Implementation of a Congestion Control Algorithm - TCP Adaptive Reno in NS-3

CSE322 : Computer Networking Sessionals

Offline 3

Objective

The objective of this assignment is to teach students how to incorporate a new model in NS-3 and learn how a congestion control algorithm works.

Overview of TCP Adaptive Reno

The widely used congestion control algorithm TCP-Reno / NewReno has some disadvantages. Its throughput decreases in networks with large bandwidth delay product and with non negligible packet loss. Other congestion control algorithms like TCP-Westwood / TCP-HighSpeed attempts to solve this problem but their potential unfriendliness to TCP-Reno is one of the reasons hampering their deployment.
In order to tackle this problem, a new congestion algorithm is proposed named TCP-AReno or Adaptive Reno [1]. This algorithm is based on TCP-Westwood-BBE. It has the following attributes:

- Estimates congestion level via RTT to determine whether a packet loss is due to congestion or not.
- Introduces a fast window expansion mechanism to quickly increase congestion window whenever it finds network underutilization. The congestion window increase will have 2 parts: The base part increases linearly in congestion avoidance phase just like TCP-Reno whereas the probe part increases exponentially to utilize the network fully.
- Adjusts congestion window reduction based on the congestion measurement. Congestion window is halved when the network is congested and a packet loss is likely to happen, while the reduction is mitigated when the network is underutilized.

Finally the paper[1] that implements this algorithm claims that the modifications result in:

- Higher throughput than TCP-Reno.
- Friendlier to TCP-Reno over a wide range of link capacities and random packet loss rates.

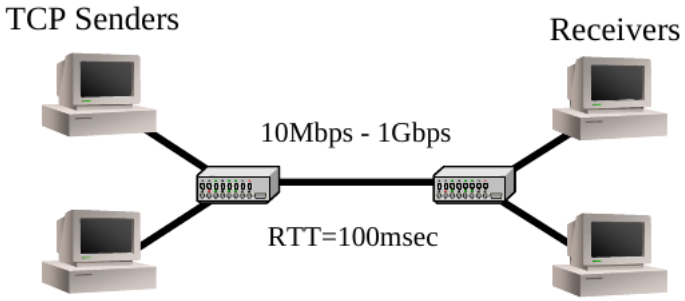

Fig. 4: Network topology (1)

Figure 1: Network Topology Presented in Paper

Task 1 - Network Topology

The first task is to design a similar dumbbell topology as proposed in the reference paper.

- There will be two leaf nodes as senders and receivers. You can use the class `PointToPointDumbbellHelper` to ease your implementation. However, remember to make a short change in the file `point-to-point-dumbbell.h`. Change its private variable `m_routerDevices` to public before using it.
- There will be only two flows in the network, each consisting of a single sender and a receiver. Each flow must use different congestion control algorithms. For example, if the flow 1 uses two nodes with TCP NewReno, then flow 2 will use the other two nodes with TCP WestwoodPlus. You can achieve this by installing different internet stacks in the nodes. For the bottleneck devices, you can install the corresponding internet stack with TCP NewReno.
- The `DataRate` and `Delay` of the senders and receivers will be **1Gbps** and **1ms** respectively. `Delay` of bottleneck link will be **100ms**.
- To set the router buffer capacity to the bandwidth delay product and to get the buffer employ drop-tail discarding as mentioned in the reference paper, add this line to your senders and receivers' P2P Helper:

  ```
  pointToPoint.SetQueue ("ns3::DropTailQueue", "MaxSize",
  StringValue (std::to_string (bandwidth_delay_product) + "p"));
  ```

  Here `pointToPoint` is your P2P Helper, and `bandwidth_delay_product` is the product of your bottleneck dataRate and delay divided by your packet size. You can set this value differently too.
- Add the `RateErrorModel` to the devices of the bottleneck link.
- Calculate the average throughput for all flows of each congestion control algorithm. You may use the library `FlowMonitorHelper`.
- You need to generate the following plots for both congestion control algorithm :
  1. Throughput Vs Bottleneck Data Rate
  2. Throughput VS Packet Loss Rate
  3. Congestion Window VS Time

Create a script that varies the bottleneck data rate from 1 to 300 Mbps while keeping the packet loss rate for the error model constant at 10^{-6}. Similarly, vary the packet loss rate from 10^{-2} to 10^{-6} while keeping the bottleneck data rate constant at 50 Mbps. An example of the graph is shown in Fig 2. The graphs should be generated when you run the script.

- You will need to use the trace source `CongestionWindow` for plotting. You can copy the contents of `tutorial-app.h` and `tutorial-app.cc` to your file. However, it would be great if you do find a solution that lets you import the tutorial-app class directly in the scratch folder.

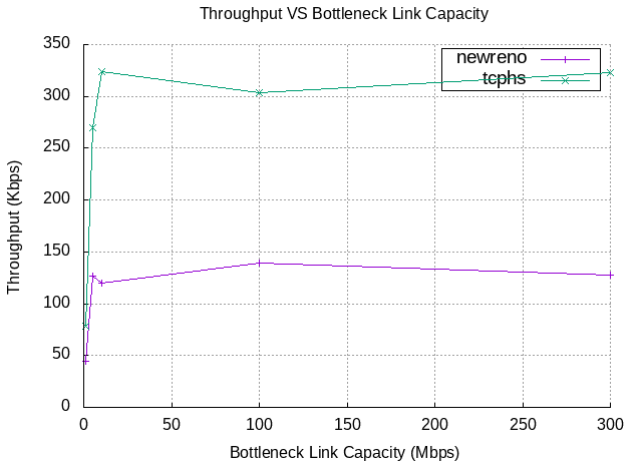

Figure 2: Throughput VS Bottleneck Link Capacity for TCPNewReno and TCP HighSpeed

Task 2 - Implementation of TCP Adaptive Reno

Include a model in NS-3

Create two files, `tcp-adaptive-reno.cc` and `tcp-adaptive-reno.h` inside `src\internet\model` folder. Include both filenames in `src\internet\CMakeLists.txt` like the following:

```
set(source_files
.....................
model/tcp-adaptive-reno.cc

set(header_files
.....................
model/tcp-adaptive-reno.h
```

Make the .h and .cc files have a similar structure like other built-in congestion control algorithms (TCP NewReno / WestwoodPlus / HighSpeed etc.). Finally run the command :

```
./ns3 build
```

If it builds successfully, then go to `examples\tutorial\fifth.cc`.
Configure the `ns3::TcpL4Protocol::SocketType` to the TypeId you gave to your model. Run the command:

```
./ns3 run fifth
```

If it runs successfully, then you have successfully included a new model inside the NS-3 codebase.

Modifications

The `TcpAdaptiveReno` class will inherit from the `TcpWestwoodPlus` class. Note, you need to change the access of the `EstimateBW()` to protected from private of `TCPWestwoodPlus` to access it from your model. You need to override the existing `PktsAcked`, `GetSsThresh` & `CongestionAvoidance` functions and implement two new ones. You need to declare necessary variables as well. The implementation details of each function is described below:

- `PktsAcked` : This function is called each time a packet is Acked. It will increase the count of acked segments and update the current estimated bandwidth.
- `EstimateCongestionLevel`: Estimates the current congestion level using Round Trip Time (RTT). First, you need to calculate the RTT_{cong}, the value RTT is estimated to take when a packet is lost due to congestion. To calculate the RTT_{cong} of jth packet loss, use the following equation:

$$RTT^{j}_{cong} = aRTT^{j-1}_{cong} + (1-a)RTT^{j}$$

  Here, assume $a = 0.85$ which is an exponential smoothing factor. Then estimate the congestion level c using the following equation:

$$c = \min(\frac{RTT - RTT_{min}}{RTT_{cong} - RTT_{min}}, 1)$$

- `EstimateIncWnd`: Calculate W^{max}_{inc} and update the value of $W_{inc}(c)$ using the following equations:

$$\begin{aligned} W^{max}_{inc} &= B/M * MSS \\ \alpha &= 10 \\ \beta &= 2W^{max}_{inc}(1/\alpha - (1/\alpha + 1)/e^{\alpha}) \\ \gamma &= 1 - 2W^{max}_{inc}(1/\alpha - (1/\alpha + 1/2)/e^{\alpha}) \\ W_{inc}(c) &= W^{max}_{inc}/e^{c\alpha} + c\beta + \gamma \end{aligned}$$

  Here, set M = 1000. B is the current bandwidth calculated in TcpWestWoodPlus's `EstimateBW()` function which is called in `PktsAcked()`. MSS is maximum segment size that is assumed as the square of the original SegmentSize. c is the estimated congestion level calculated in the previous function.

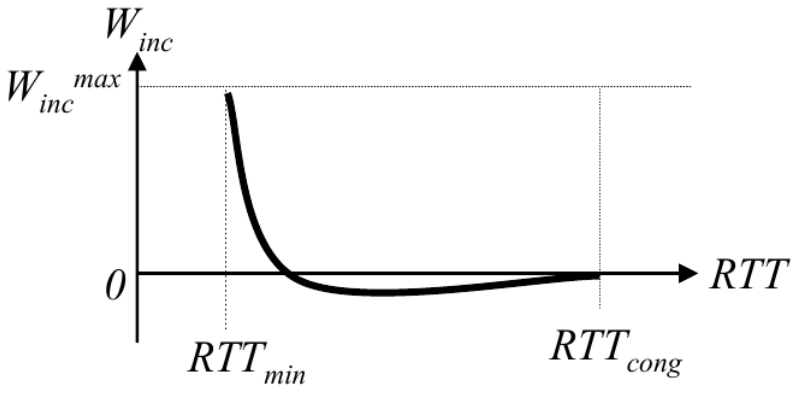

Figure 3: Window calculation in EstimateIncWnd

- `CongestionAvoidance` : This function increases congestion window in congestion avoidance phase. In TCP-AReno, congestion window W is managed in two parts; a base part, W_{base}, and a fast probing part, W_{probe}, as shown Fig. 4. The base part maintains a congestion window size equivalent to TCP-Reno and the probing part is introduced to quickly fill the bottleneck link. The base part is always increased like TCP-Reno, i.e. 1MSS/RTT, while the increase of the probing part, W_{inc}, is dynamically adjusted as illustrated in Fig. 4.
  When the network is congested, the value of W_{inc} gets close to zero, and the congestion window increases such as TCP-NewReno. Implement the following two equations in this function to calculate the new congestion window W.

$$W_{base} = W_{base} + 1MSS/W$$
$$W_{probe} = \max(W_{probe} + W_{inc}/W, 0)$$
$$W = W_{base} + W_{probe}$$

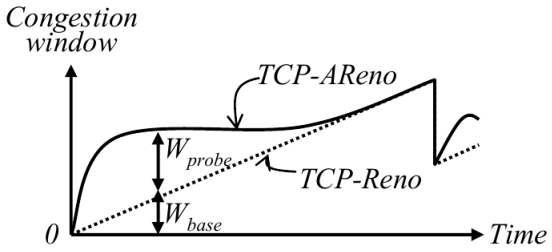

Figure 4: Window calculation in CongestionAvoidance

- `GetSsThresh` : This function is called when a loss event has occured. Firstly, here we need to update the RTT_{cong}^{j-1} & RTT^{j}. Then we need to estimate the new reduced congestion window size according to the congestion level. Like TCPW-BBE, the congestion window is halved when the network is congested and a packet loss is likely to happen; while the reduction is mitigated when the network is underutilized, as shown in Fig. 5.
  Calculate the following equations in this function:

$$W_{base} = W * W_{dec} = \frac{W}{1+c}; \quad W_{probe} = 0$$

  Here, c is the calculated congestion level. Update the ssthresh value to W_{base}. Note: the minimum value the ssthresh can take is 2*segmentSize similar to the other algorithms.

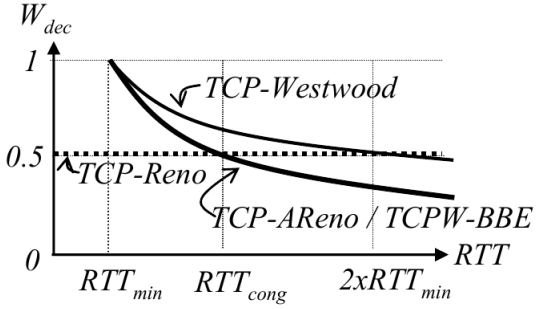

Figure 5: Window calculation in GetSsThresh

Performance Evaluation

Finally compare the performance of TCP HighSpeed and TCP Adaptive Reno alongside TCP NewReno by using them in task-1 and generating plots such as Figure 2.

Bonus

Calculate the Jain's Fairness Index for co-existing flows (NewReno + Adaptive Reno, NewReno + HighSpeed) and plot the following two graphs for each:

- Fairness Index vs Bottleneck Data Rate
- Fairness Index vs Packet Loss Rate

Reading Materials

To understand how the congestion control algorithm will work and in order to implement it, you need to understand the contents of the following models:

- `TcpSocketState`: mainly the enums: `TcpCongState_t, TcpCAEvent_t`
- `TcpCongestionOps`
- `TcpNewReno`
- `TcpWestwoodPlus`

Read the header files and the descriptions first then look at the functions implemented in the .cc files. For better understanding, enable the log component of each model one by one and run the `fifth.cc` to see the flow of functions called. **You must understand how the above models work before you start working on the implementation.**

Read the Section 2 of the reference paper [1] thoroughly. Identify the variables that are needed to be declared in your class and understand where they need to be updated exactly.

Submission Guidelines

You need to submit the following files in a folder named after your student id:

- tcp-adaptive-reno.cc
- tcp-adaptive-reno.h
- 1905***.cc file for simulation of the network.
- 1905***.sh file which will run the simulation and generate necessary plots.
- Any other files (.py, .plt etc) necessary for running the script successfully.

Zip the folder and submit it in Moodle. **Please note that usage of any unfair means will be duly punished and will result in a -100% mark.**

Submission Deadline : August 28, 2023 11:55 PM

References

[1] SHIMONISHI, H., AND MURASE, T. Improving efficiency-friendliness tradeoffs of tcp congestion control algorithm. *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.* (2005).