Detailed Analysis of TCP BBR Implementation in Ns-3 Network Simulator under Heavy Traffic Loads

Toshihiko Kato†, Takahiko Kato‡, Ryo Yamamoto†,

Satoshi Ohzahata†

† University of Electro-Communications ‡ University of Fukui

Japan

I. Introduction

- A lot of TCP congestion control algorithms.
  - Loss-based approach: Detects a congestion by consecutive packet losses
    NewReno, HighSpeed TCP, CUBIC TCP, Hamilton TCP
  - Delay-based approach: Detects a congestion by an increase of RTT
    Vegas
  - Hybrid approach: Combines loss-based and delay based
    TCP Veno, Compound TCP
- TCP BBR (Bottleneck Bandwidth and Round-trip propagation time)
  - Congestion-based congestion control, proposed by Google in 2016
  - Estimate bottleneck bandwidth (BtlBw) and minimum RTT (RTprop)
  - Send data by rate-based control of BtlBw × RTprop

I. Introduction (2)

- Studies on BBR performance
- In early stages: BBR in Linux and physical network
- BBR software in ns-3: Implemented by Vivek Jain et al in version 3.27, in 2018.
  Officially included in ns-3 at version 3.34 released in 2021.
  Some performance evaluation studies.
- May be different from BBR software in Linux
- Our experience:
  - TCP BBR throughput test where sixteen BBR flows share a 1Gbps bottleneck link and output buffer to bottleneck link is limited.
  - Throughput of some BBR flows becomes extremely low.
  - Experimental results were different between ns-3 version 3.40 and version 3.44.

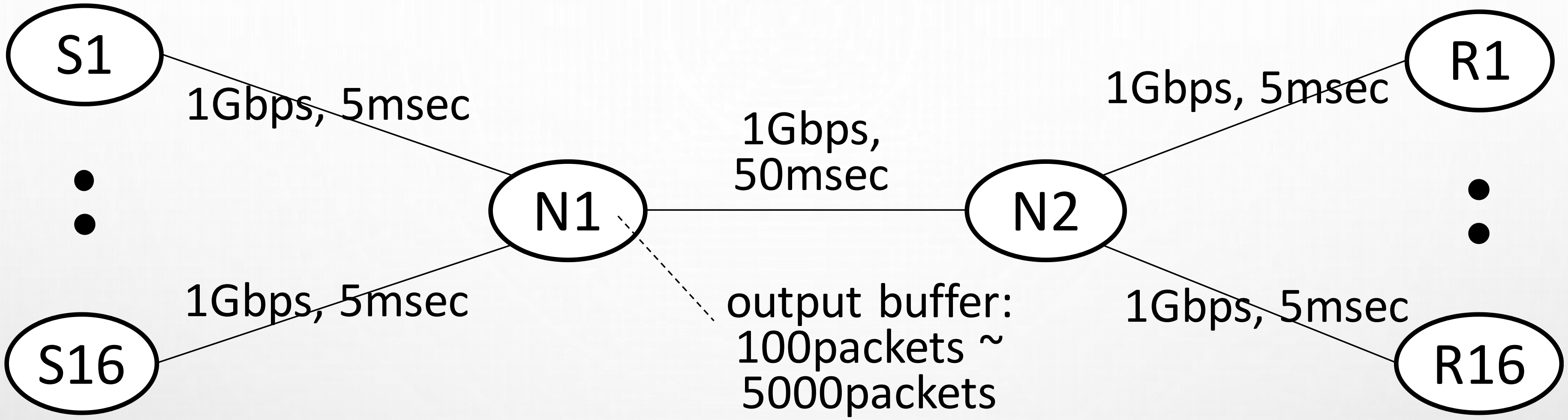
II. Backgrounds   A. Overview of TCP BBR

- TCP BBR estimates for the available bottleneck bandwidth, BtlBw, and the minimal round-trip propagation time, RTprop, to calculate a path's available BDP.
- A BBR sender data segments with the pacing rate given by pacing_gain × BltBw × RTprop with the help of pacing.
- The parameter pacing_gain, which is set to 1 most of the time, is used to control the actual pacing rate.
- BBR algorithm has four different phases: Startup, Drain, Probe Bandwidth (ProbeBW), and Probe RTT (ProbeRTT).
- ProbeRTT is shifted in time window based (every 10 second).

II. Backgrounds B. Performance Evaluation Studies

reference	BBR code	network	evaluation target	max number of flows
[10]	Linux	physical	BBR, BBR + CUBIC	6 BBRs, 1 BBR + 1 CUBIC
[11]	Linux	physical	BBR + CUBIC	10 BBRs + 10 CUBICs
[12]	Linux	Mininet	BBR, BBR + CUBIC	6 BBRs, 10 BBRs + 10 CUBICs
[13]	Linux	physical	BBR, BBR + CUBIC	8 BBRs, 3 BBRs + 3 CUBICs
[14]	ns-3 (Vivek Jain’s version)	ns-3	BBR, BBR + BIC, BBR + BIC + NewReno + Vegas	2 BBRs, 2 BBRs + 2 BIC, 1 BBR + 1 BIC + 1 NewReno + 1 Vegas
[15]	Linux	Mininet	BBR, (Delay-aware BBR)	2 BBRs
[16]	ns-3 (Vivek Jain’s version)	ns-3	BBR, (other BBR variants including BBRv2)	4 BBRs
[17]	ns-3	ns-3	BBR, BBR + CUBIC	8 BBRs, 2 BBRs + 2 CUBICs

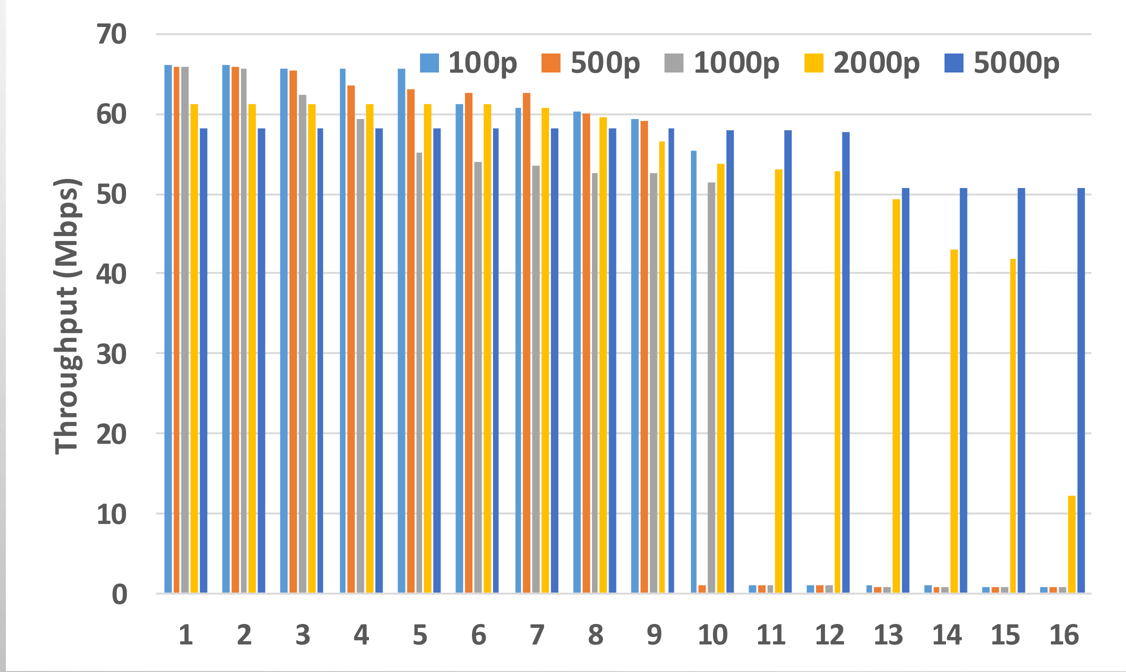
III. Performance Evaluation A. Experimental Setup

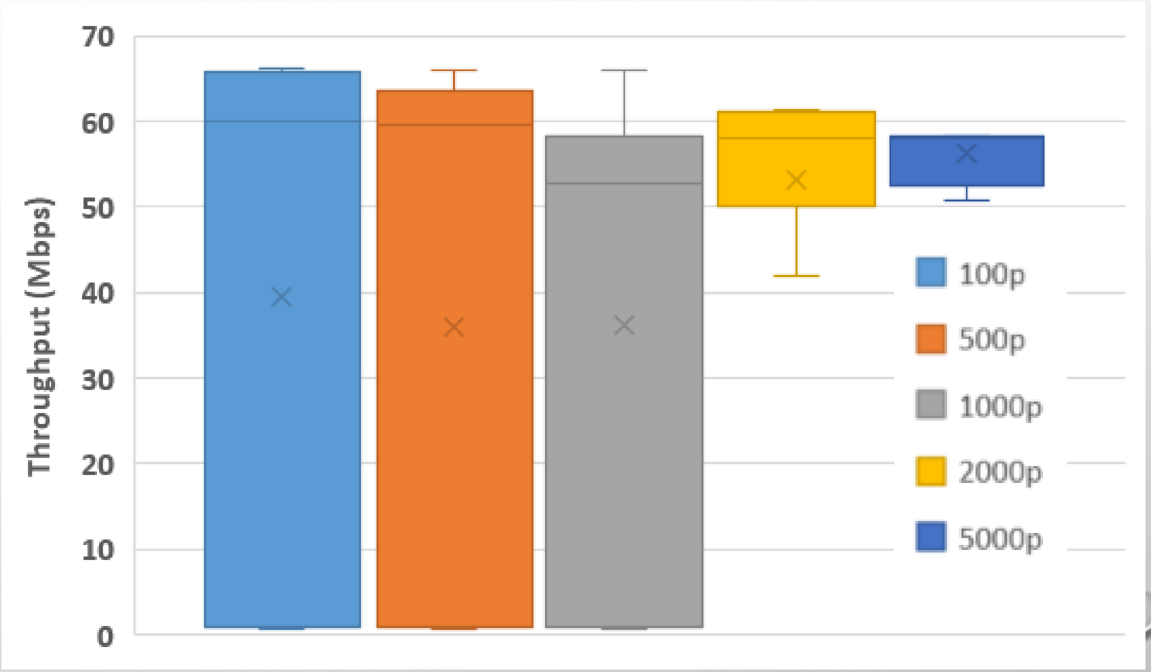


- Queue discipline: First-In First-Out queue discipline following the drop tail policy.
- TCP loss recovery algorithm: TCP classic recovery.
- DelAckCount (Number of packets to wait before sending a TCP Ack): 1.
- Explicit congestion notification functionality: not used.
- SACK: used.

III. Performance Evaluation B. Results with Version 3.40

- Output buffer set to 100, 500, 1000, 2000, 5000 packets

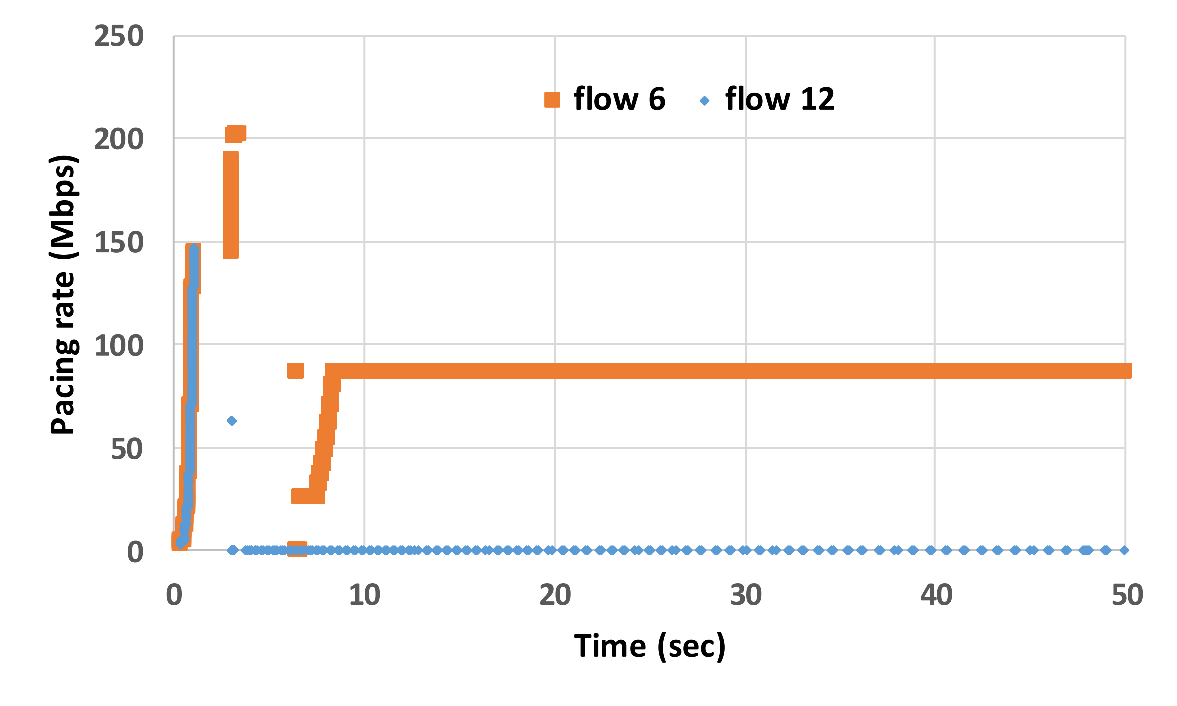




III. Performance Evaluation B. Results with Version 3.40

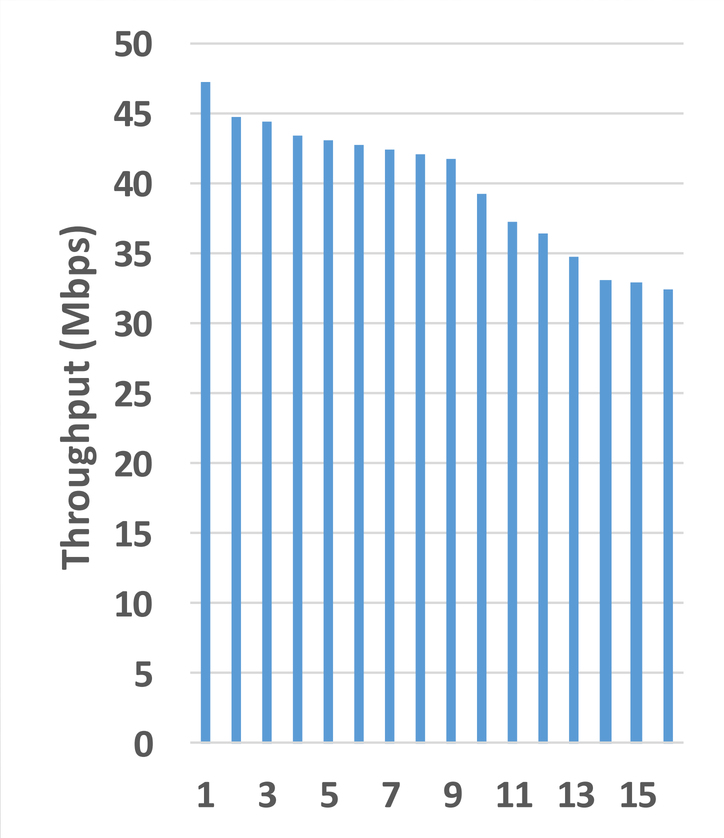
Throughput (Mbps), and pacing rate for high throughput flow and low throughput flow

buffer size	average	std. dev.	total
100 pkts.	39.5	31.0	632.0
500 pkts.	35.9	32.0	574.8
1000 pkts.	36.1	28.5	577.9
2000 pkts.	53.1	12.7	850.3
5000 pkts.	56.3	3.3	900.2

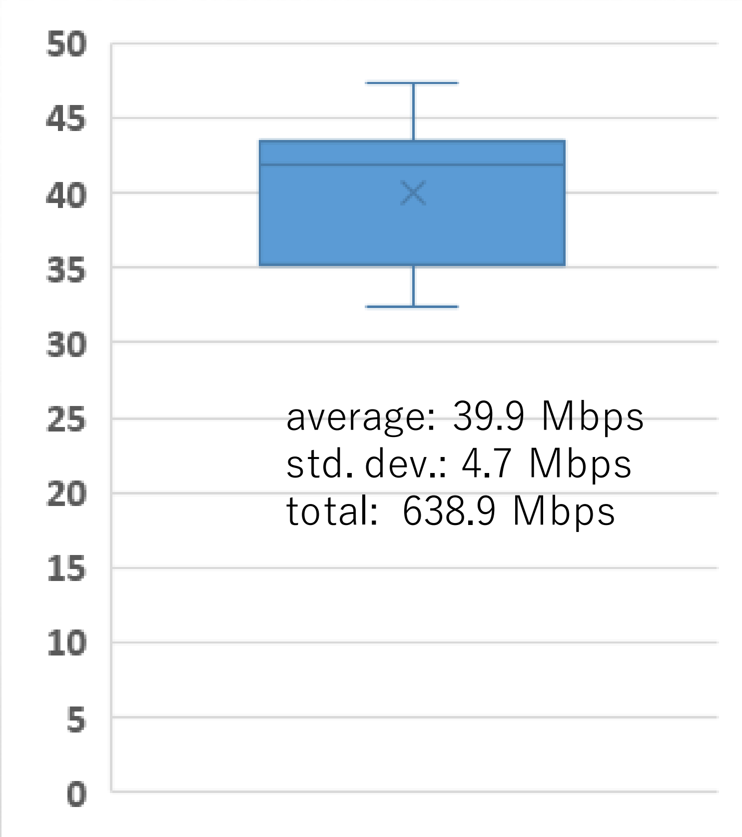


III. Performance Evaluation C. Results with Version 3.44

- There are no low throughput flows

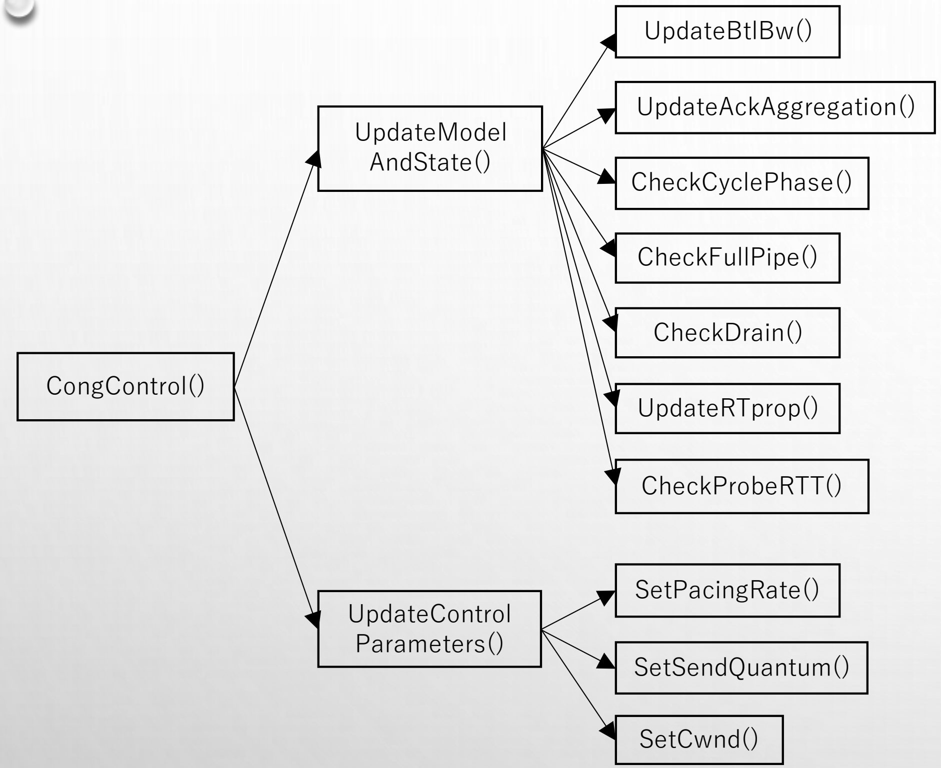


(a) Individual throughput

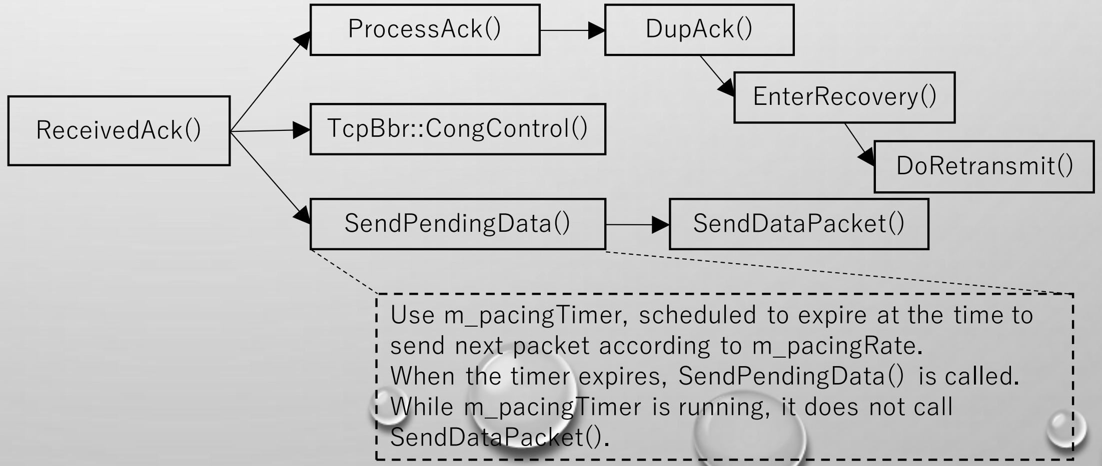


(b) Box-and-whisker plots

IV. Source Code Detailed Analysis A. Ns-3 Program Structure



function	behavior
UpdateBtlBw()	Updates maxBwFilter if a new delivery rate is larger than prior value.
UpdateAck Aggregation()	Takes account of delayed ACK.
CheckCyclePhase()	If in ProbeBW phase and a RTT elapsed, advance cycle.
CheckFullPipe()	If delivery rate becomes large enough, set m_isPipeFilled = true
CheckDrain()	If in Startup phase and m_isPipeFilled is true, enter Drain phase. If in Drain phase and inflight data decreased, enter ProbeBW phase.
UpdateRTprop()	If RTT didn't increase or W_R (10 sec) elapsed (m_rtPropExpired = true), update m_rtProp.
CheckProbeRTT()	If not in ProbeRTT phase and m_rtPropExpierd, enter ProbeRTT phase and save cwnd. If in ProbeRTT phase and 200 msec elapsed, enter ProbeBW if m_isPipeFilled or Startup otherwise.
SetPacingRate()	Set rate = Max value in maxBwFilter × pacing_gain. If m_isPipeFilled or rate > m_pacingRate, set m_pacingRate = rate.
SetSendQuantum()	Set m_sendQuantum = MSS.
SetCwnd()	Determine cwnd considering targetCwnd and packet losses. If in ProbeRTT phase, set cwnd to 4 MSS.



IV. Source Code Detailed Analysis B. Difference of Ns-3 3.40 and 3.44

Change history from 3.40 to3.44 related to TCP

1) TCP Cubic now supports TCP-friendliness by default, making the congestion window growth somewhat more aggressive. (from 3.40 to 3.41)

2) TcpCubic and TcpLinuxReno will no longer grow their cwnd when application-limited. (from 3.41 to 3.42)

3) Deprecated EventId::IsRunning(). It has been replaced with EventId::IsPending(). (from 3.41 to 3.42)

4) TCP Proportional Rate Reduction (PRR) recovery has been aligned to the updates in draft-ietf-tcpm-prr-rfc6937bis. (from 3.42 to 3.43)

5) A new trace source TcpSocketBase::LastRtt has been added for tracing the last RTT sample observed. The existing trace source TcpSocketBase::Rtt is still providing the smoothed RTT, although it had been incorrectly documented as providing the last RTT. (from 3.42 to 3.43)

IV. Source Code Detailed Analysis B. Difference of Ns-3 3.40 and 3.44

BBR related difference in source code level

tcp-bbr.cc:

- `m_rtProp` in 3.40 is changed to `m_minRtt` in 3.44. This seems to be just a text-level modification.
- After rate is calculated in `SetPacingRate()` as shown in Table III, the following modification is added.
  `rate *= (1.f – m_pacingMargin),`
  `m_pacingMargin` is set to 0.01
  This needs to be considered.
- One `if` sentence is modified in `UpdateBltBw()`.
  3.40: `if (rs.m_deliveryRate == 0)`
  3.44: `if (rs.m_delivered < 0 || rs/<_interval.IsZero())`
  This needs to be considered.
- `m_rtProp` (`m_minRtt`) is set to `m_lastRtt` in 3.40, but to `m_srtt` in 3.44. This is related to item 5) in the change history. This may need to be considered.

tcp-socket-base.cc

- One else if sentence is modified in `DupAck()`:
  3.40: `else if (m_txBuffer->IsLost(m_highRxAckMark + m_tcb->m_segmentSize)`
  3.44: `else if (m_txBuffer->IsLost(m_highRxAckMark)`
  This needs to be considered.
- Related to item 5) in the change history, `EstimateRtt()` is largely modified. This needs to be considered.

IV. Source Code Detailed Analysis B. Difference of Ns-3 3.40 and 3.44

- Use tcp-bbr.cc and tcp-socket-base.cc of version 3.44 in ns-3 version 3.44.
- Modify the followings.

tcp-socket-base.cc to be ported:

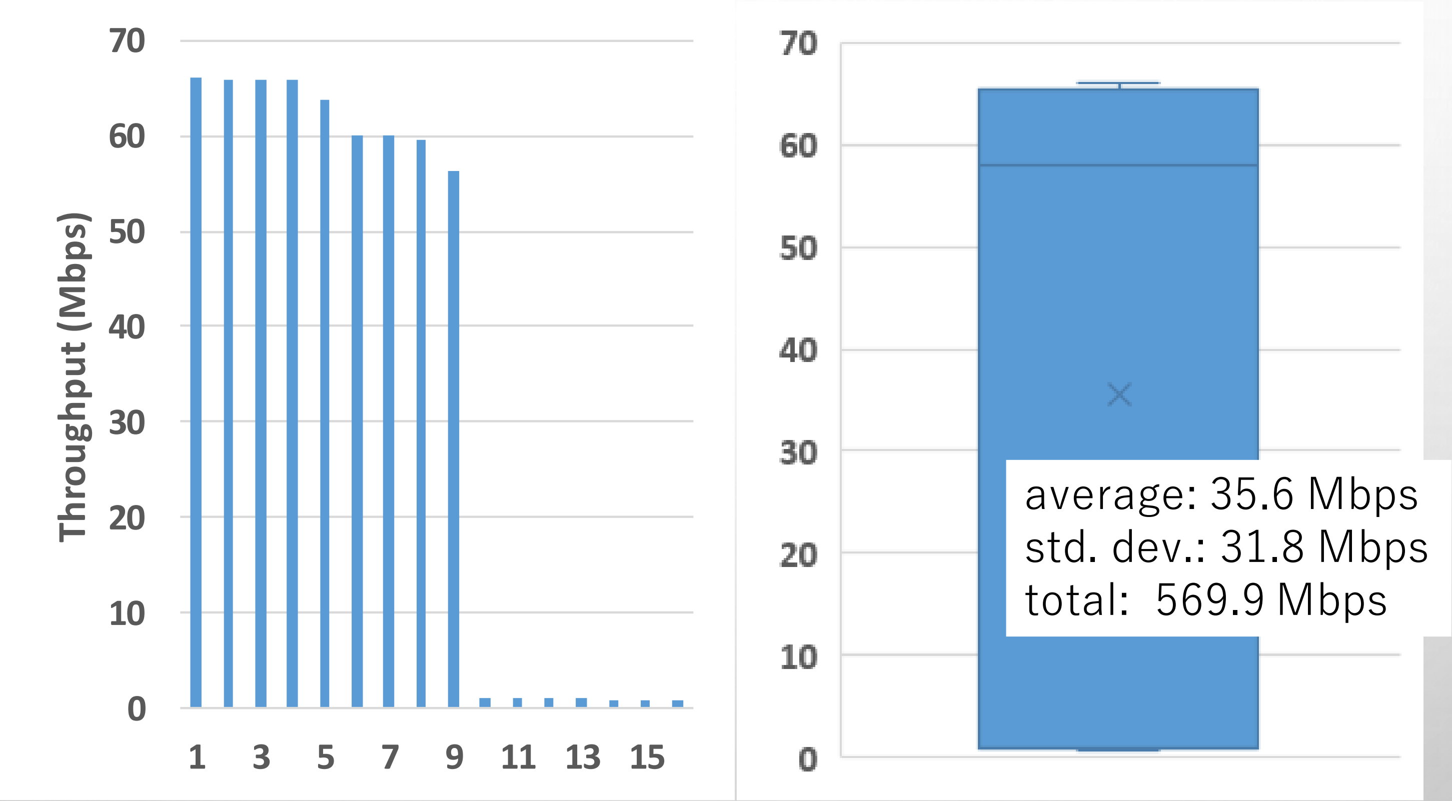
- Change `IsPending()` for `EventId` variables back to `IsRunning()`. This is related to item 3) in the change history.

tcp-socket-state.h in version 3.40:

- Define `TraceValue<Time> m_srtt` and `bool m_isCwndLimited`.

IV. Source Code Detailed Analysis B. Difference of Ns-3 3.40 and 3.44

- Result: A little improvement. More investigation required.



V. Conclusions

In this paper, we presented some results of TCP BBR performance evaluation where sixteen BBR flows share a 1 Gbps bottleneck link whose output router has a limited output buffer. We used 100, 500, 1,000, 2,000, and 5000 packets as the output buffer size. For, small buffer size, 100/500/1,000 packets, some flows provided high throughput, but the performance of the others were extremely low. Although the BBR intra-protocol unfairness is mentioned in [10], the degree of unfairness is much larger in this result. This experiment was done using ns-3 version 3.40. We also performed the same experiment using ns-3 version 3.44, which was the highest as of writing this paper. The result was completely different from that by version 3.40. All of the sixteen BBR flows provided high throughput, and the fairness was high.

We examined the BBR source codes of version 3.40 and 3.44. We analyzed the source codes in detail, and decided that there are not large differences between them. Next, we ported the version 3.44 BBR source code into ns-3 version 3.40, and conducted the same experiment using 100 packet output buffer. The result was similar with that in original version 3.40 case. This means that the difference between version 3.40 and 3.44 does not depend on the TCP related source code.

Although this paper could not point out the reason for the performance difference, we showed the details on TCP BBR behaviors over ns-3 network simulator. When using ns-3 for network performance evaluation, much care needs to be paid. We are planning to examine the ns-3 TCP BBR source code in more detail.