool, and R. Kinicki, "A TCP CUBIC implementation in ns-3," in *Proceedings of the 2014 Workshop on ns-3*, 2014, p. 3.

[8] S. Gangadhar, T. A. N. Nguyen, G. Umapathi, and J.P.G. Sterbenz, "TCP Westwood (+) protocol implementation in ns-3," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, 2013, pp. 167–175.

[9] K. Nagori, M. Balachandran, A. Deepak, M.P. Tahiliani, and B. R. Chandavarkar, "Common TCP Evaluation Suite for ns-3 : Design , Implementation and Open Issues," pp. 9–16, 2017.

[10] B.P. Swenson and G.F. Riley, "Implementing Explicit Congestion Notification in ns-3."