TCP CUBIC in ns-3

CS577

Brett Levasseur

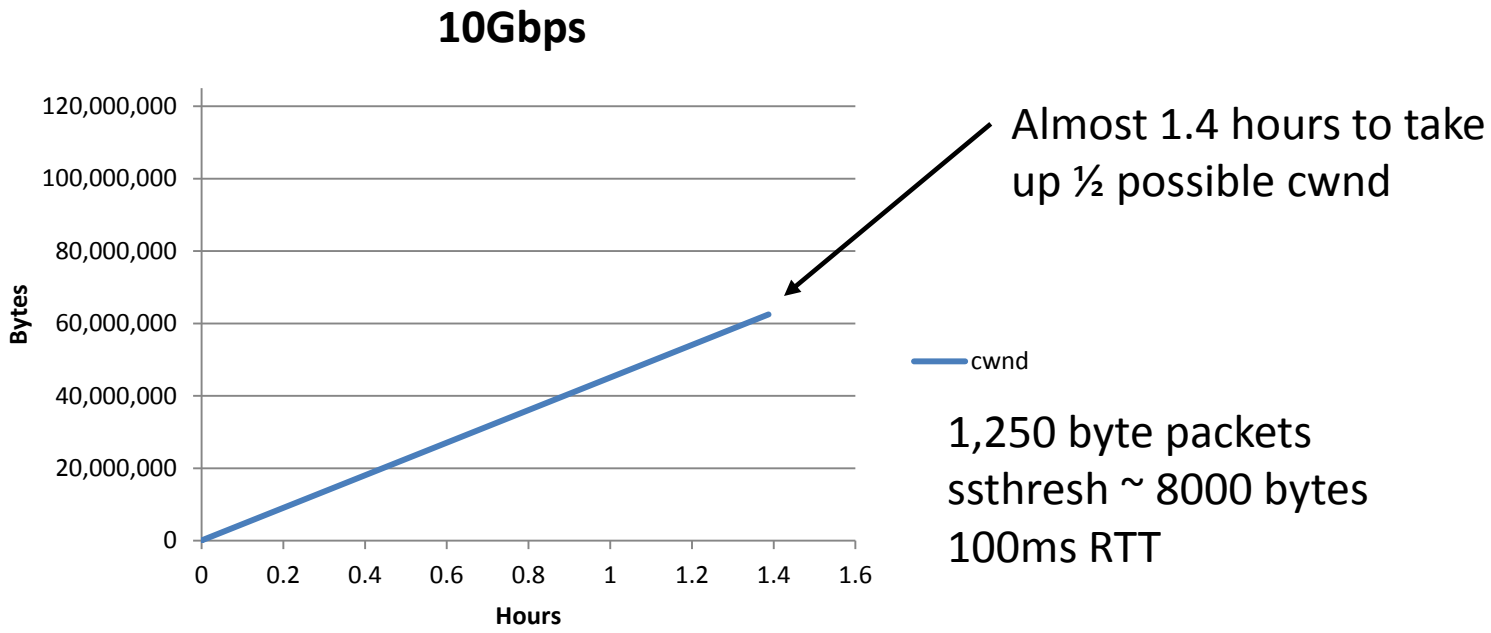
Outline

- Introduction
- CUBIC
- CUBIC in Linux
- ns-3 Implementation
- Results
- Conclusions

Introduction

WPI

- TCP grows cwnd too slowly for large bandwidth connections
- New TCP Variant needed



CUBIC

WPI

- BIC was first attempt
- CUBIC simplified and improved upon BIC
- Grow cwnd slower around loss events

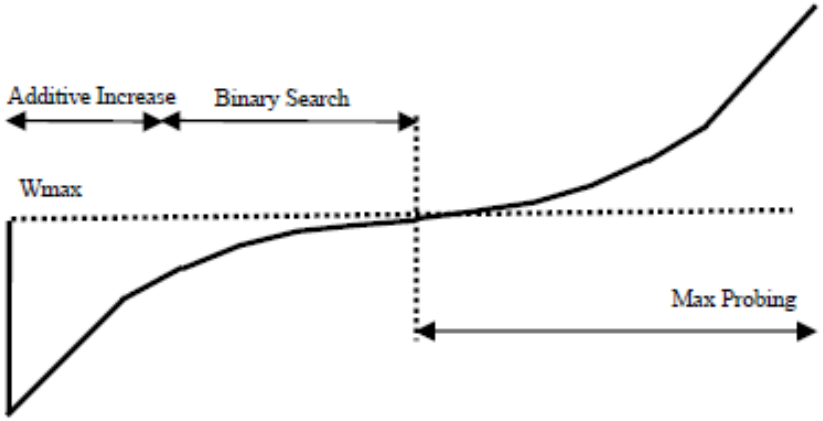


Fig. 1: The Window Growth Function of BIC

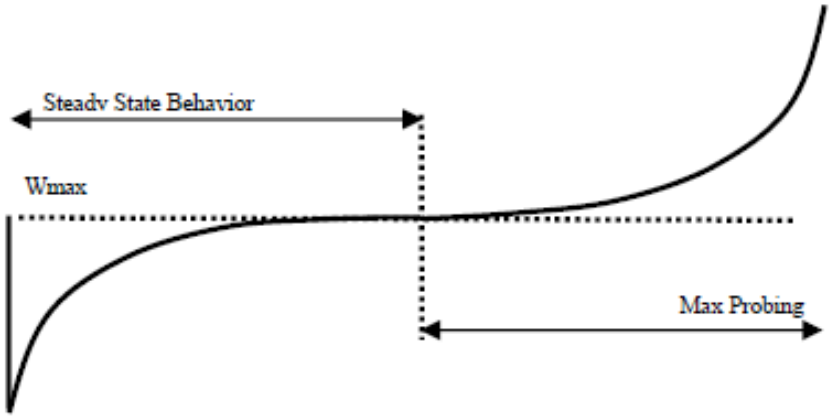


Fig. 2: The Window Growth Function of CUBIC

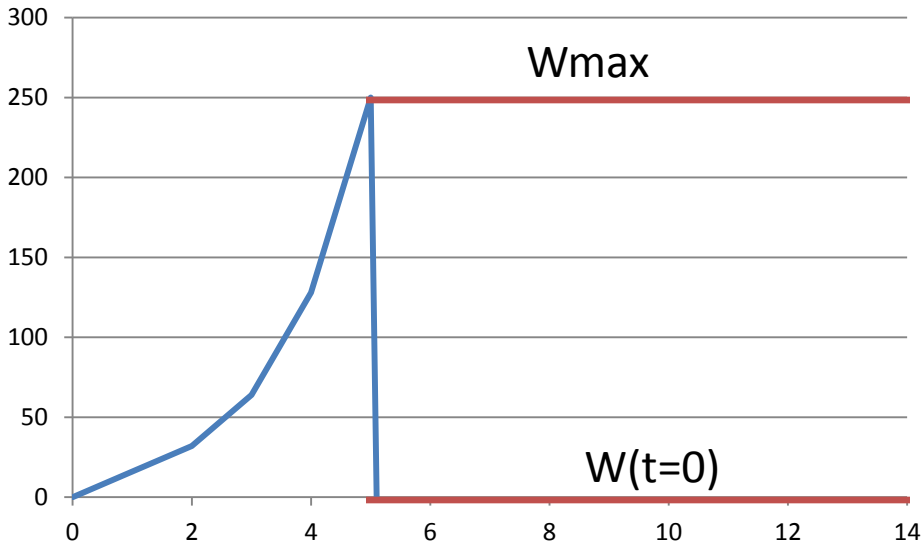
CUBIC Basics

- cwnd growth $$W(t) = C(t-K)^3 + W_{max}$$
- Packet loss $$K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

C	CUBIC parameter
t	Elapsed time from the last window reduction
K	Time period to increase W to Wmax
W	Current cwnd
Wmax	cwnd at last window reduction
β	Window decrease constant

CUBIC Basics

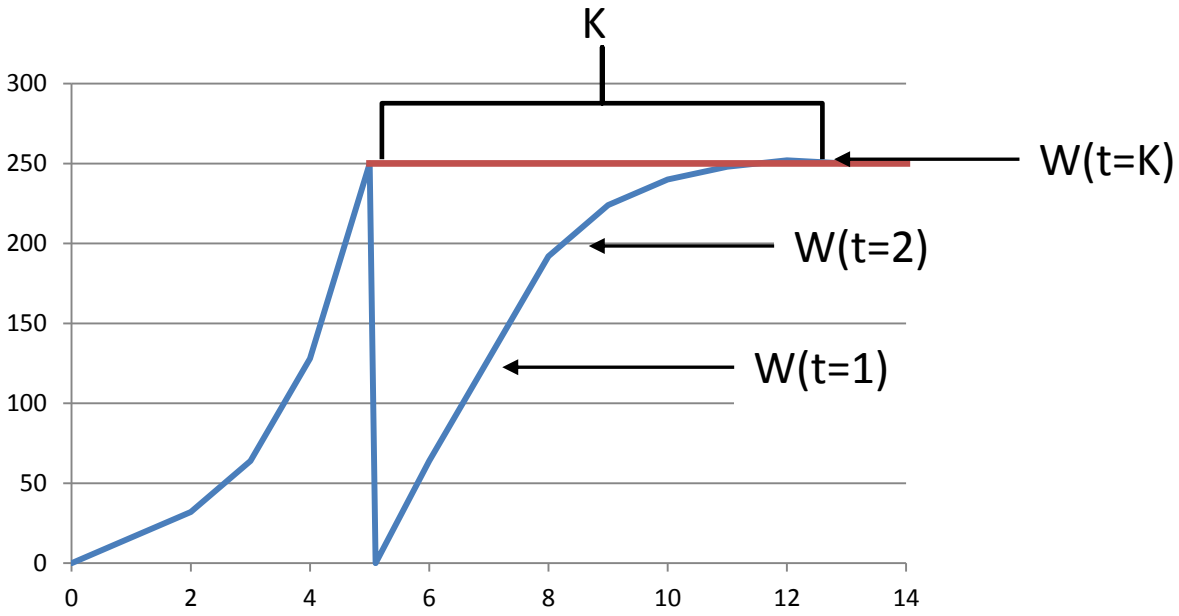
WPI



- At loss event set Wmax, reduce cwnd by β and calculate K

CUBIC Basics

WPI



- cwnd grows back to K when t = K

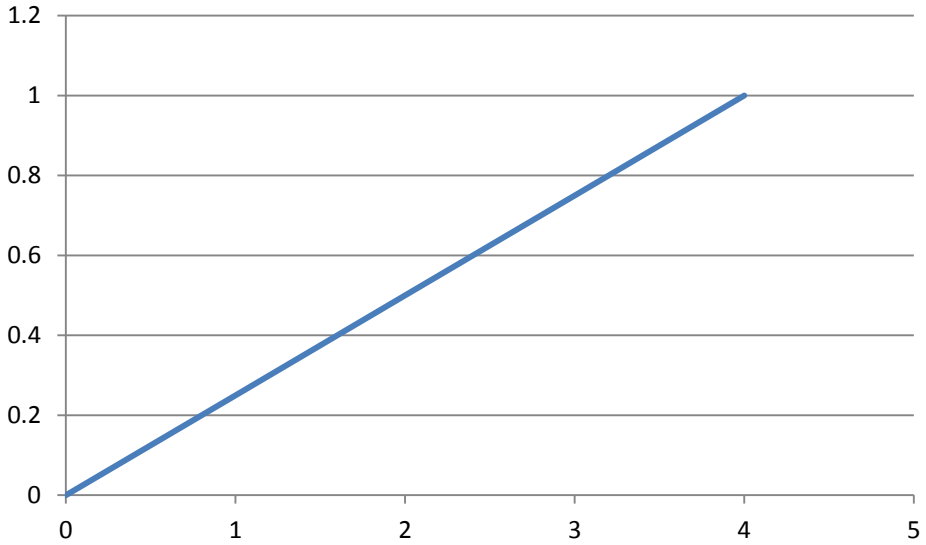
CUBIC in Linux

- Not implemented as in the CUBIC paper
- cwnd grows in increments of segment sizes
- Custom method for calculating cube roots
- Checks for error conditions
- Unit scaling

Growing cwnd

WPI

- Linux only grows cwnd by full segments
- CUBIC can grow cwnd less than full segment
- Same impact by increasing amount of time between updates



0	**0**
1	0.25
2	0.5
3	0.75
4	**1**

0	**0**
4	**1**

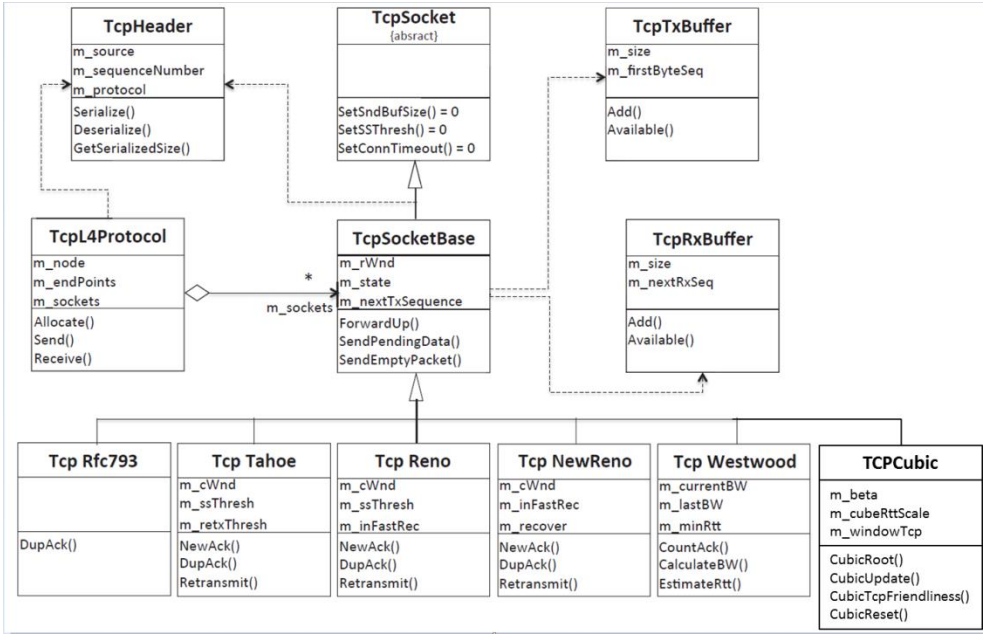
Scaling in CUBIC

- Most scaling is related to time
- Variable ‘t’ measured with TCP timestamps
  - Timestamps use clock cycles to increment
  - Units are called jiffies in the Linux Kernel
- Number of milliseconds in a jiffy depends on the CPU’s clock
- Scaling required to get time units correct

ns-3 Implementation

WPI

- Object oriented design
- Generic TCP defined
- TCP variants are extended from base
- TCP headers and buffers provided
- Added TcpCubic object
  - tcp-cubic.cc
  - tcp-cubic.h



ns-3 Methods

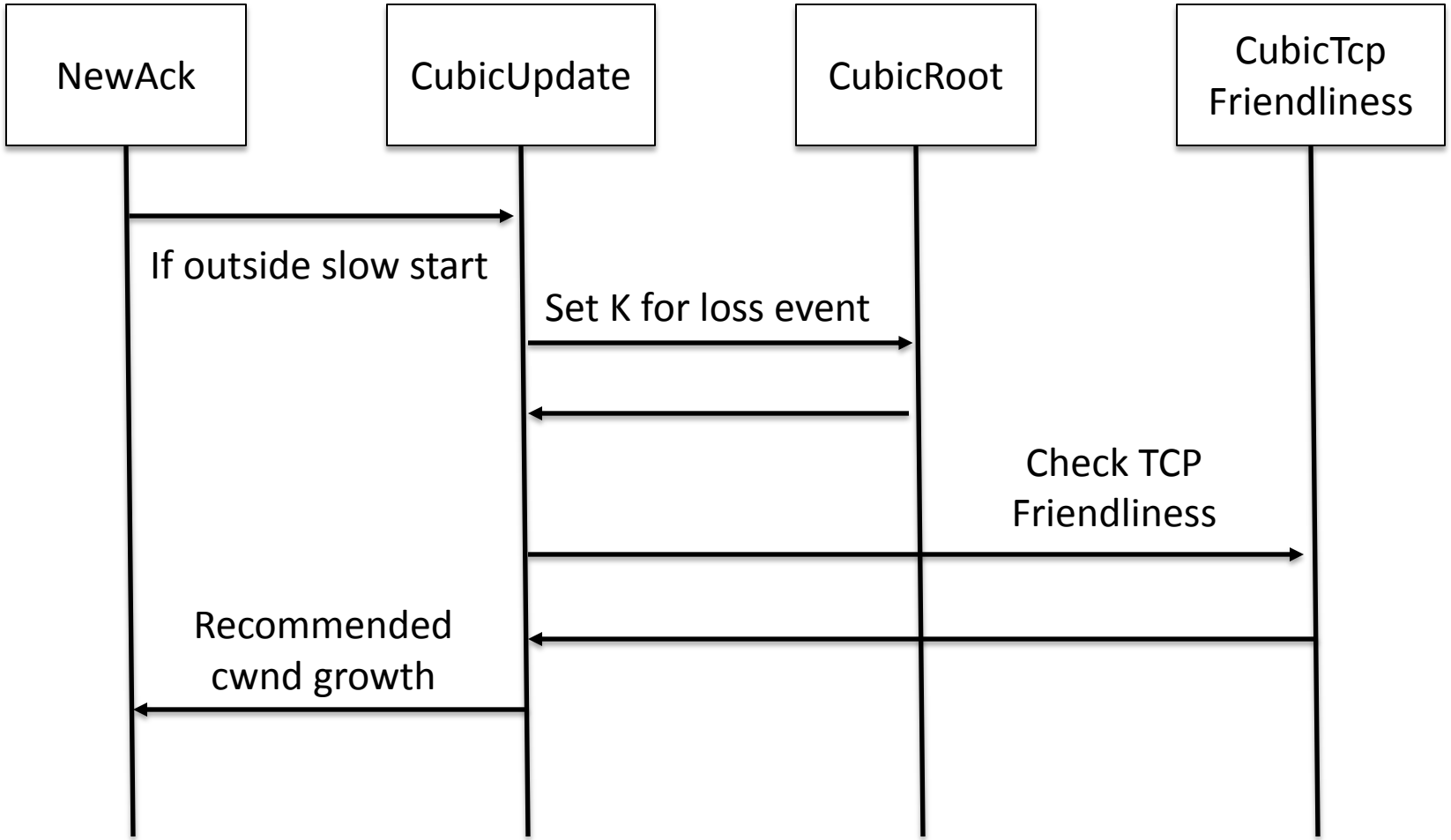
- NewAck – called for every new ACK received
  - Normal cwnd updates in slow start
  - CUBIC updates otherwise
- DupAck – called for every duplicate ACK received
  - Normal operation when < 3 duplicates
  - For 3 duplicate ACKs reduce cwnd

CUBIC Methods

- CubicRoot – Find the cubic root of a number
  - Based on Linux Kernel implementation
- CubicUpdate – Calculate the cwnd target for CUBIC
- CubicTcpFriendliness – Change the cwnd target for TCP Friendliness
- CubicReset – Reset CUBIC parameters

CUBIC Flow

WPI



Issues

- ns-3 does not have TCP timestamps
- Simulation clock used instead
- Requires adjustments to calculating ‘t’ due to different units
- Could remove the use of jiffy code but much of the Linux implementation relies on scaling factors based on the system clock

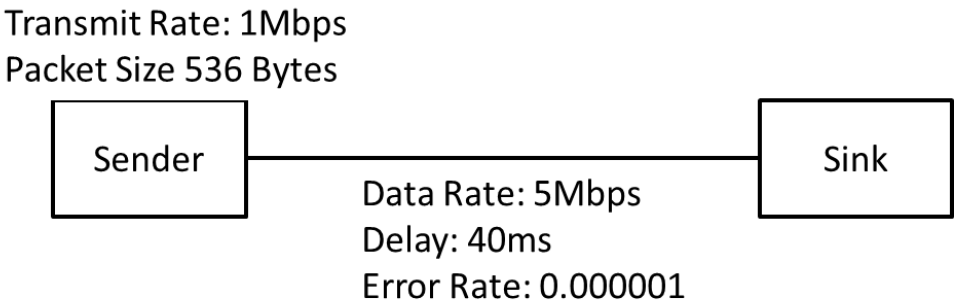
Results

- Compare to real world CUBIC example
- Examine simulation results
  - Verify cwnd reduction
  - Verify cwnd growth in relation to Wmax
- Compare simulated CUBIC to simulated NewReno

Simulation Scenario

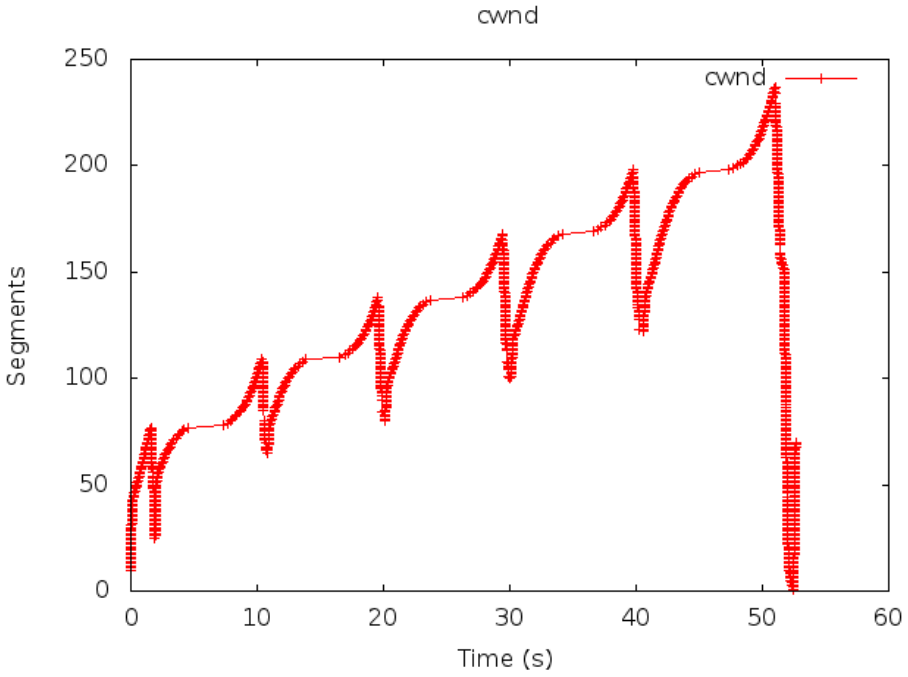
WPI

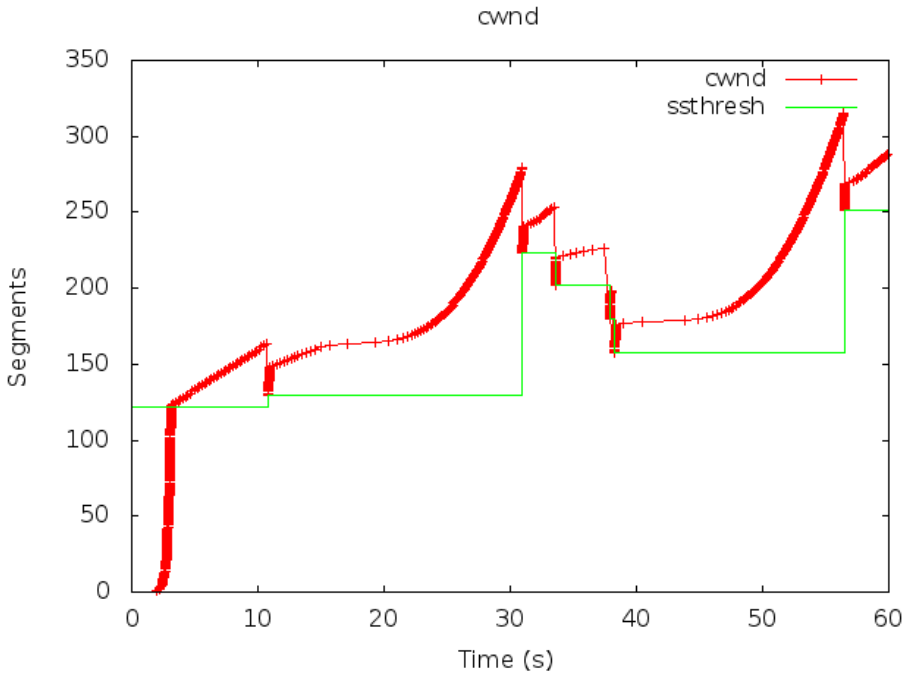
- Simple sender and sink topology
- Packet sizes 536 bytes
- Transmission rate 1Mbps
- Delay 40ms
- Error rate – Causes lost packets at the receiver



Measurements

WPI





- Measurement and simulation have similar CUBIC curve
- Number of segments similar

Packet Loss

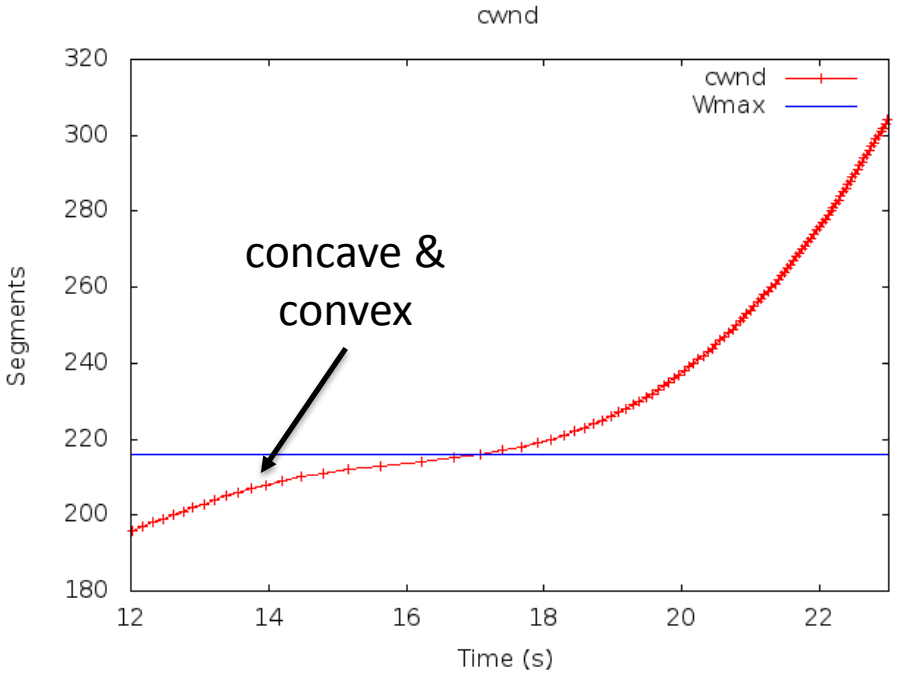
- Before loss cwnd = 216
- After loss cwnd = 172
- β = 819
- BICTCP_BETA_SCALE = 1024

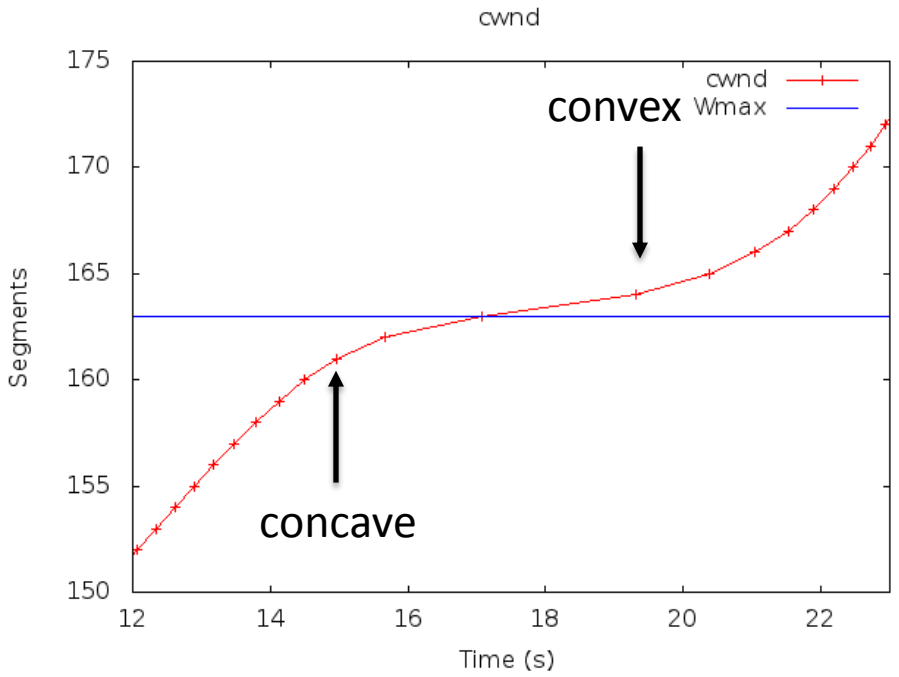
$$cwnd = ssthresh = max(\frac{cwnd * \beta}{BICTCP\_BETA\_SCALE}, 2)$$

$$172.76 = max(\frac{216 * 819}{1024}, 2)$$

CUBIC Growth

WPI



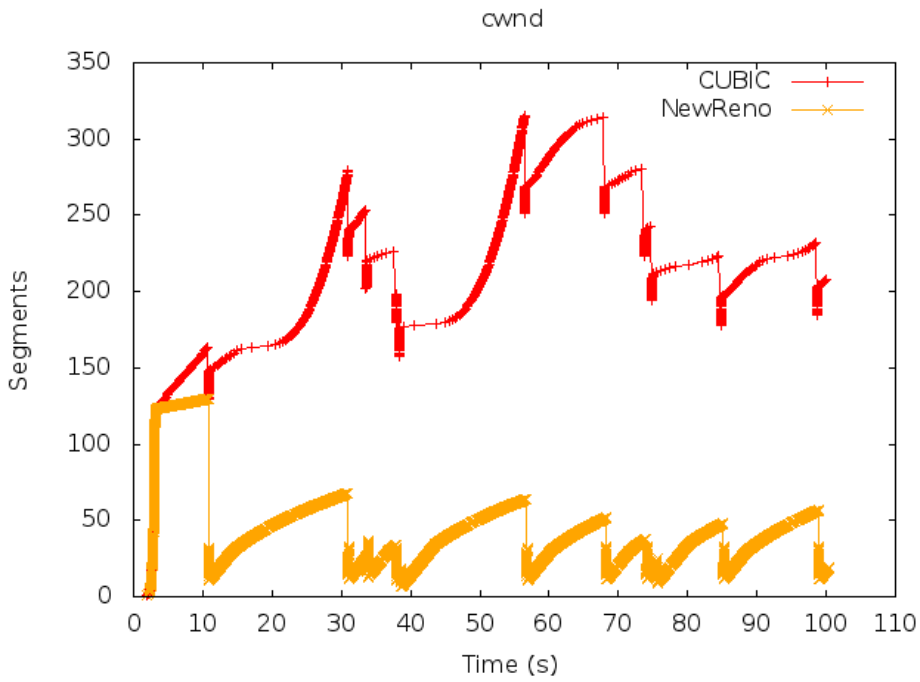


- Before and after additional scaling of ‘t’
- More work is needed for using simulator clock with ‘t’

NewReno Comparison

WPI

- Same simulation run with CUBIC and NewReno
- Both increment the same under slow start
- CUBIC grows cwnd faster
- CUBIC handles packet loss better than NewReno



Conclusions

- Created a CUBIC implementation in ns-3
- Similar cwnd growth to actual CUBIC measurements
- Current version outperforms NewReno
- Scaling adjustments required

Questions