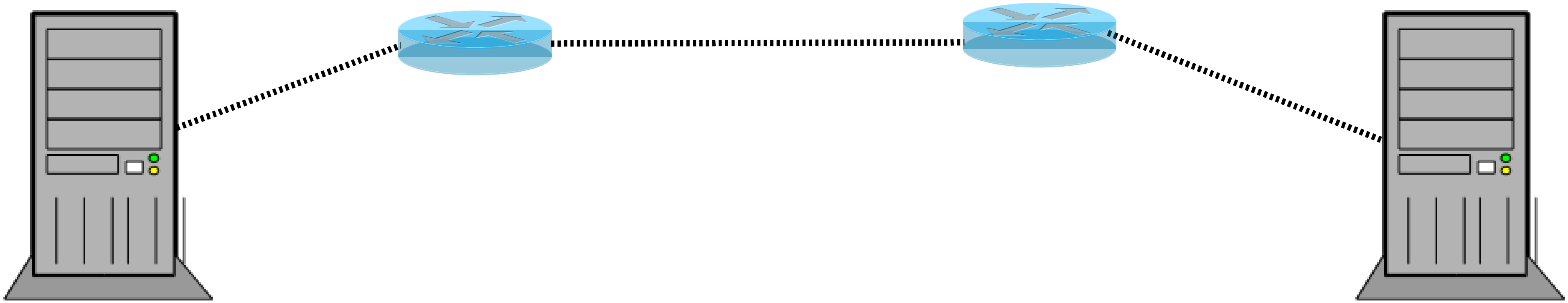
TCP Variants
NS3.25



STVN-2016
Jointly organized by
Poornima University, Jaipur & IIIT-Kota

Rahul Hada
hada.rahul@gmail.com

TCP model

- ns-3.25 support many TCP variants and more TCP variants have been added recently to appear in current development ns-3.26 release.
- **Model code**
  - **...src/internet/...**
- **Important abstract classes**
  - **`TcpSocket`** **`(...src/internet/model/tcp-socket.[cc,h])`**
    - To host TCP Socket attributes common to all implmentations as follow :-
      - **Send Buffer**
      - **Receive Buffer**
      - **Segment Size**
      - **Slow Start Threshold**
      - **Initial Congestion Window**
      - **and few more**
    - **`TcpSocketFactory(...src/internet/model/tcp-socket.[cc,h])`**
      - It is used by the layer-4 protocol instance to create TCP socket of the right type

NS-3 :TCP Variant

- ns3.25 version support following TCP Variants :-
  - **Tahoe**
  - **Reno**
  - **NewReno (default)**
  - **Westwood**
  - **Westwood+**
  - **Hybla**
  - **High Speed**
  - **few more**

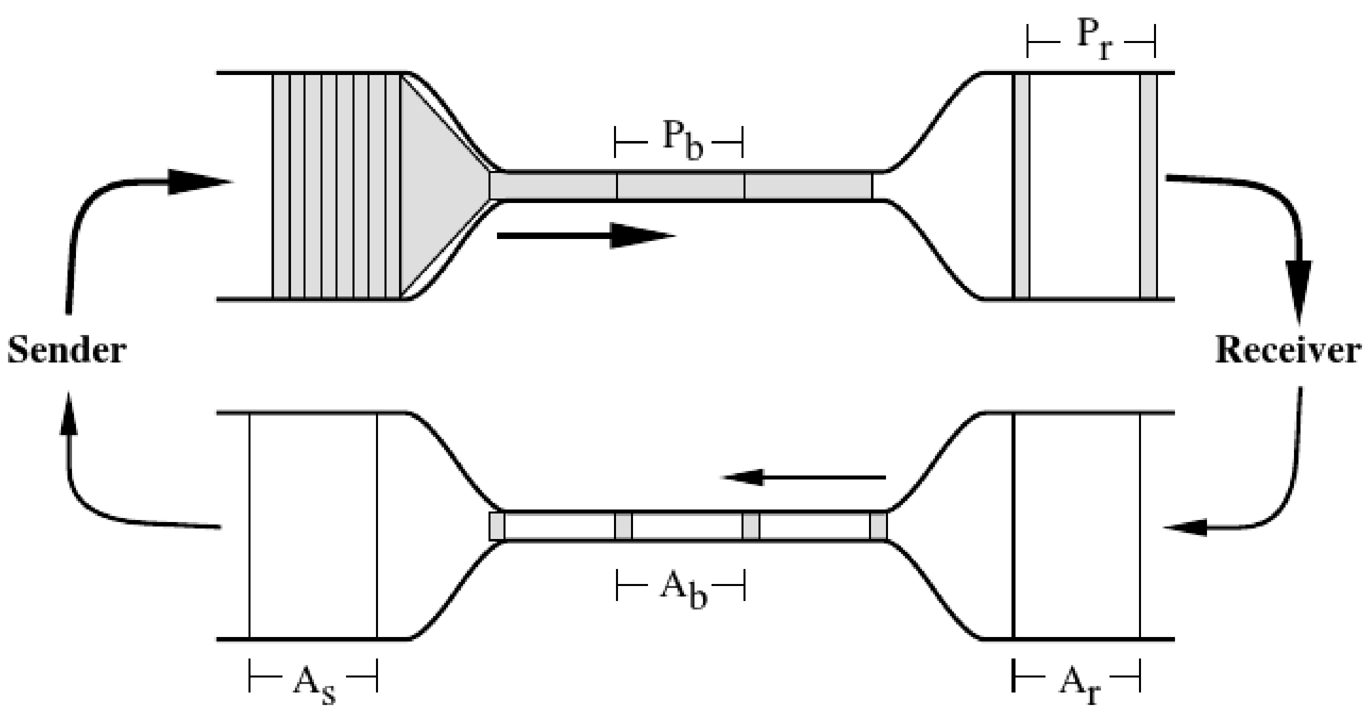
Why TCP Congestion Control ?

- Old TCP would start a connection with the sender injecting multiple segments into the network , up to the window size advertised by the receiver.[RFC-2001][1997]
- In Oct'86 , internet first face the congestion collapse*.
- The throughput from LBL to UC Berkeley (sites seperated by 400 yards and two IMP hops) dropped from 32 Kbps to 40 bps*.

** Congestion Avoidance and Control , Van Jacobson , Michael J. Karels Nov, 1998*

Where Congestion Occur ?

- When data arrives on big pipe(a fast LAN) and gets sent out a smaller pipe(a slow LAN).
- A router whose output capacity is less than the sum of multiple inputs.



* *Congestion Avoidance and Control , Van Jacobson , Michael J. Karels Nov, 1998*

Van Jacobson

Congestion Control Algorithms

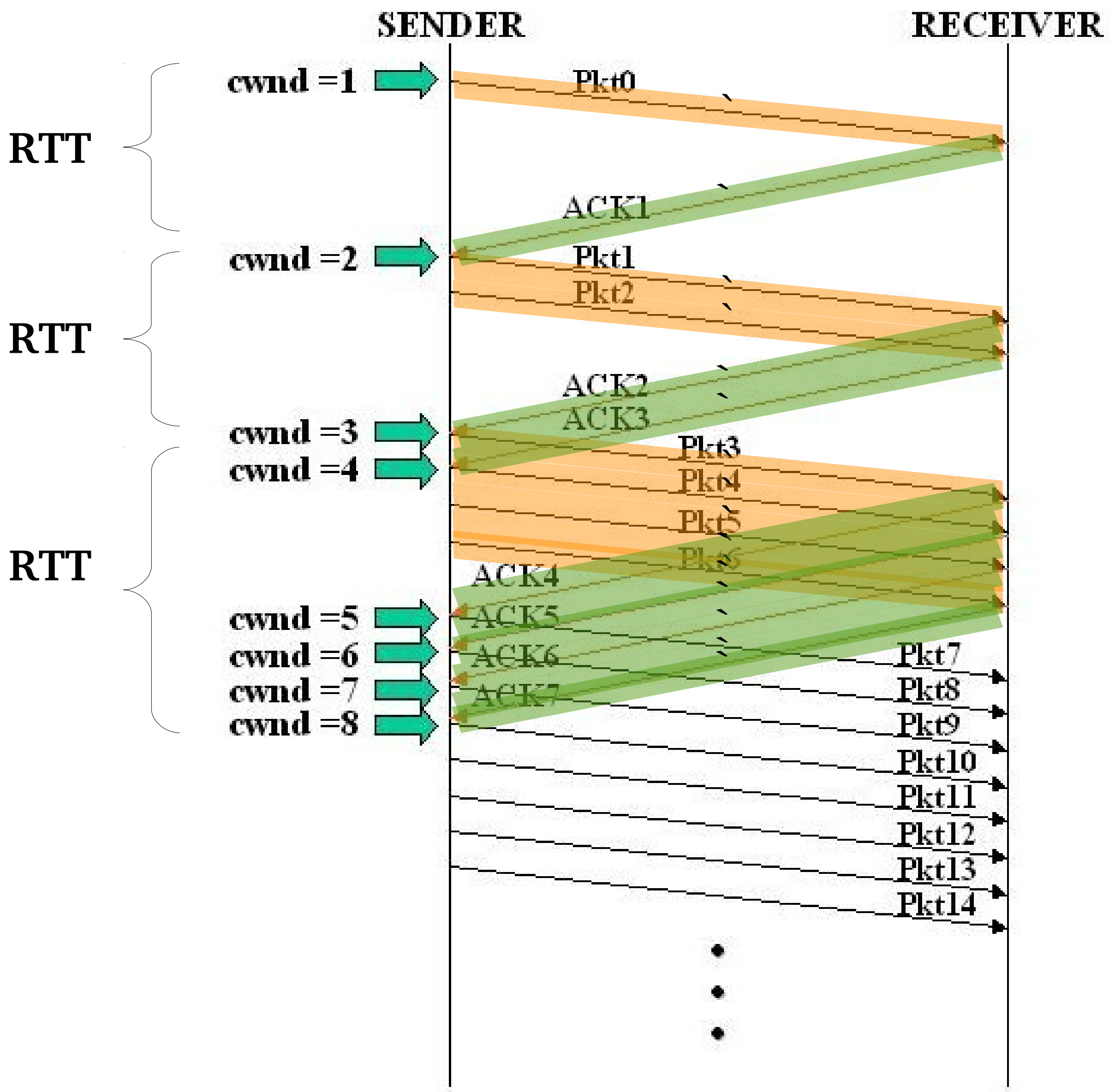
By Van Jacobson

- **Four Algorithms :-**
  - **Slow Start**
  - **Congestion Avoidance**
  - **Fast Retransmit**
  - **Fast Recovery**

Slow Start

- It introduce another window to the sender's TCP: ***the congestion window, called cwnd***
- Cwnd window initialized to one when new connection is establised.
- ***Increase cwnd by one segment when ACK reveived.***

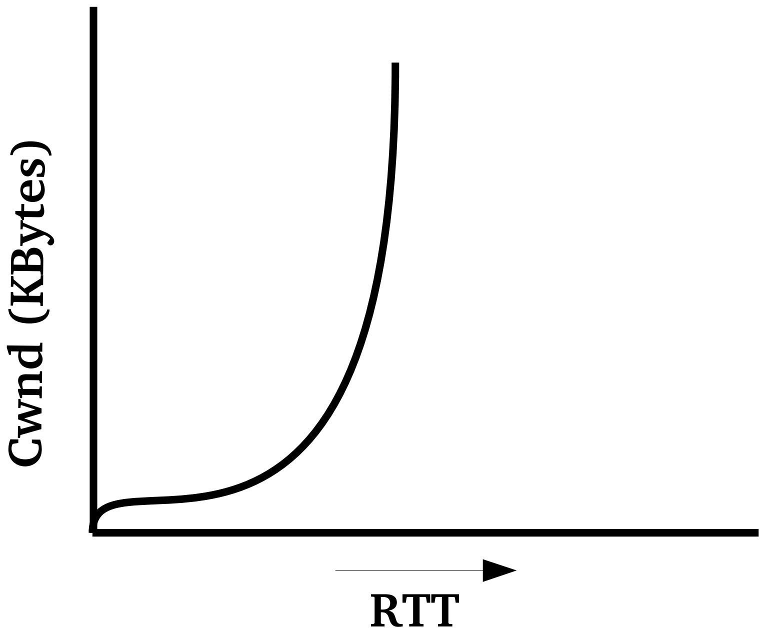
Slow Start



SENDER
RECEIVER
RTT
RTT
RTT
cwnd =1
cwnd =2
cwnd =3
cwnd =4
cwnd =5
cwnd =6
cwnd =7
cwnd =8
Pkt0
ACK1
Pkt1
Pkt2
ACK2
ACK3
Pkt3
Pkt4
Pkt5
Pkt6
ACK4
ACK5
ACK6
ACK7
Pkt7
Pkt8
Pkt9
Pkt10
Pkt11
Pkt12
Pkt13
Pkt14

Slow Start

- Double cwnd every RTT.



- Initial rate is slow but ramps up **exponentially fast**.
- Congestion window become **too large**
- At some point intermediate router will **start discarding packets.**

Slow Start/Congestion Avoidance

- Introduce another variable **ssthresh**
- **`ssthresh`** – threshold point of slow start(exponential growth)
- **FROM/TO Slow Start TO/FROM Congestion Avoidance**
  - `if cwnd < ssthresh --> slow start (SS)`
  - `if cwnd > ssthresh --> congestion avoidance (CA)`
  - `if cwnd == ssthresh --> SS OR CA`

ssthresh

- Initial value of ssthresh set arbitrarily high (size of largest possible advertised window)
  - **ssthresh = adv_window_size**
- When TCP sender ***detect segment loss*** using the retransmission timer , then
  - **ssthresh = cwnd/2**

cwnd < ssthresh
Slow Start

- The congestion window update using following formula
  - `cwnd = cwnd + SMSS`
- *Increment of 1 full size segment per ACK*

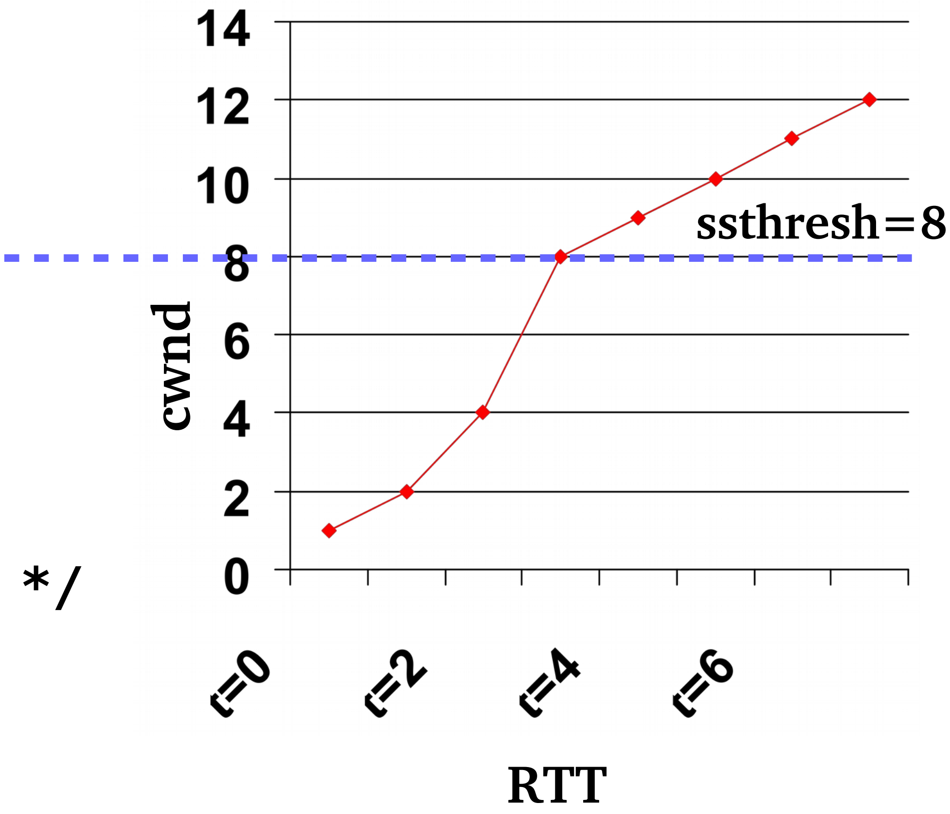
`cwnd > ssthresh` Congestion Avoidance

- The congestion window update using following formula
  - `cwnd = cwnd + 1/cwnd`
- *Approximation of increment of 1 full-size segment per RTT.*
- When TCP sender ***detect segment loss*** using the retransmission timer , then
  - `ssthresh = cwnd/2`
  - `cwnd = 1`

Slow Start/Congestion Avoidance

<u>TCP Pseudocode</u>

```
Initially:
   cwnd = 1;
   ssthresh = adv_wnd;
New ack received:
   if (cwnd < ssthresh)
         /* Slow Start*/
         cwnd = cwnd + 1;
   else
         /* Congestion Avoidance */
         cwnd = cwnd + 1/cwnd;
Timeout:
   /* Multiplicative decrease */
   ssthresh = cwnd/2;
   cwnd = 1;
```



Name of TCP Variant

OldTahoe

Problem:TCP OldTahoe

- **In OldTahoe**
  - If segment is lost , their is long wait until RTO(Retransmission Time)
- **New Version Introduce(Fast Retransmit)**
  - Retransmit after 3(three) duplicate ACKs
- **Now Indications of Congestion in Network**
  - Retransmission Timeout
  - 3(Three) duplicate ACKs

Fast Retransmit

- When 3 (third) duplicate ACK is received , set **`ssthresh = max(FlightSize/2,2*SMSS)`**
- Retransmit the lost segment and set **`(cwnd=1)`**
- Increment the congestion window(cwnd) by 1

Name of the TCP Variant

Tahoe

Problem:TCP Tahoe

- Indication of Congestion
  - **3 (Three) Duplicate ACK**
    - Moderate Congestion
  - **Retransmission Timeout**
    - Heavy Congestion
- In Tahoe
  - cwnd drop to 1 under heavy & moderate congestion
  - In moderate congestion no need to drop cwnd so drastically.
- New Version Introduce (**Fast Recovery**)
  - It is used to handle Moderate Congestions

Fast Recovery

- Algorithm
  - Step-1 `Set (ssthresh = cwnd/2)`
  - Step-2 `Set (cwnd = ssthresh + 3*SMSS)`
  - Step-3 `Loop: Check Receive ACK`
    - Step-4 `if Receive ACK == Duplicate ACK`
      - Step-5 `Set cwnd = cwnd+1*SMSS`
    - Step-6 `Goto Step-3`
    - Step-7 `if Receive ACK == Higher ACK`
      - Step-8 `Set cwnd = ssthresh`
      - Step-9 `Follow congestion avoidance`

Change in cwnd

- **Retransmission Timeout (--> Slow Start)**
  - SET cwnd=1
- **3(three) duplicate ACKs (--> Fast Recovery)**
  - SET cwnd=ssthresh/2

Name of TCP Variant

Reno

Problem:Reno

- Multiple Loss in the same window then Reno enters Fast Recovery multiple times thus decreases cwnd by half every time.
- Multiple Packet Loss is common thus Reno doesn't increase the throughput significantly.
- New Version Introduce (Modified Fast Recovery)
  - To provide significant better throughput in case of multiple packet loss in the same window

Modified Fast Recovery

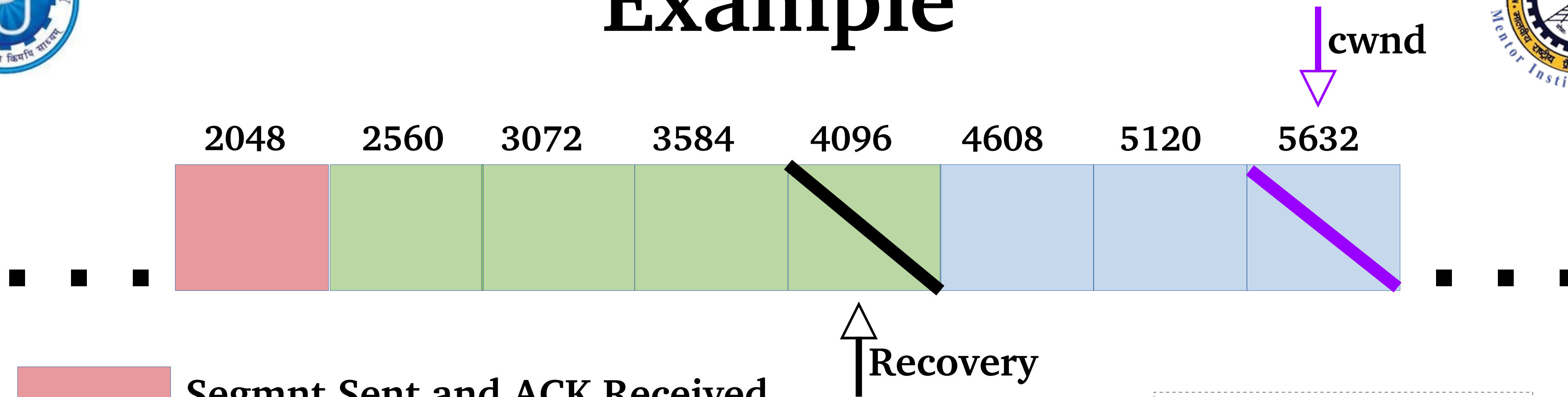
- Introduce two new terms
  - **Recovery** – sequence number of the highest data packet which is sent when the third duplicate ACK arrives
  - **Partial ACK** – a new ACK arrives which has an ACK number lower than the recover packet

Modified Fast Recovery

- Algorithm
  - **Step-1** Set **recovery** to highest data packet which is sent when the third duplicate ACK arrives
  - **Step-2** Loop : Fast Recovery
    - **Step-3** If ACK arrive having **ACK number < recovery**
      - **Step-4** Then **Partial ACK**
      - **Step-5** Change cwnd (**cwnd = cwnd-newdata+SMSS)**

        newdata --> the amount of data that has been ACK after the retransmitted packet
      - **Step-6** If receive partial ACK goto Step-2 else goto Step-7
- Step-7 Exit from Fast Recovery

Example



Segmnt Sent and ACK Received

Segment Sent and No ACK received

Segment NOT Sent

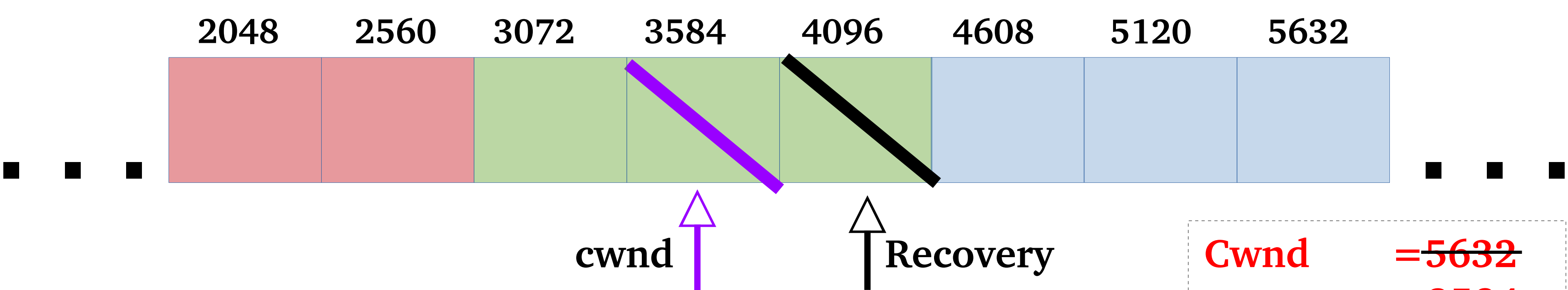
Cwnd =5632
Recovery = 4096

T2

Partial ACK

ACK arrives : ACK number <Recovery

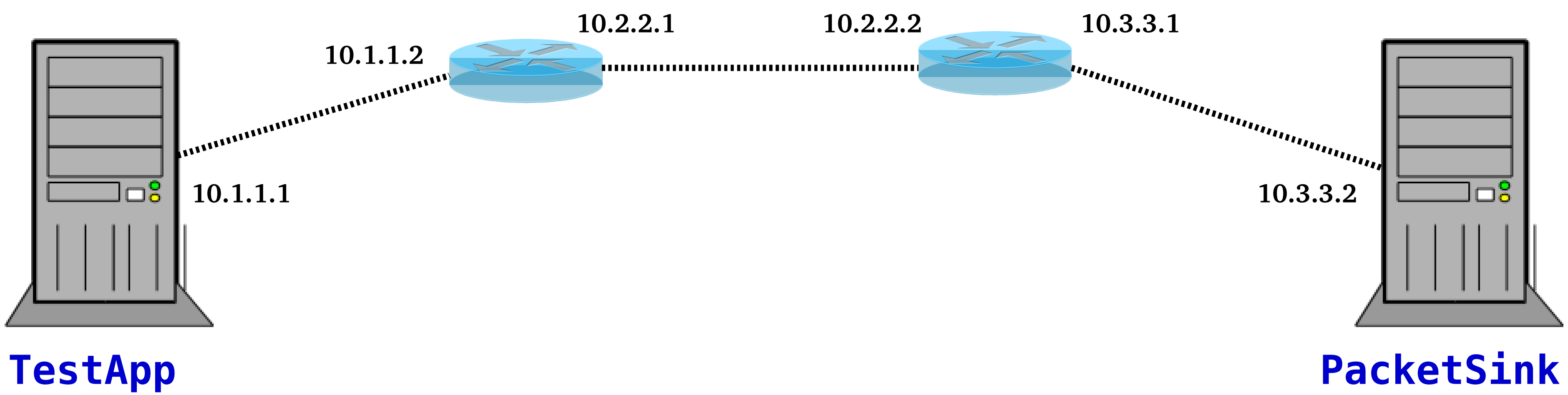
(3072<4096)



Cwnd = ~~5632~~
3584
Recovery = 4096

Now ReBuild a Scenario

Scenario 2 ii)



How to Create ??

- Set the NS-3 Application layer by telling the kind of socket factory to use
- Create ApplicationHelper :-
  - `PacketSinkHelper`
    - It is used to receive traffic send by the user/ns3 applications
  - `OnOffHelper`
    - It is used to send traffic with some random On/OFF times
  - `BulkSendHelper`
    - It is used to send bulk data typically used UDP socket
  - etc.

Modify Scenario 2 i) TestApp

- Create **TestApp** with following attributes:-
  - **PacketSize** – size of application packet
  - **DataRate** – rate of data generated by the application
  - **Address (IP + Port)** – address of application bind's
  - **Number of Packets** – number of packet
  - **Socket** – type of socket
  - **Event Id** – event detail to schedule by Simulator

Config::SetDefault()

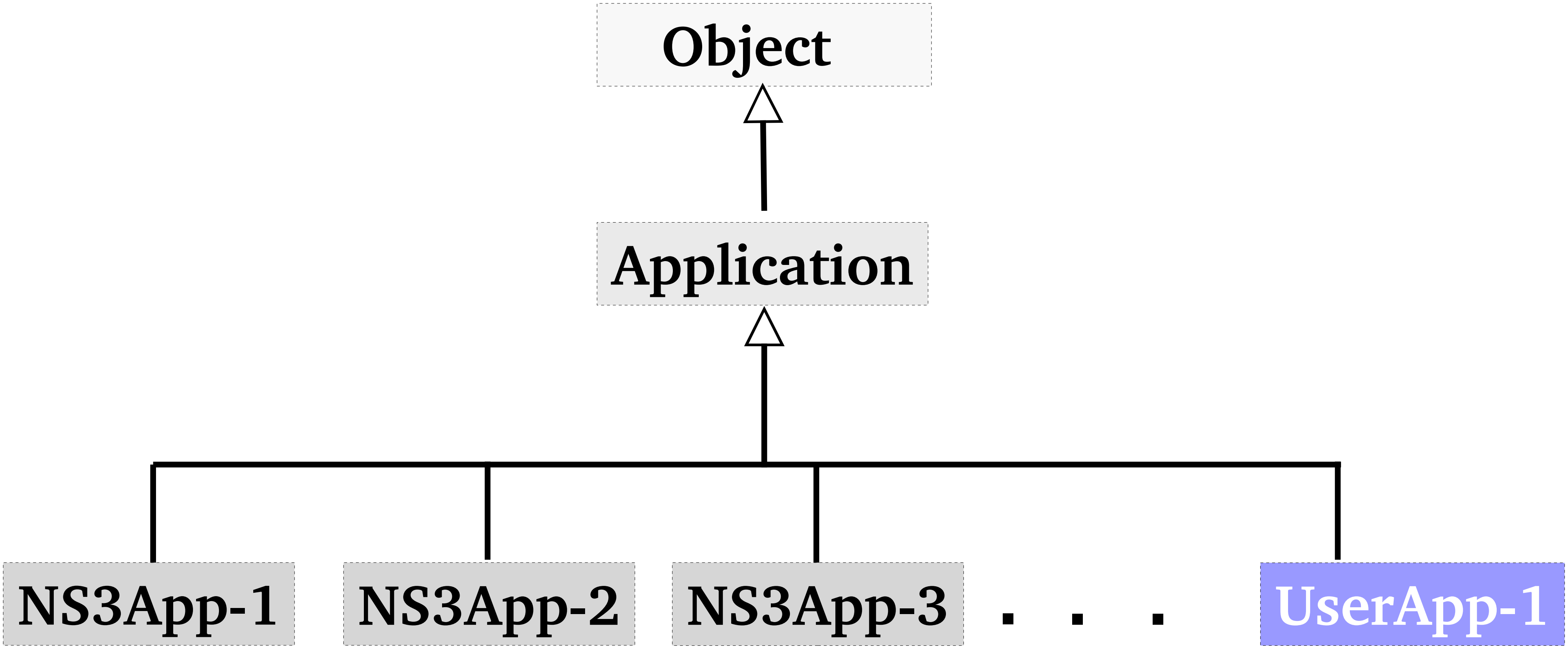
- To set the default socket type before any internet stack related objects are created

  Config::SetDefault(full_name,value)

- **Example**

```
Config::SetDefault ("ns3::TcpL4Protocol::SocketType", StringValue
("ns3::TcpTahoe"));
```

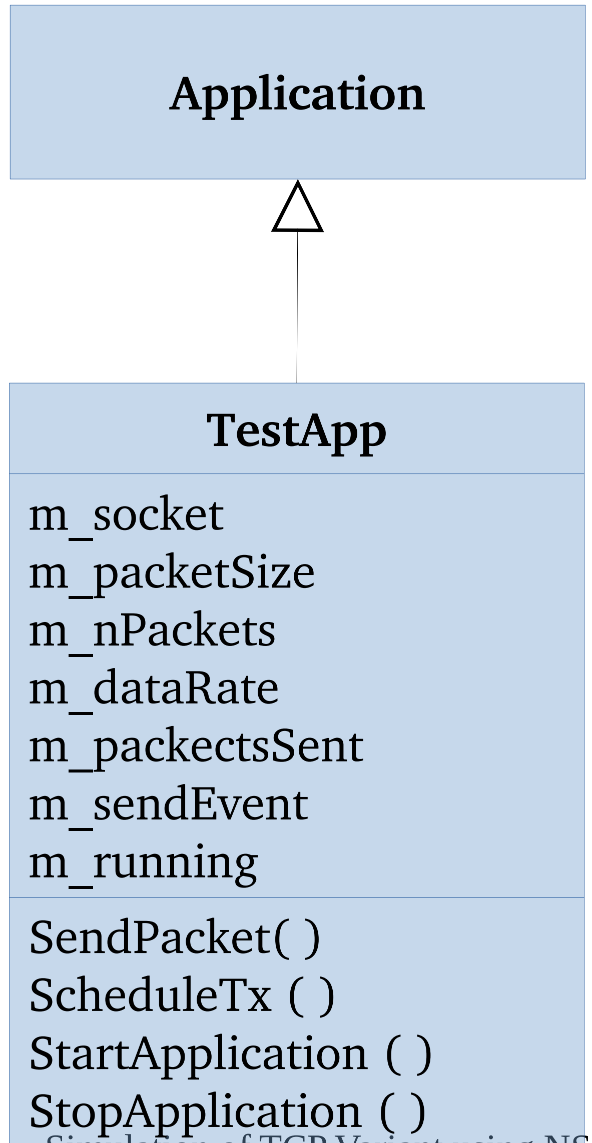
Inherit Application



Application

- It is base/abstrace class of all ns3 Applications
- It is derived from ns3 Object class
- It contains two virtual functions :-
  - `void StartApplication(void)`
  - `void StopApplication(void)`

TestApp:Class Diagram



DataMembers

- Data Members of **`TestApp`** Class
  - **`m_socket`** – Socket object
  - **`m_PacketSize`** – size of the application packet
  - **m_nPackets** – number of packets send by application
  - **`m_dataRate`** – rate of data send by application
  - **`m_sendEvent`** – store schedule/duration of the event
  - **`m_running`** – flag variable to store status

Member Functions

- Member Function of **`TestApp`** Class
  - `void Setup( Ptr<Socket>,Address , uint32_t , uint32_t , DataRate)`
    - To initialize the socket , address , packetsize , number_of_packets and data rate for the application
  - `void SendPacket(void)`
    - Create packet
    - Send packet using socket
  - `void ScheduleTx(void)`
    - To set simulation time to schedule an event
  - `void StartApplication(void)`
    - Bind and connect the socket
    - Change the status of the application
  - `void StopApplication(void)`
    - Cancel the simulator
    - Close the socket

ApplicationDevelopment
Using
Sample Code

FlowMonitor

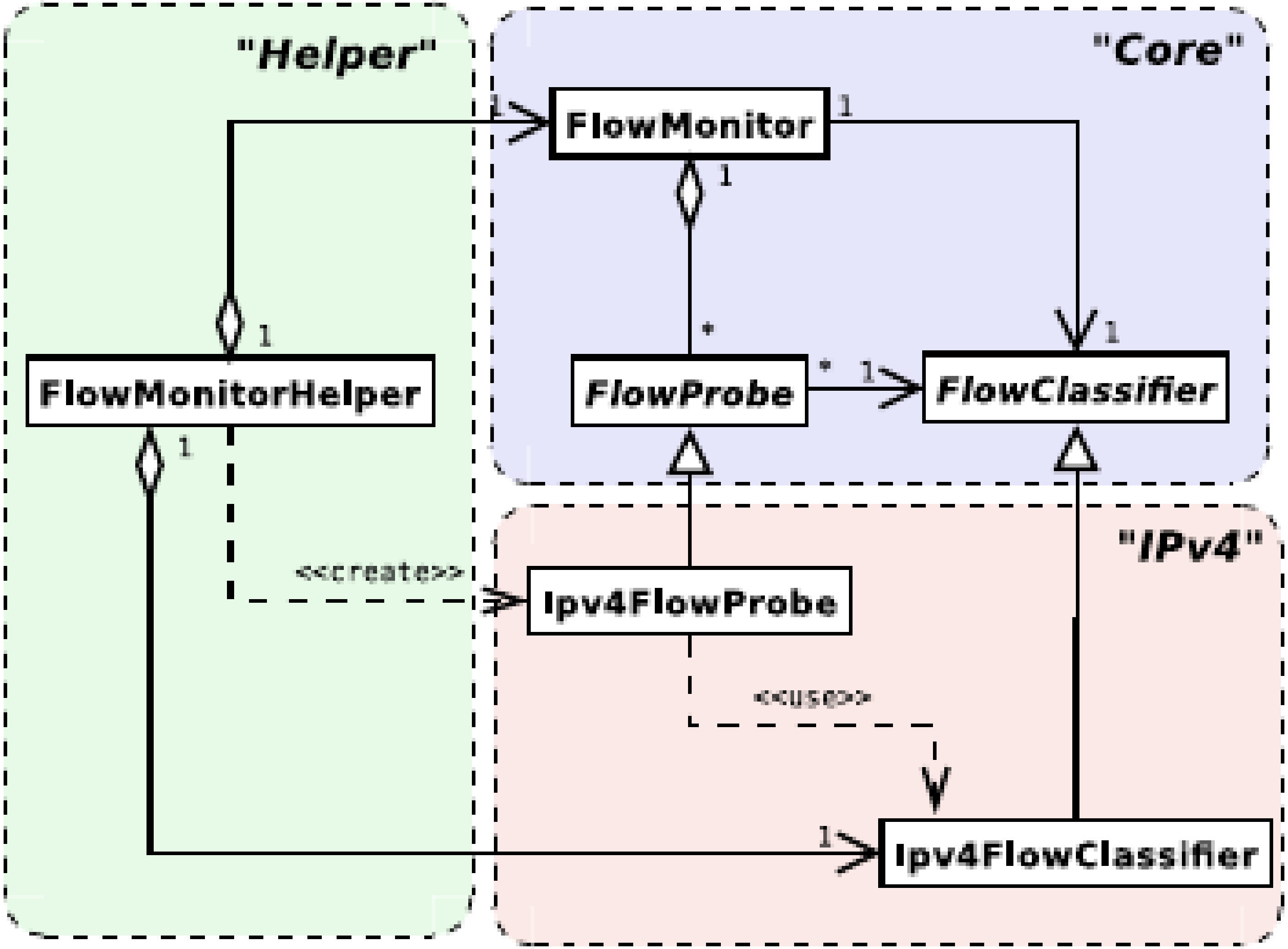
FlowMonitor

- To collect and store to persistent storage a common set of network performance metrics.
- To analyze the flow such as :-
  - Bitrates
  - Duration
  - Delays
  - PacketSize
  - Packet Loss Ratio

Inside FlowMonitor

- It is organized in three groups
  - **FlowMonitor**
    - It is responsible for coordinating efforts regarding probes and collect end-to-end flow statistics.
  - **FlowProbe**
    - It is responsible for listening for packet events in a specific point of the simulated space.
  - **FlowClassifier**
    - It provides a method to translate raw packet data into abstract "flow identifier" and "packet identifier" parameters

High Level View FlowMonitor



FlowMonitorHelper

- Code Flow

```
Step-1 Create flow_monitor using FlowMonitorHelper

Step-2 Install flow monitor on all nodes

Step-3 Create Classifier

Step-4 Map Flow Id with FlowStat
Step-5 Loop : to read flow
      {
         . . .
      }
```

Example

- Use Scenario 0.2 i)
  - **<u>Task</u>** – Calculate End-to-End throughput

    <u>Throughput</u> *– rate of successful message delivery over a communication channel (bits per seconds)*