CSE 322

TCP-FUSION : A HYBRID CONGESTION CONTROL

Project Presentation
Student ID: 1705048

In Short

- The paper targets to calculate a congestion window size
- The algorithm determines how the congestion window size is increased or decreased

A. Congestion Window Reduction

B. Congestion Window Increase

Reducing the Congestion Window

- If last RTT grows more than 2*RTT min, congestion is detected.
- In TCP Reno, the congestion window is halved when congestion is detected
- But in this work, the congestion window will be reduced using the following formula-

$$cwnd_{new} = \max\left(\frac{RTT_{\min}}{RTT} cwnd_{last}, \frac{cwnd_{last}}{2}\right)$$

Reducing the Congestion Window

- Details:

$$cwnd_{new} = \max\left(\frac{RTT_{\min}}{RTT} cwnd_{last}, \frac{cwnd_{last}}{2}\right)$$

where $cwnd_{new}$, and $cwnd_{last}$ are congestion window sizes right after and before the packet loss, respectively.

RTT_{min} and RTT are the minimum RTT and the RTT right before the packet loss, respectively.

Increasing or Decreasing the Congestion Window

In this work, the congestion window will be increased or decreased using the following formulas:

$$diff = cwnd \frac{(RTT - RTT_{\min})}{RTT}$$

$$cwnd_{new} = \begin{cases} cwnd_{last} + W_{inc} / cwnd_{last}, & \text{if } diff < \alpha \\ cwnd_{last} + (-diff + \alpha) / cwnd_{last}, & \text{if } diff > 3 * \alpha \\ cwnd_{last}, & \text{otherwise} \end{cases}$$

$$\text{cwnd}_{\text{new}} = reno\_cwnd, \quad \text{if } cwnd_{new} < reno\_cwnd$$

Increasing the Congestion Window

We need to calculate a threshold value alpha

$$cwnd_{new} = \begin{cases} cwnd_{last} + W_{inc} / cwnd_{last}, & \text{if } diff < \alpha \\ cwnd_{last} + (-diff + \alpha) / cwnd_{last}, & \text{if } diff > 3 * \alpha \\ cwnd_{last}, & \text{otherwise} \end{cases}$$

$$\text{cwnd}_{\text{new}} = reno\_cwnd, \quad \text{if } cwnd_{new} < reno\_cwnd$$

Increasing the Congestion Window

If the *diff* is less than the *lower bound threshold*, the link is determined as underutilized, and its congestion window size is increased rapidly to fill the pipe size.

$$cwnd_{new} = \begin{cases} cwnd_{last} + W_{inc} / cwnd_{last}, & \text{if } diff < \alpha \\ cwnd_{last} + (-diff + \alpha) / cwnd_{last}, & \text{if } diff > 3 * \alpha \\ cwnd_{last}, & \text{otherwise} \end{cases}$$

$$\text{cwnd}_{\text{new}} = reno\_cwnd, \quad \text{if } cwnd_{new} < reno\_cwnd$$

Increasing the Congestion Window

Winc is the increment parameter to increase congestion window size rapidly.

$$cwnd_{new} = \begin{cases} cwnd_{last} + W_{inc} / cwnd_{last}, & \text{if } diff < \alpha \\ cwnd_{last} + (-diff + \alpha) / cwnd_{last}, & \text{if } diff > 3 * \alpha \\ cwnd_{last}, & \text{otherwise} \end{cases}$$

$$\text{cwnd}_{\text{new}} = reno\_cwnd, \quad \text{if } cwnd_{new} < reno\_cwnd$$

Increasing the Congestion Window

If the *diff* is larger than the *upper bound threshold*, the link is determined as utilized and early congestion, and its congestion window size is decreased to the value that has at least the *lower bound threshold* in the bottleneck queue.

$$cwnd_{new} = \begin{cases} cwnd_{last} + W_{inc} / cwnd_{last}, & \text{if } diff < \alpha \\ cwnd_{last} + (-diff + \alpha) / cwnd_{last}, & \text{if } diff > 3 * \alpha \\ cwnd_{last}, & \text{otherwise} \end{cases}$$

$$\text{cwnd}_{\text{new}} = reno\_cwnd, \quad \text{if } cwnd_{new} < reno\_cwnd$$

Setting Parameter

- Calculating alpha :

RE is the achieved rate estimation governed by TCPW-RE (Rate Estimation) scheme.

B is the bandwidth of bottleneck link.

coexisting *N* TCP-Fusion flows

$$\alpha = \frac{G}{N} = \frac{(B/N)*D_{\min}}{packet\_size*8} \approx \frac{RE*D_{\min}}{packet\_size*8} = cwnd\frac{D_{\min}}{RTT}$$

an assumption that no routers have smaller than *G* packets that corresponds to the queuing delay D_{min} in the bottleneck queue.

Setting Parameter

- Calculating Winc:

B is the bandwidth of bottleneck link.

$$W_{inc} = \frac{B * D_{\min}}{8 * packet\_size}$$

an assumption that no routers have smaller than G packets that corresponds to the queuing delay D_{min} in the bottleneck queue.

Implementation in NS-3

- Need to write a new class named: TCP Fusion

Implementation in NS-3

ns-allinone-3.29 | ns-3.29 | src | internet | model

- Working directory:

All algorithms of TCP (like TCP vegas or westwood) are implemented in these files

Implementation in NS-3

- Create new file named `tcp-fusion.h` to write new TCPFusion class.

tcp-vegas.h

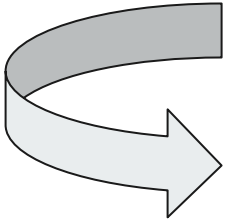
D: > drive > ns-allinone-3.35 > ns-3.35 > src > internet > model > tcp-vegas.h

```
class TcpVegas : public TcpNewReno
{
public:
  /**
   * \brief Get the type ID.
   * \return the object TypeId
   */
  static TypeId GetTypeId (void);

  /**
   * Create an unbound tcp socket.
   */
  TcpVegas (void);

  /**
   * \brief Copy constructor
   * \param sock the object to copy
   */
  TcpVegas (const TcpVegas& sock);
  virtual ~TcpVegas (void);

  virtual std::string GetName () const;
```

Implementation in NS-3

- Create new file named `tcp-fusion.cc` to implement TCPFusion .

Add attributes like alpha, increment factor here.



tcp-vegas.cc

D: > drive > ns-allinone-3.35 > ns-3.35 > src > internet > model > tcp-vegas.cc

```
#include "tcp-socket-state.h"

#include "ns3/log.h"

namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("TcpVegas");
NS_OBJECT_ENSURE_REGISTERED (TcpVegas);

TypeId
TcpVegas::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::TcpVegas")
    .SetParent<TcpNewReno> ()
    .AddConstructor<TcpVegas> ()
    .SetGroupName ("Internet")
    .AddAttribute ("Alpha", "Lower bound of packets in network",
                   UintegerValue (2),
                   MakeUintegerAccessor (&TcpVegas::m_alpha),
                   MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("Beta", "Upper bound of packets in network",
                   UintegerValue (4),
                   MakeUintegerAccessor (&TcpVegas::m_beta),
                   MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("Gamma", "Limit on increase",
                   UintegerValue (1),
                   MakeUintegerAccessor (&TcpVegas::m_gamma),
                   MakeUintegerChecker<uint32_t> ())
  ;
  return tid;
}
```

Implementation in NS-3

- Existing algorithms are written in files as `tcp-vegas.cc`and `tcp-westwood.cc`
- Implement appropriate parameter initialization in the constructor of tcp-fusion.cc file like this

```
TcpVegas::TcpVegas (void)
  : TcpNewReno (),
    m_alpha (2),
    m_beta (4),
    m_gamma (1),
    m_baseRtt (Time::Max ()),
    m_minRtt (Time::Max ()),
    m_cntRtt (0),
    m_doingVegasNow (true),
    m_begSndNxt (0)
{
  NS_LOG_FUNCTION (this);
}

TcpVegas::TcpVegas (const TcpVegas& sock)
  : TcpNewReno (sock),
    m_alpha (sock.m_alpha),
    m_beta (sock.m_beta),
    m_gamma (sock.m_gamma),
    m_baseRtt (sock.m_baseRtt),
    m_minRtt (sock.m_minRtt),
    m_cntRtt (sock.m_cntRtt),
    m_doingVegasNow (true),
    m_begSndNxt (0)
```

Implementation in NS-3

- implement `IncreaseWindow`method in `tcp-fusion.cc` using the following formulas-

```
::IncreaseWindow (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
```

$$diff = cwnd \frac{(RTT - RTT_{\min})}{RTT}$$

$$cwnd_{new} = \begin{cases} cwnd_{last} + W_{inc} / cwnd_{last}, & \text{if } diff < \alpha \\ cwnd_{last} + (-diff + \alpha) / cwnd_{last}, & \text{if } diff > 3 * \alpha \\ cwnd_{last}, & \text{otherwise} \end{cases}$$

$$\text{cwnd}_{\text{new}} = reno\_cwnd, \quad \text{if } cwnd_{new} < reno\_cwnd$$

Implementation in NS-3

- implement `CongestionAvoidance`method in `tcp-fusion.cc` using the following formulas-

```
::CongestionAvoidance (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
```

$$cwnd_{new} = \max\left(\frac{RTT_{\min}}{RTT} cwnd_{last}, \frac{cwnd_{last}}{2}\right)$$

Summary

- Create new class files associated with tcp-fusion
- Implement the congestion window increasing and reducing methods

End of Presentation