CSE 322 : Computer Networks Sessional

NS3 PROJECT : TCP-AR (Adaptive Reno)

1705044

Modification：Congestion Window Increase

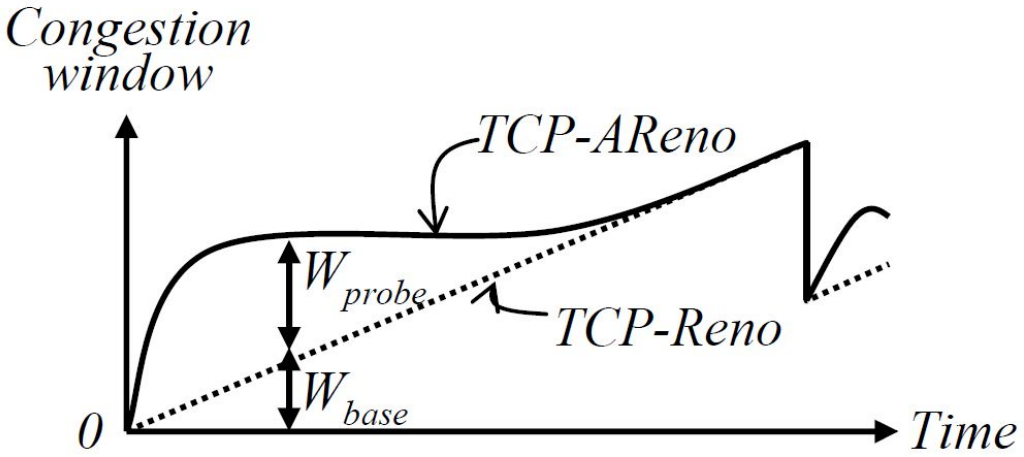
- 

Fig. 1: Congestion window increase of TCP-AReno during congestion avoidance

Modification：Congestion Window Increase

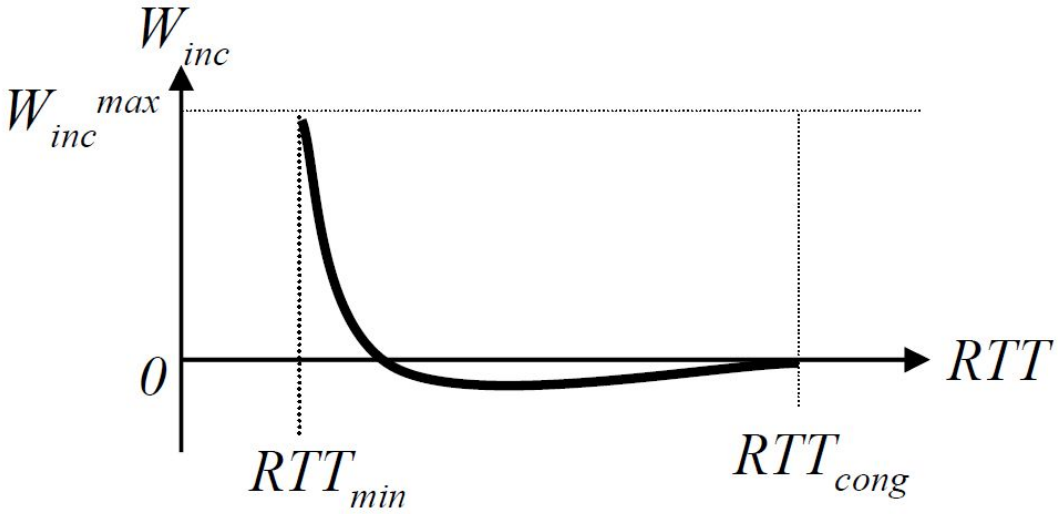
- 

Fig. 2: Congestion window increase per RTT (W_{probe} part)

Modification：Congestion Window Decrease

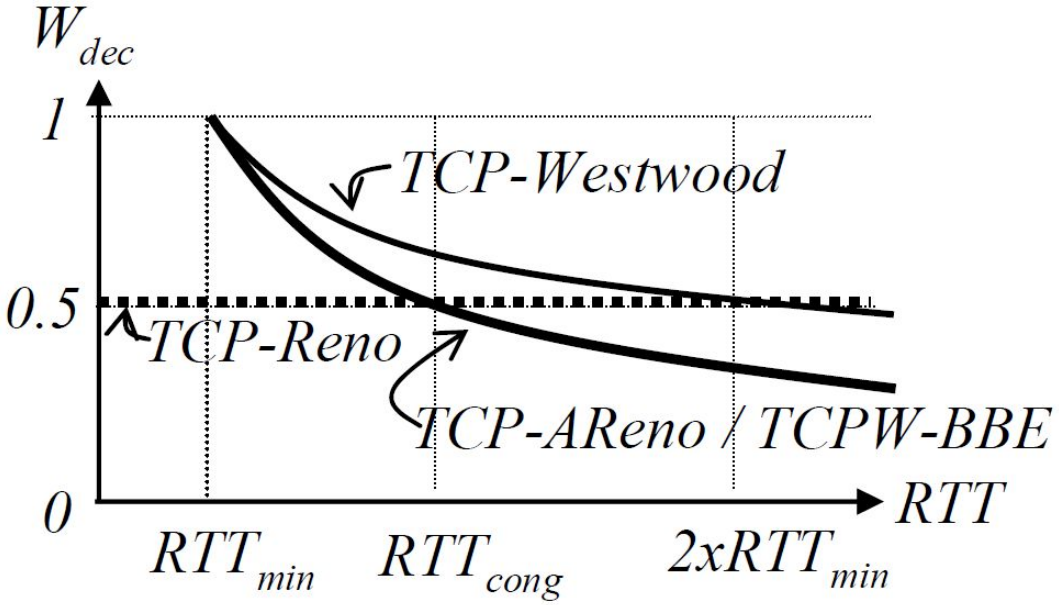
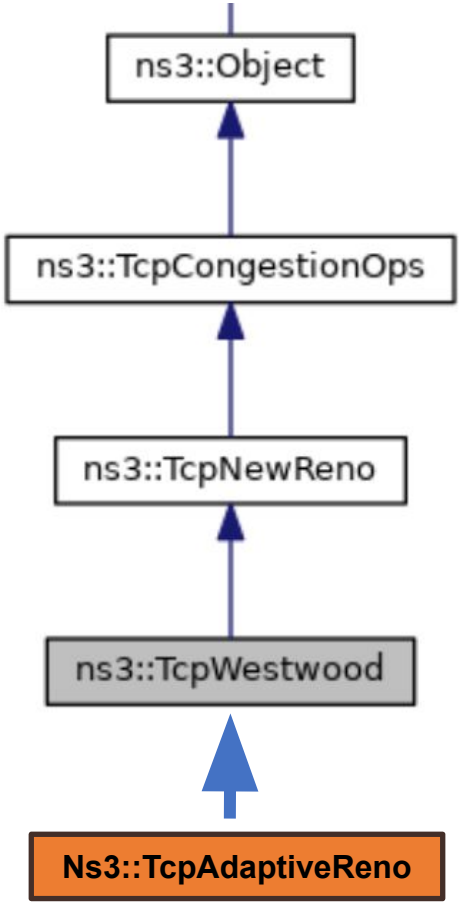
- 

Fig. 3: Congestion window decrease at a packet loss

Class Hierarchy



Variables

TCP Westwood

```
protected:
  TracedValue<double>     m_currentBW;          //!< Current value of the estimated BW
  double                  m_lastSampleBW;       //!< Last bandwidth sample
  double                  m_lastBW;             //!< Last bandwidth sample after being filtered
  enum ProtocolType       m_pType;              //!< 0 for Westwood, 1 for Westwood+
  enum FilterType         m_fType;              //!< 0 for none, 1 for Tustin

  uint32_t                m_ackedSegments;      //!< The number of segments ACKed between RTTs
  bool                    m_IsCount;            //!< Start keeping track of m_ackedSegments for Westwood+ if TRUE
  EventId                 m_bwEstimateEvent;    //!< The BW estimation event for Westwood+
  Time                    m_lastAck;            //!< The last ACK time
```

TCP AdaptiveReno

```
protected:
  Time                    m_minRtt;             //!< Minimum RTT
  Time                    m_maxRtt;             //!< Maximum RTT (j th event)
  Time                    m_currentRtt;         //!< Current RTT
  Time                    m_prevMaxRtt;         //!< Previous Maximum RTT (j-1 th event)

  // Window calculations
  uint32_t                m_incWnd;             //!< Increment Window
  uint32_t                m_probeWnd;           //!< Probe Window
```

Functions

TCP NewReno

```
virtual std::string GetName () const;
virtual uint32_t GetSsThresh (Ptr<const TcpSocketState> tcb, uint32_t bytesInFlight);
virtual void IncreaseWindow (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked);
virtual void PktsAcked (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked,const Time& rtt);
virtual Ptr<TcpCongestionOps> Fork ();
virtual void CwndEvent (Ptr<TcpSocketState> tcb, const TcpSocketState::TcpCaEvent_t event);
```

TCP Westwood

```
virtual uint32_t GetSsThresh (Ptr<const TcpSocketState> tcb, uint32_t bytesInFlight);
virtual void PktsAcked (Ptr<TcpSocketState> tcb, uint32_t packetsAcked, const Time& rtt);
void UpdateAckedSegments (int acked);
void EstimateBW (const Time& rtt, Ptr<TcpSocketState> tcb);
```

TCP AdaptiveReno

```
double EstimateCongestionLevel(const Time& rtt, Ptr<TcpSocketState> tcb);
void EstimateIncWnd(const Time& rtt, Ptr<TcpSocketState> tcb);
```

Modifications

- Need to initialize and track several variables in ***PktsAcked()*** function

```
/*
The function is called every time an ACK is received (only one time
also for cumulative ACKs) and contains timing information
*/
void
TcpAdaptiveReno::PktsAcked (Ptr<TcpSocketState> tcb, uint32_t packetsAcked,
                            const Time& rtt)
{
  NS_LOG_FUNCTION (this << tcb << packetsAcked << rtt);

  if (rtt.IsZero ())
    {
      NS_LOG_WARN ("RTT measured is zero!");
      return;
    }

  m_ackedSegments += packetsAcked;

  /*

      INITIALIZE AND SET VALUES FOR
          m_minRtt, m_maxRtt, m_currentRtt, m_prevMaxRtt
          m_incWnd, m_probeWnd

  */

  EstimateBW (rtt, tcb);
}
```

Modifications

```
double
TcpAdaptiveReno::EstimateCongestionLevel(const Time &rtt, Ptr<TcpSocketState> tcb)
{
  /*

      VARIABLE m_a = EXPONENTIAL SMOOTHING FACTOR
      m_maxRTT = (1-m_a)*m_prevMaxRtt + m_a*m_currentRtt

      RETURN:
      congestion_level = min((
          (m_currentRtt - m_minRtt) / (m_maxRTT - m_minRtt)
      ), 1)

  */
}
```

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

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

- Retrieve the current congestion level in ***EstimateCongestionFunction()*** function

Modifications

```
void
TcpAdaptiveReno::EstimateIncWnd(const Time& rtt, Ptr<TcpSocketState> tcb)
{
  /*

      c = EstimateCongestionLevel() // congestion level
      scalingFactor_m = 10 // in Mbps
      m_maxIncWnd = EstimateBW() / scalingFactor_m

      alpha = 10
      beta = 2 * m_maxIncWnd * (1/alpha - (1/alpha + 1)/(e^alpha))
      gamma = 1 - 2 * m_maxIncWnd * (1/alpha - (1/alpha + 1/2)/(e^alpha))

      m_incWnd = (m_maxIncWnd / (e^(alpha * c))) + (beta * c) + gamma

  */
}
```

$$W_{inc}^{max} = B / M * MSS.$$

$$\beta = 2W_{inc}^{max}(1/\alpha - (1/\alpha+1)/e^{\alpha})$$
$$\gamma = 1 - 2W_{inc}^{max}(1/\alpha - (1/\alpha+1/2)/e^{\alpha}).$$

$$W_{inc}(c) = W_{inc}^{max}/e^{\alpha c} + \beta c + \gamma$$

- Retrieve the current increment window size in ***EstimateIncWindow()*** function

Modifications

```
void
TcpAdaptiveReno::CongestionAvoidance (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
{
  /*

      base_window = USE NEW RENO IMPLEMENTATION
      m_probeWnd = max(m_probeWnd + m_incWnd / current_window,  0)
      current_window = base_window + m_probeWnd

  */
}
```

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

- Calculate the current window size for the Congestion Avoidance Phase in ***CongestionAvoidance()*** function

Modifications

```
uint32_t
TcpAdaptiveReno::GetSsThresh (Ptr<const TcpSocketState> tcb,
                              uint32_t bytesInFlight)
{
  /*

    c = EstimateCongestionLevel() // congestion level
    RETURN:
    new_window = current_window / (1 + c)

  */

}
```

$$W_{base} = W^{*}W_{dec} = W / (1+c), \qquad W_{probe} = 0.$$

- Calculate the Slow Start Threshold in the ***GetSsThresh()*** function

Reference

- H. Shimonishi and T. Murase, “**Improving efficiency friendliness tradeoffs of TCP congestion control algorithm**,” in Proc. IEEE GLOBECOM, 2005.

- Multiple NS3 Tutorials and documentations.

Paper Link

Thank you

Any Questions?