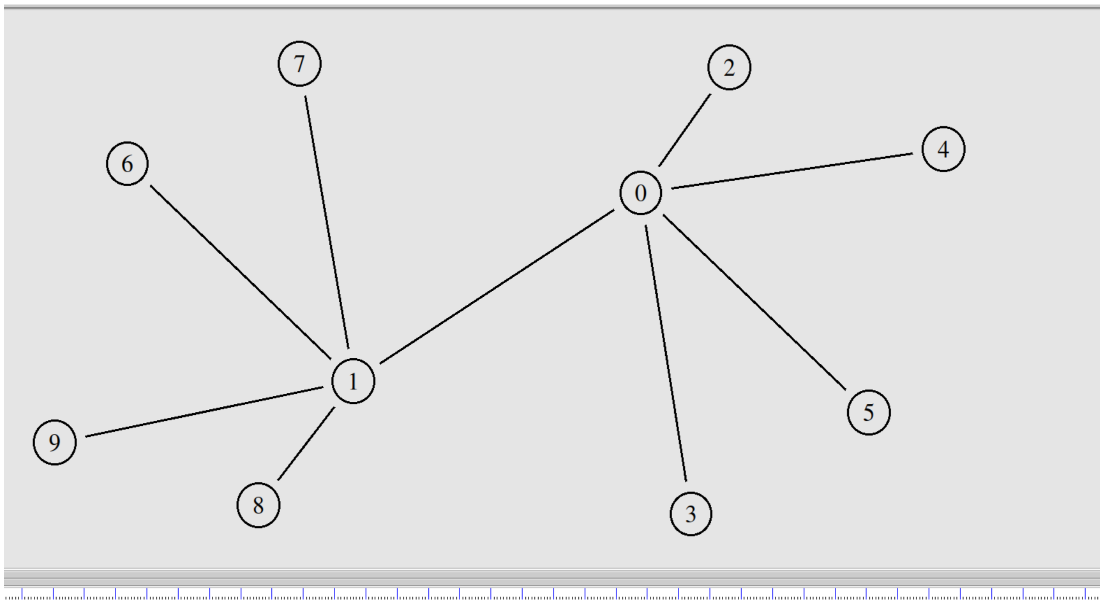
Report ON NS-2 PROJECT

SUBMITTED BY
MOHAMMAD ABRARUL HASANAT
(1805023)

WIRED

TOPOLOGY:

1.We have divided the area size into 2 parts (divide the x axis)
2.keep all the source nodes in the left part and all the destination nodes in the right part
3.create a node in the left part (determined source node)
4.create a node in the right part (determined destination node)
5. After that we have created a bridge



Parameters under variation:

1.Queue Size
2.Number of Nodes
3.Number of Flows
4.Number of Packets
5.Coverage Area(in Wireless)

While varying other parameters the default value is for these are kept at (5,40,20,200) respectively

Algorithm:

```
1. select the value of λmin =1 and  λmax = 3
2. select he value of β1 = 0.95 and β2 = 0.90
3. select $cwnd = 2$
 Event on Ack:
    calculate gapcurrent as in Equation (3)
    calculate gaptotal as in Equation (2)
    calculate λ as in Equation (1)
    $\alpha = \lambda/cwnd$
    $cwnd = cwnd + \alpha$
 Even on loss
    $cwnd_{loss} = cwnd$
     if ($cwnd < ssthread$):
         $cwnd = cwnd * \beta1$
     else:
         $cwnd = cwnd * \beta2$
    $cwnd_{degraded} = cwnd$
     $ssthreash = cwnd - 1$
```

$$gap_{current} = max(cwnd_{loss} - cwnd, 1)$$

$$gep_{total} = max(cwnd_{loss} - cwnd_{degraged}, 1)$$

$$\lambda = \sqrt{max(\frac{\lambda_{max}.gap^2_{current}}{gap^2_{total}}, 1)}$$

MODIFICATION:

NS-2.35 => TCP => LINUX => TCP_NAIVE_RENO
COMPARED WITH LINUX IMPLEMENTATION TCP_CUBIC, TCP_HIGHSPEED AND TCP_COMPOUND

THE IMPLEMENTATION IS GIVEN BELOW : (github link for better view)

```
#define NS_PROTOCOL "tcp_naive_reno.c"

#include "ns-linux-c.h"
#include "ns-linux-util.h"

static int alpha = 1;
static int beta = 2;
module_param(alpha, int, 0644);
MODULE_PARM_DESC(alpha, "AI increment size of window (in unit of
pkt/round trip time)");
module_param(beta, int, 0644);
MODULE_PARM_DESC(beta, "MD decrement portion of window: every
loss the window is reduced by a proportion of 1/beta");
static int beta1_nom = 1;
static int beta1_denom = 10;
static int beta2_nom = 5;
static int beta2_denom = 100;
static int lambda_min = 1;
static int lampbda_max = 3;
static int cwnd_loss = -1;
static int cwnd_degraded = -1;
```

```
void tcp_naive_reno_cong_avoid(struct tcp_sock* tp, u32 ack, u32
rtt, u32 in_flight, int flag)
{
     ack = ack;
     rtt = rtt;
     in_flight = in_flight;
     flag = flag;
     // calculating gapCurrent
     int gap_current = max(cwnd_loss - tp->snd_cwnd, 1U);
     gap_current = gap_current * gap_current;
     int gap_total = max(cwnd_loss - cwnd_degraded, 1U);
     gap_total = gap_total * gap_total;
     int lambda = (gap_current * lampbda_max) / gap_total;
     if (lambda < lambda_min) lambda = lambda_min;

     int root = 1;
     int i;
     for ( i = 1; i * i <= lambda; ++i) {
          root = i;
     }


     lambda = root;
     int avg = (gap_current + gap_total) / 2;
     if (lambda - avg >= lampbda_max && avg > 0 ) {
          lambda = lambda - avg;
          avg = (avg * 95) / 100;
     }


     alpha = max(1U, lambda / tp->snd_cwnd);
```

```
    if (tp->snd_cwnd < tp->snd_ssthresh) {
        tcp_slow_start(tp);
    }
    else {
        if (tp->snd_cwnd_cnt >= tp->snd_cwnd) {
            tp->snd_cwnd += alpha;
            tp->snd_cwnd_cnt = 0;
            if (tp->snd_cwnd > tp->snd_cwnd_clamp)
                tp->snd_cwnd = tp->snd_cwnd_clamp;
        }
        else {
            tp->snd_cwnd_cnt += alpha;
        }
    }
}

u32 tcp_naive_reno_ssthresh(struct tcp_sock* tp)
{
    //int reduction = tp->snd_cwnd / beta;
    //return max(tp->snd_cwnd - reduction, 2U);
    cwnd_loss = tp->snd_cwnd;
    int reduction = 0;
    if (tp->snd_cwnd < tp->snd_ssthresh) {
        if (beta1_denom != 0)
         reduction = (tp->snd_cwnd * beta1_nom) /beta1_denom;
    }
    else {
        if (beta2_denom != 0)
            reduction = (tp->snd_cwnd * beta2_nom) / beta2_denom;

    }
    cwnd_degraded = tp->snd_cwnd - reduction;
    tp->snd_cwnd = cwnd_degraded;
    return max(cwnd_degraded - 1, 2U);
}
```

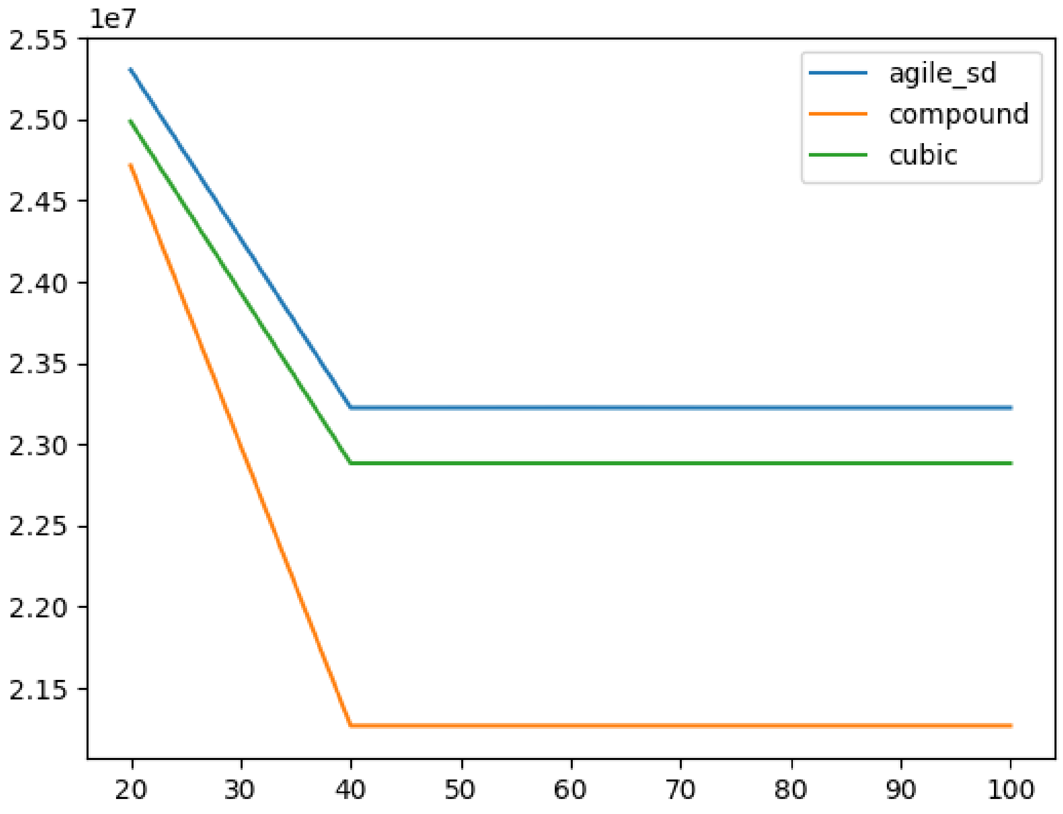
```
u32 tcp_naive_reno_min_cwnd(const struct tcp_sock* tp)
{
     return tp->snd_ssthresh;
}

static struct tcp_congestion_ops tcp_naive_reno = {
          .name = "agileSD",
          .ssthresh = tcp_naive_reno_ssthresh,
          .cong_avoid = tcp_naive_reno_cong_avoid,
          .min_cwnd = tcp_naive_reno_min_cwnd
};

int tcp_naive_reno_register(void)
{
     tcp_register_congestion_control(&tcp_naive_reno);
     return 0;
}
module_init(tcp_naive_reno_register);
```

Plots

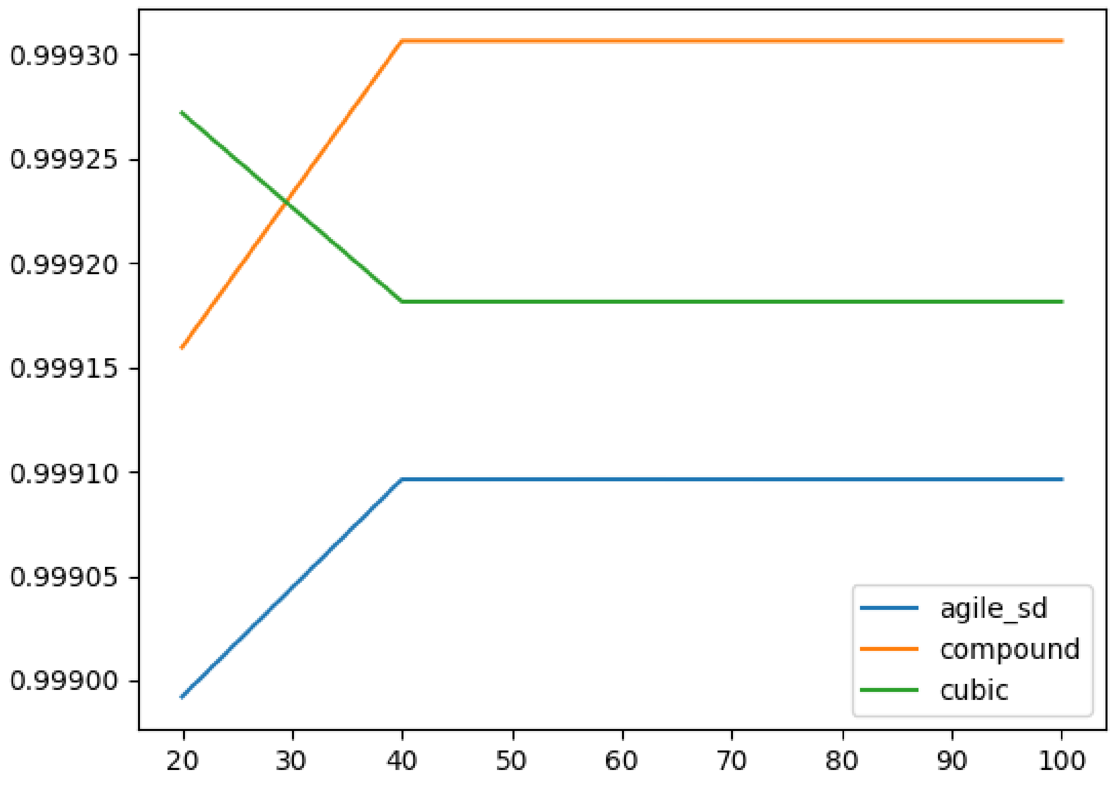
Wired.

1.Varying Number of Nodes

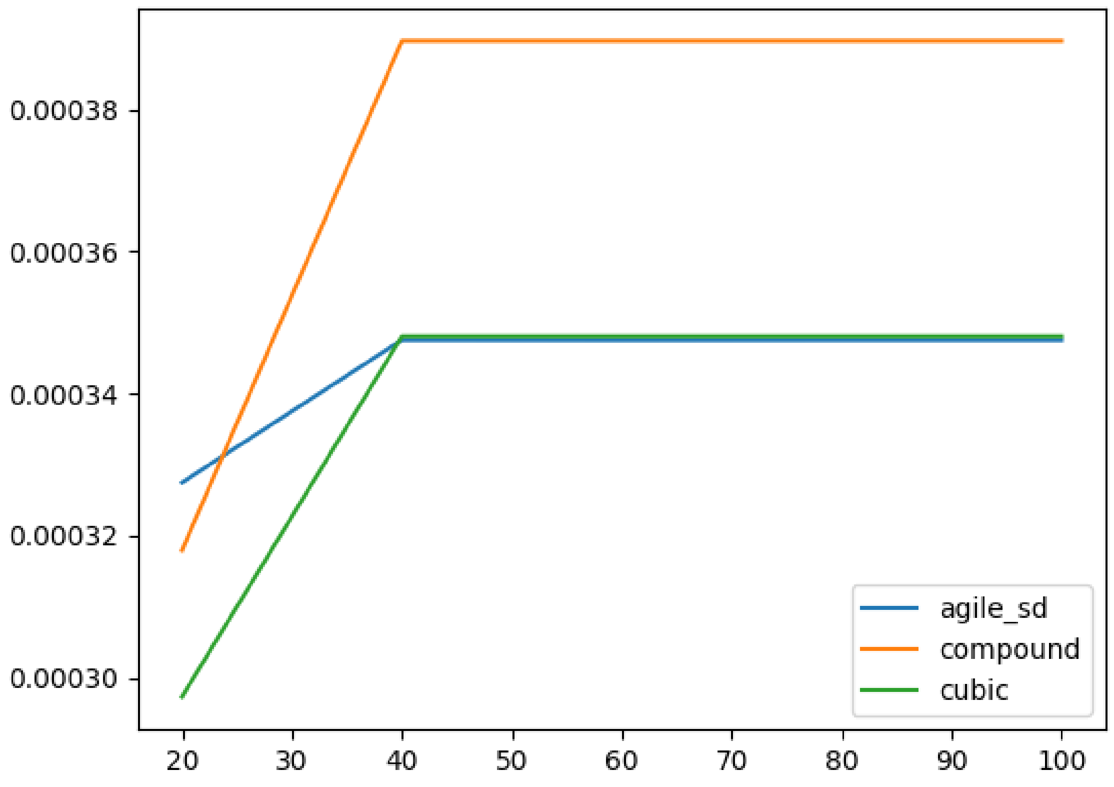
Throughput



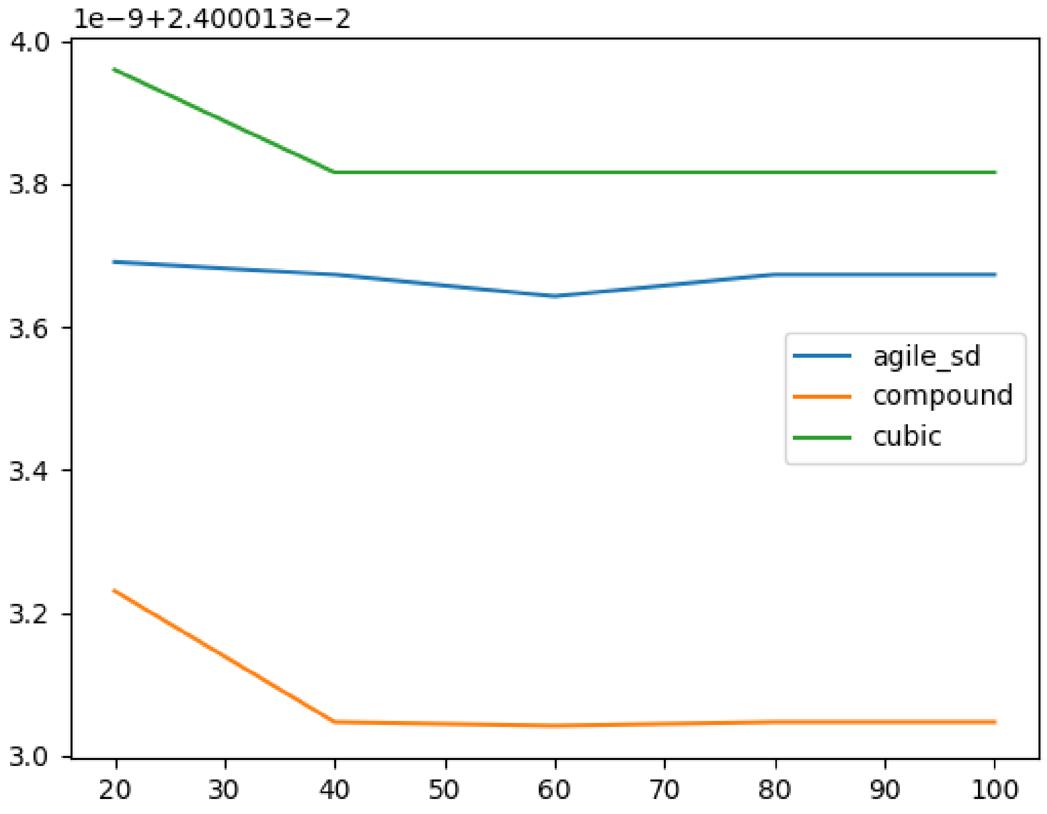
Delivery Ratio:



Drop Ratio:

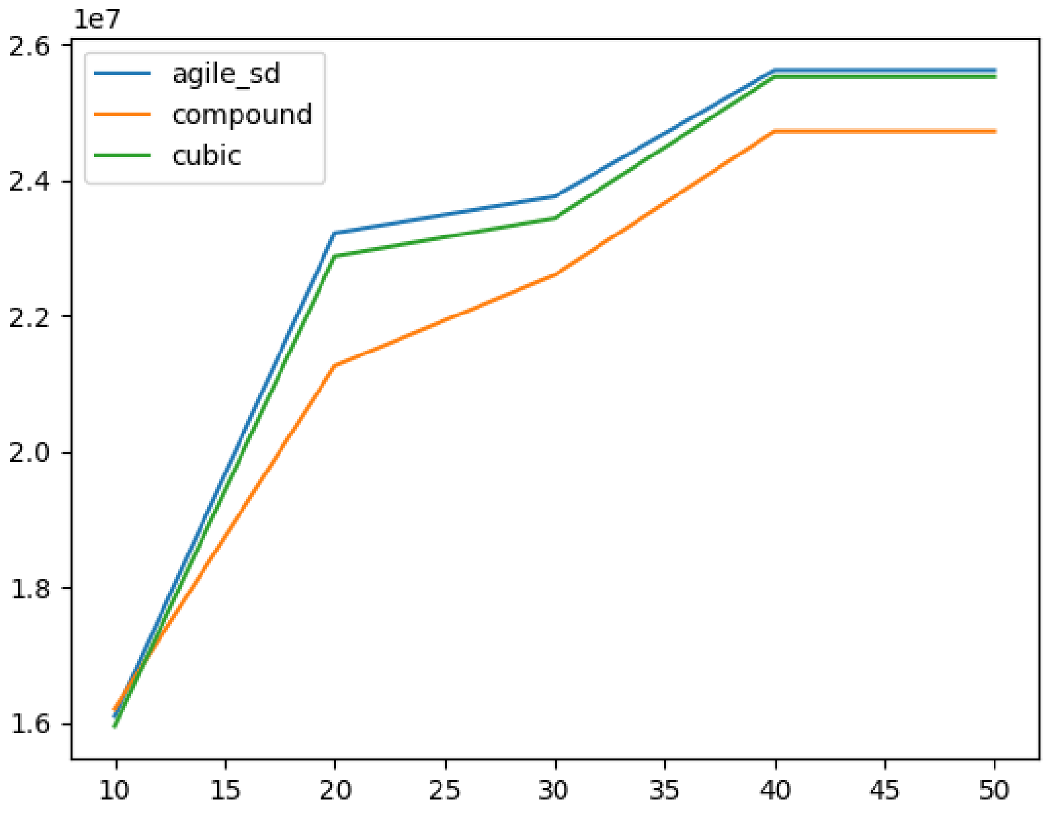

0.00038
0.00036
0.00034
0.00032
0.00030
20
30
40
50
60
70
80
90
100
agile_sd
compound
cubic

Delay

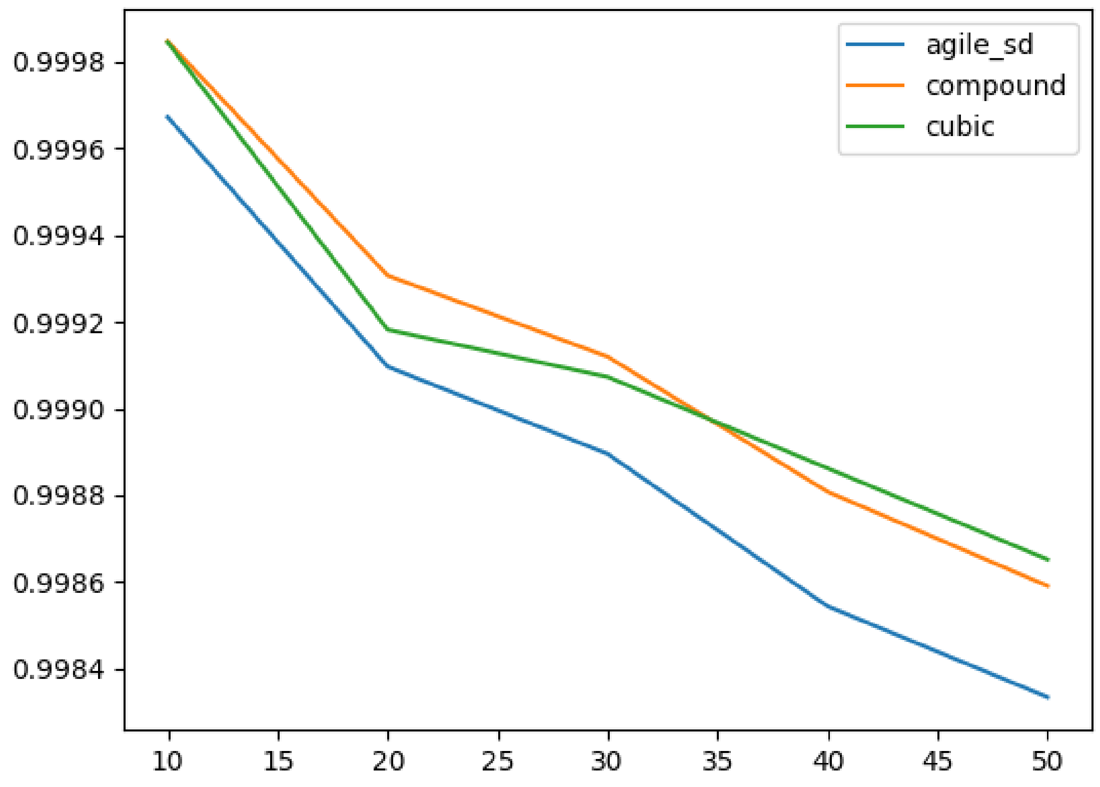

1e−9+2.400013e−2
4.0
3.8
3.6
3.4
3.2
3.0
20
30
40
50
60
70
80
90
100
agile_sd
compound
cubic

2.Varying Number of Flows

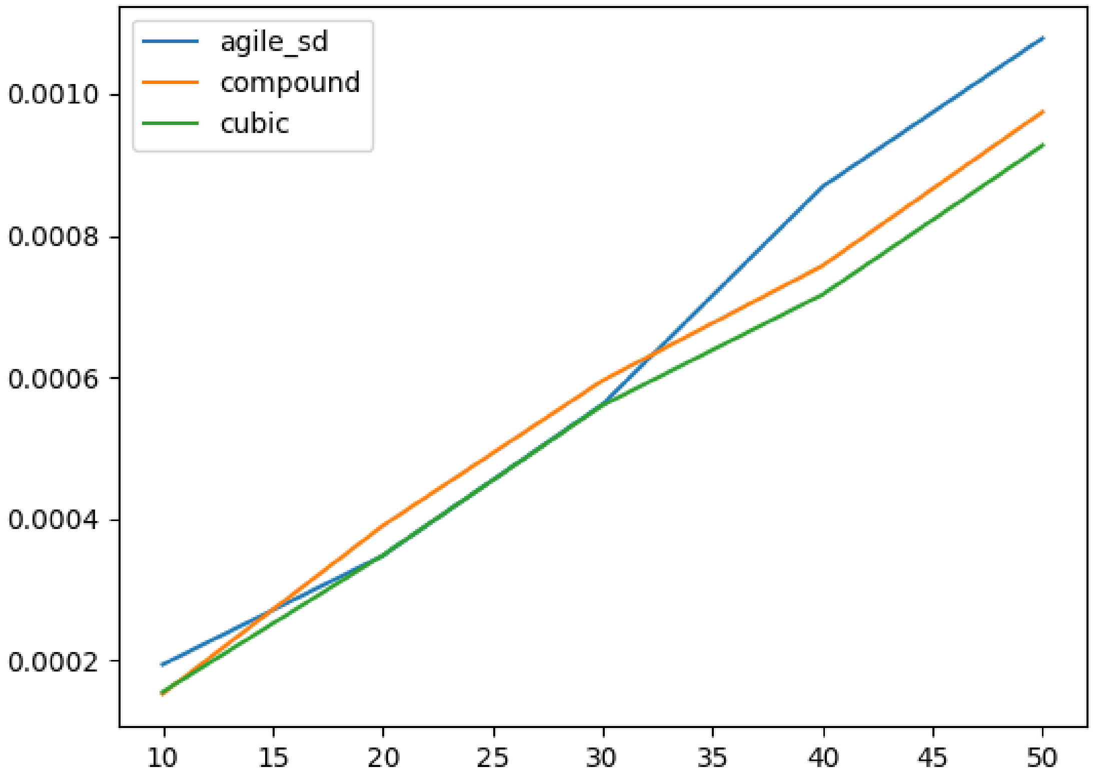
Throughput:



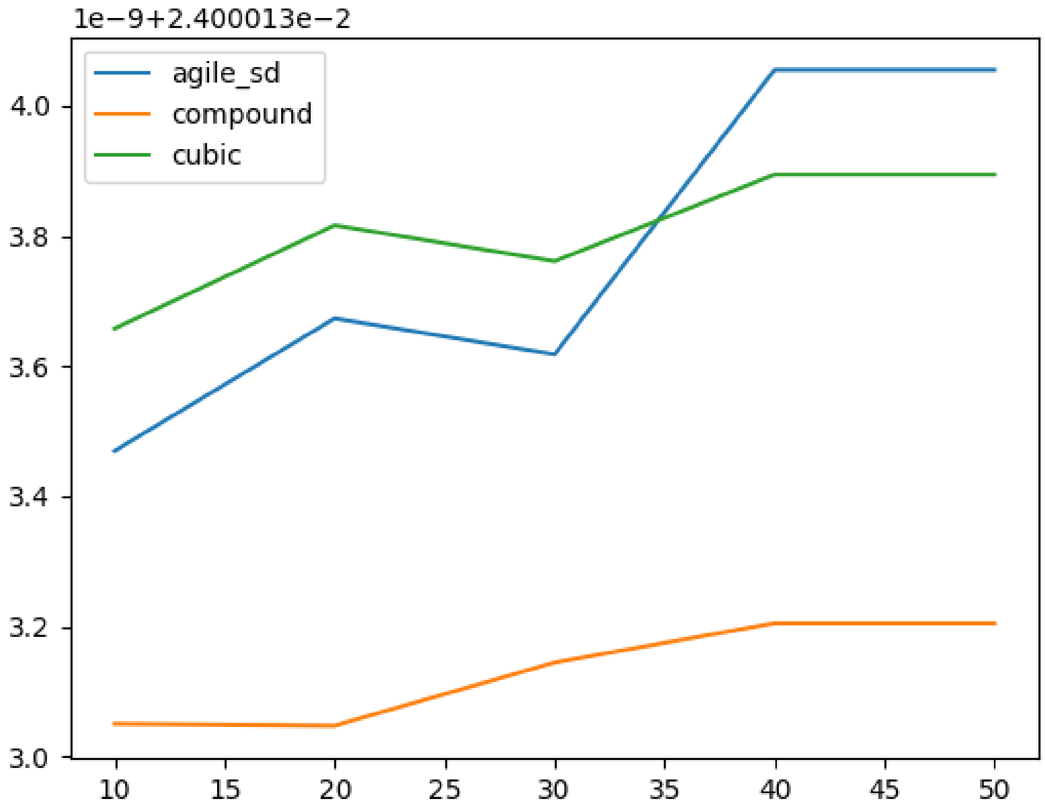
Delivery Ratio:



Drop ratio


agile_sd
compound
cubic
0.0010
0.0008
0.0006
0.0004
0.0002
10
15
20
25
30
35
40
45
50

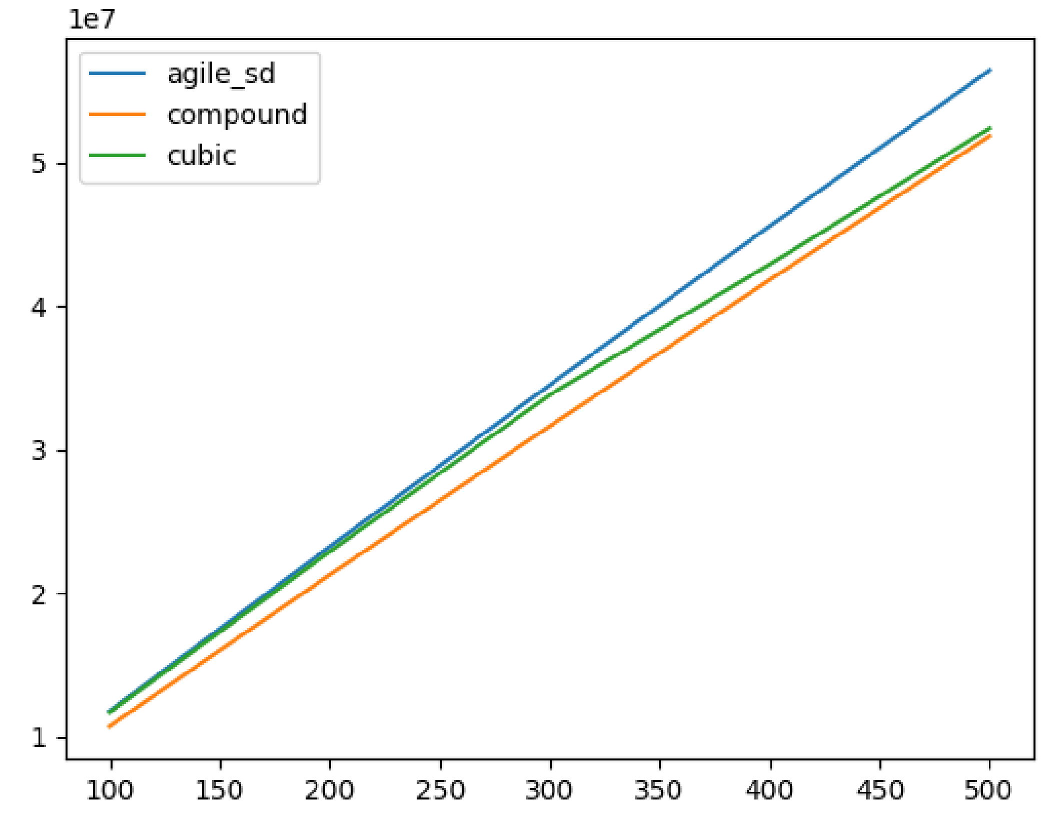
Delay:



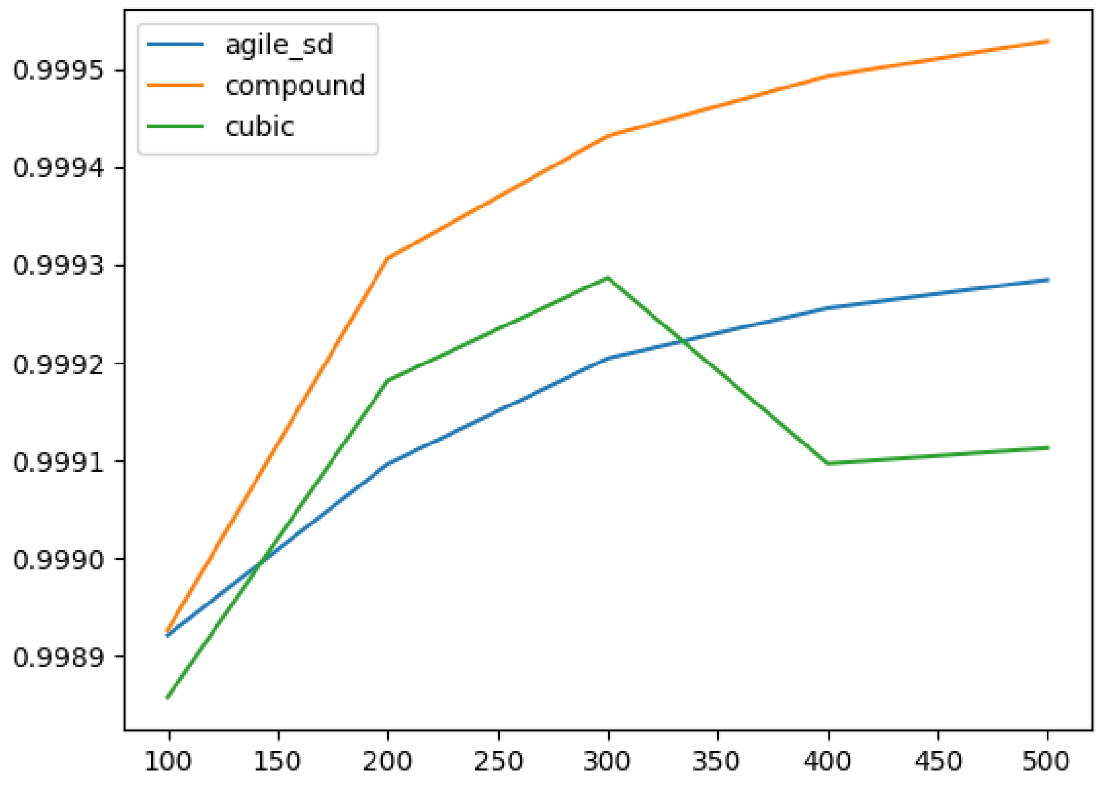
1e−9+2.400013e−2
agile_sd
compound
cubic
4.0
3.8
3.6
3.4
3.2
3.0
10
15
20
25
30
35
40
45
50

3.Varying Number packet per sec

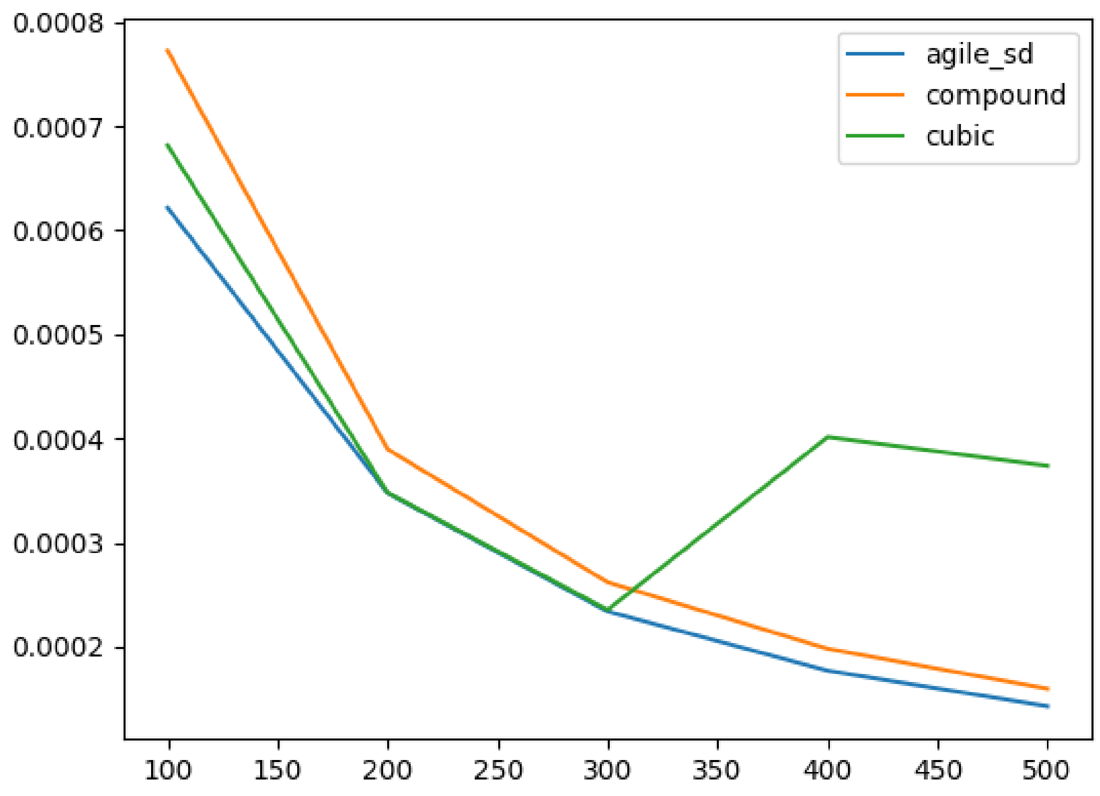
Throughput



Delivery Ratio

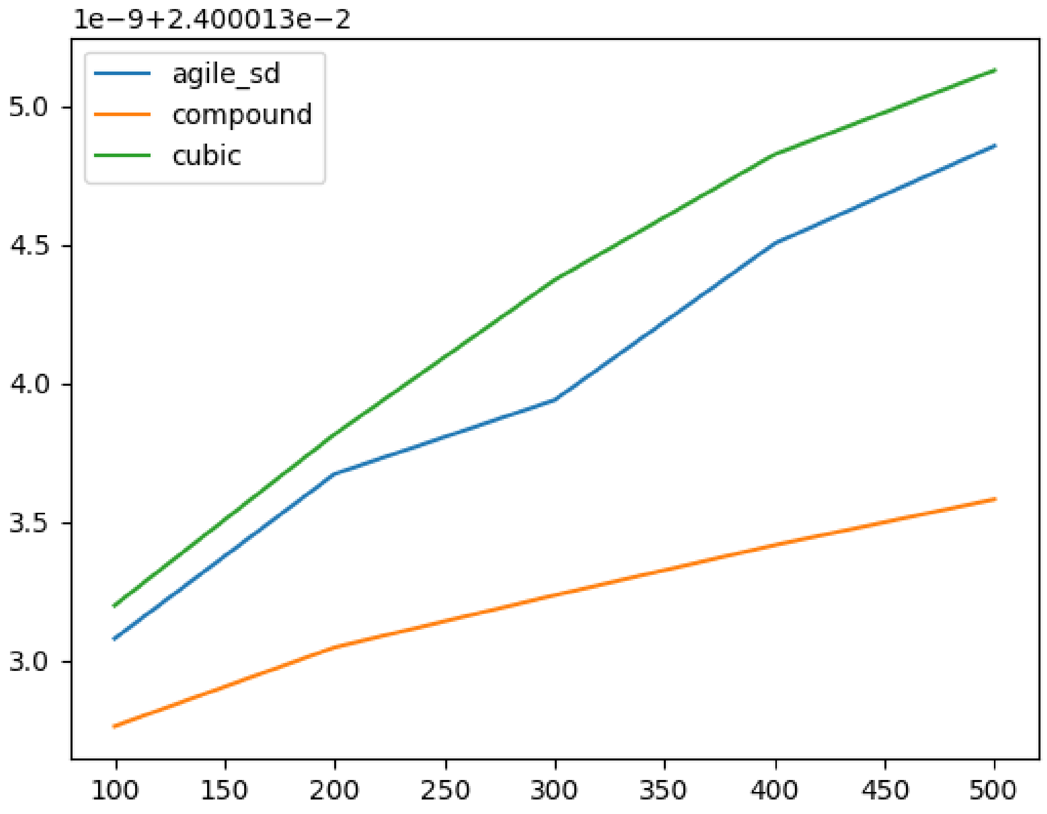


Drop Ratio



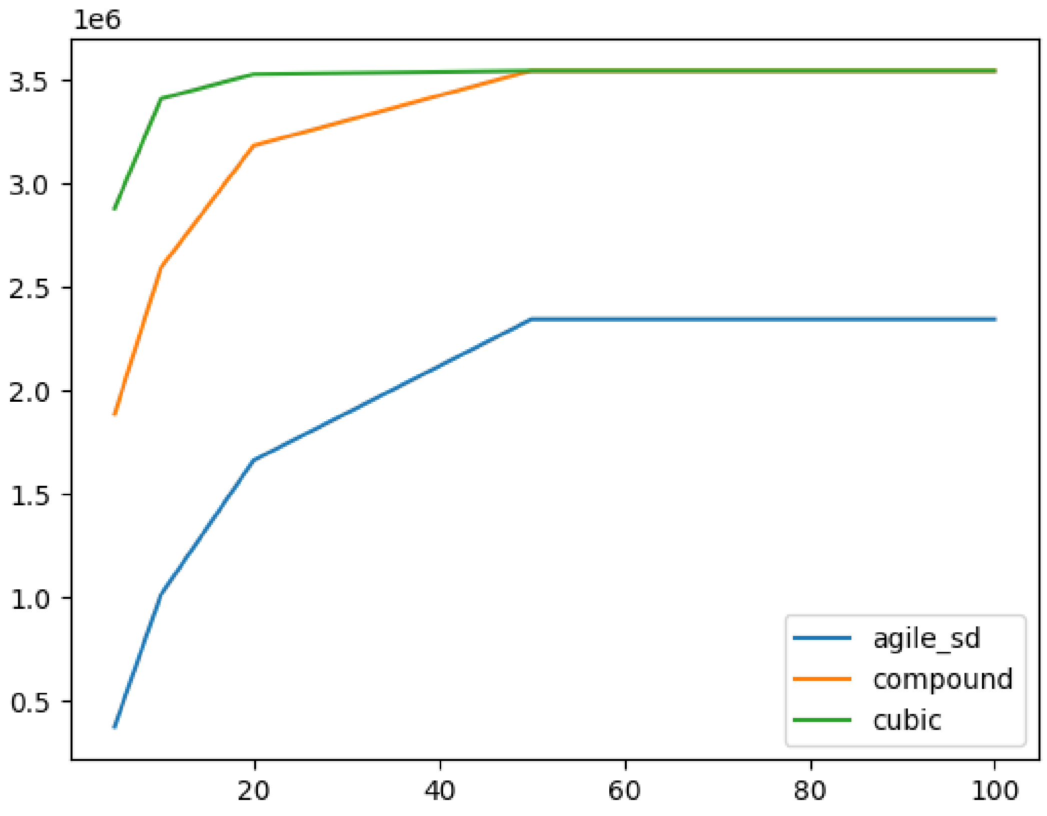
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
100
150
200
250
300
350
400
450
500
agile_sd
compound
cubic

Delay

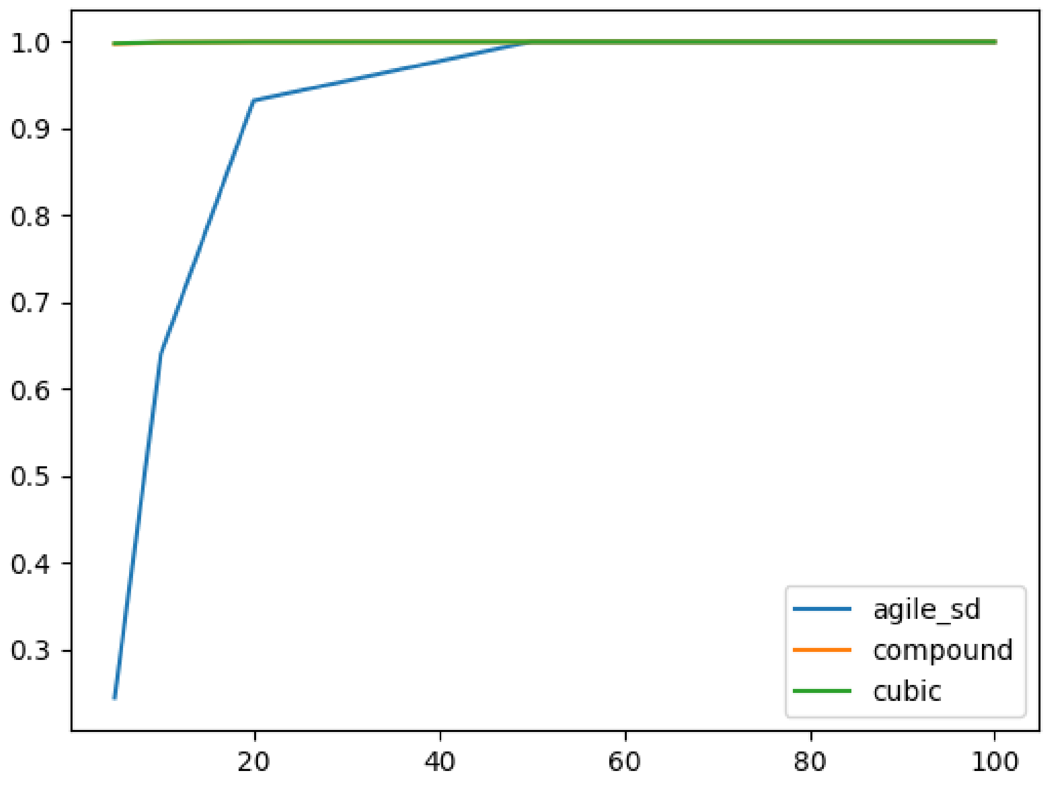

1e−9+2.400013e−2
agile_sd
compound
cubic
5.0
4.5
4.0
3.5
3.0
100
150
200
250
300
350
400
450
500

4.Varying Queue Size

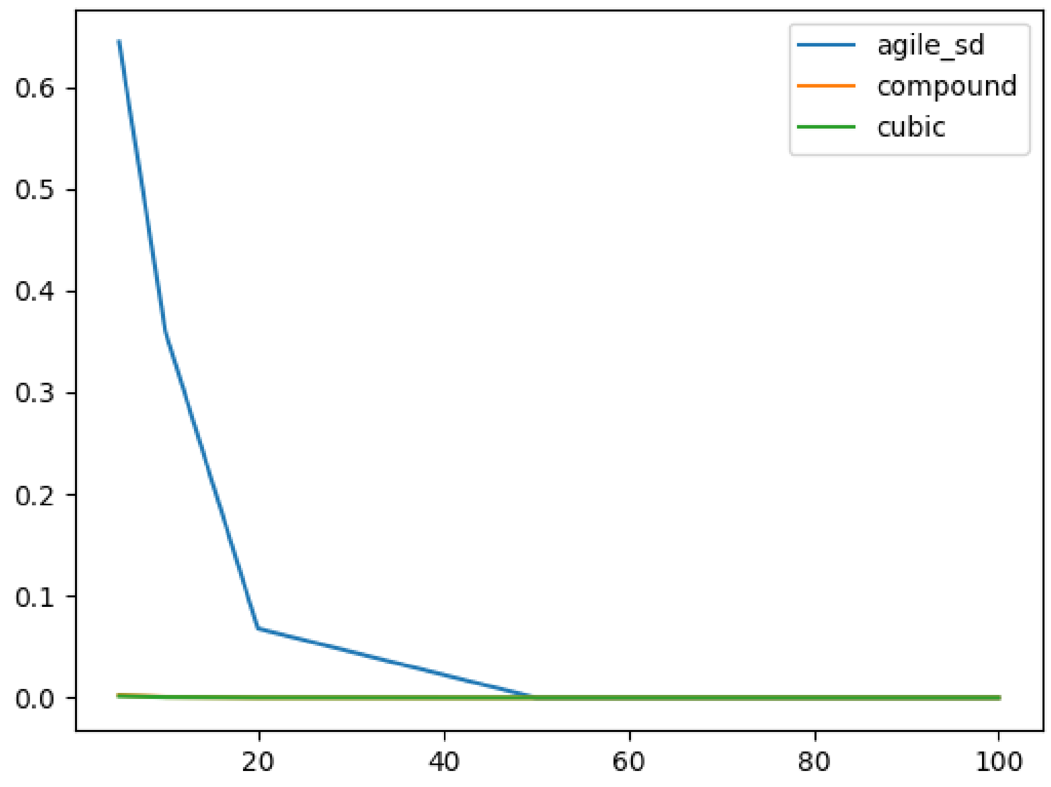
Throughput

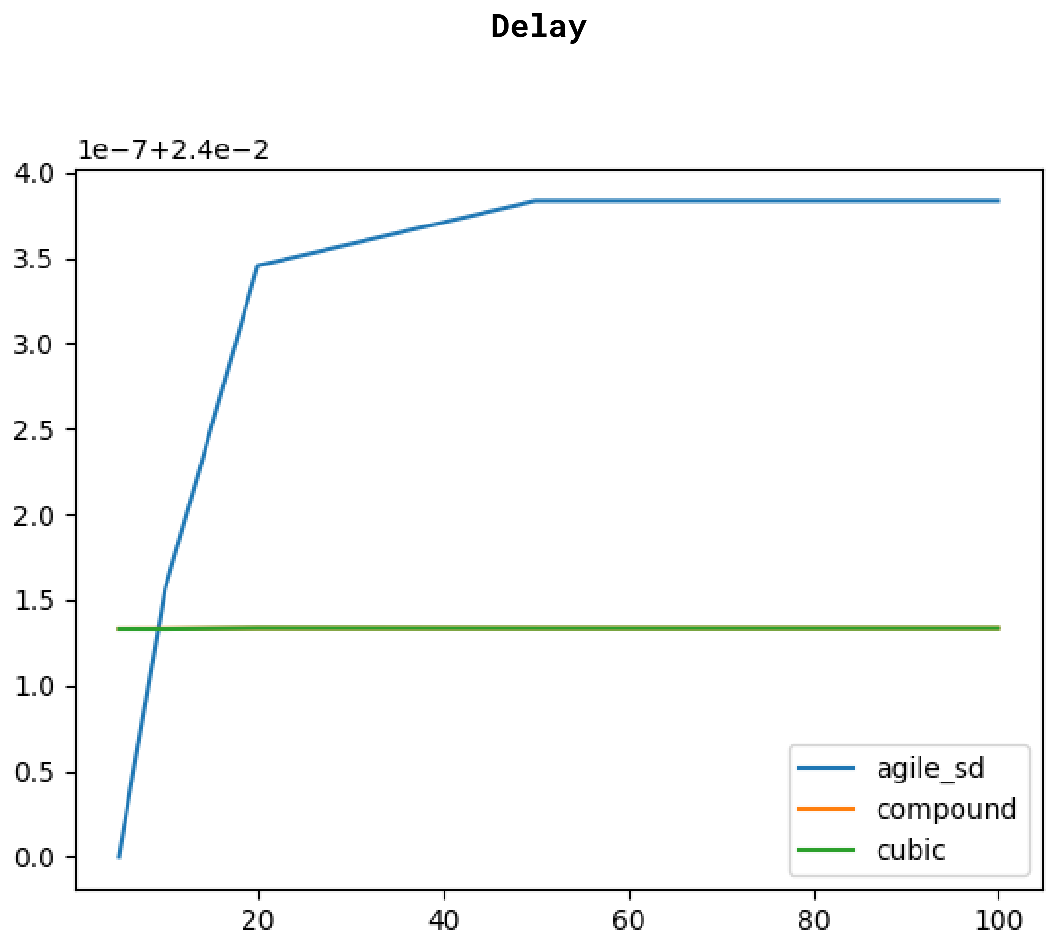


Delivery Ratio



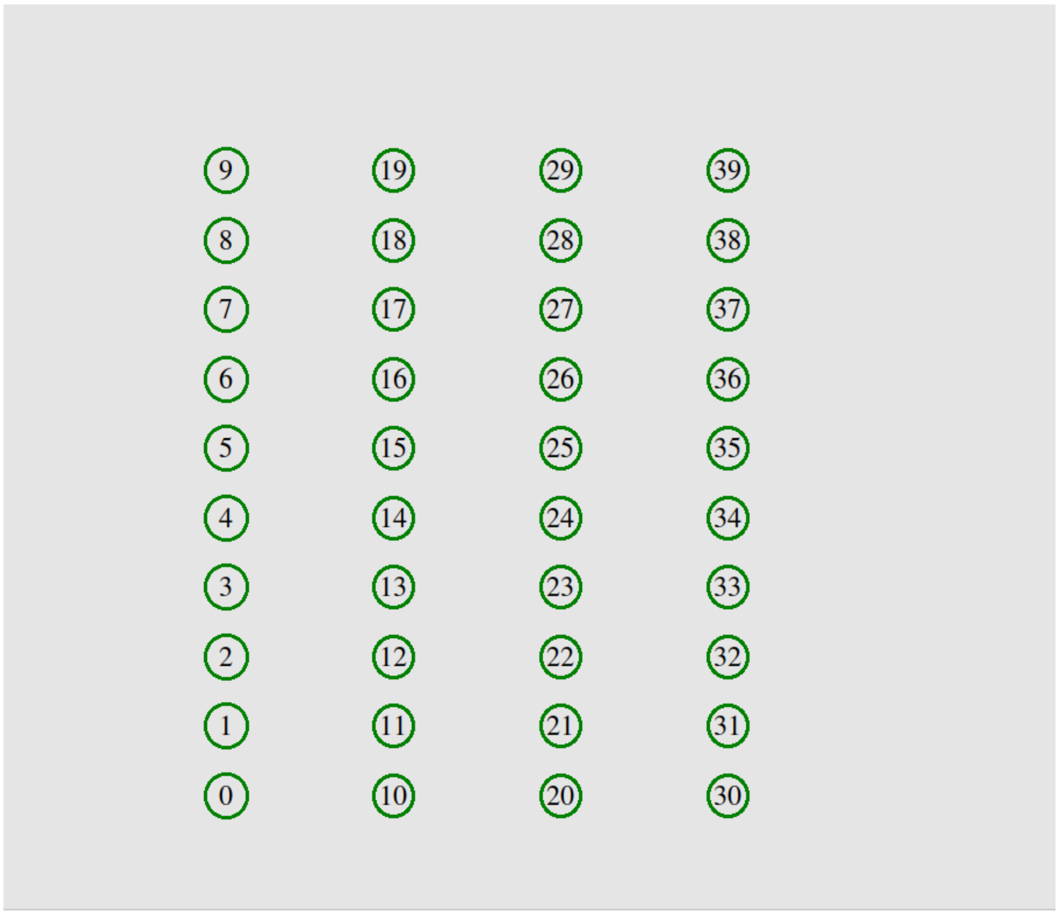
Drop Ratio




Delay
1e−7+2.4e−2
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
20
40
60
80
100
agile_sd
compound
cubic

Wireless (topology)

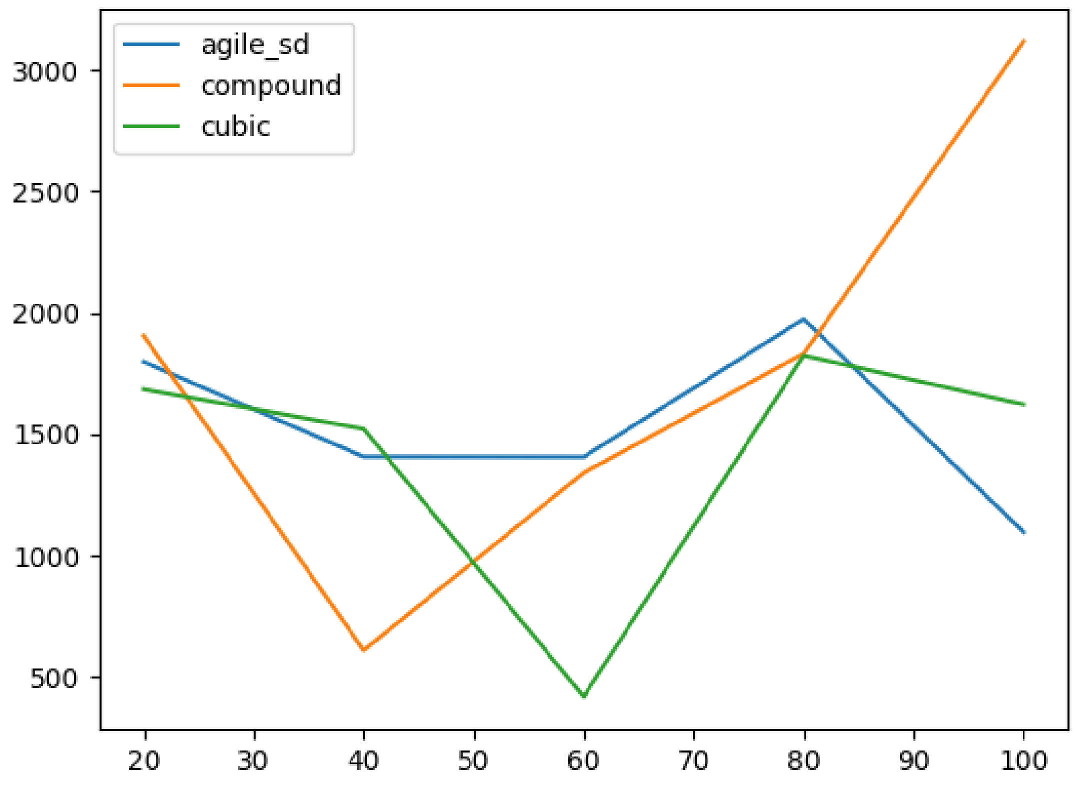
Specification
Wireless MAC type: Wireless 802.15.4
Routing Protocol : DSDV
Agent : TCP Linux
Application: FTP
Node Positioning : Grid
Flow : Random source and destination


9
19
29
39
8
18
28
38
7
17
27
37
6
16
26
36
5
15
25
35
4
14
24
34
3
13
23
33
2
12
22
32
1
11
21
31
0
10
20
30

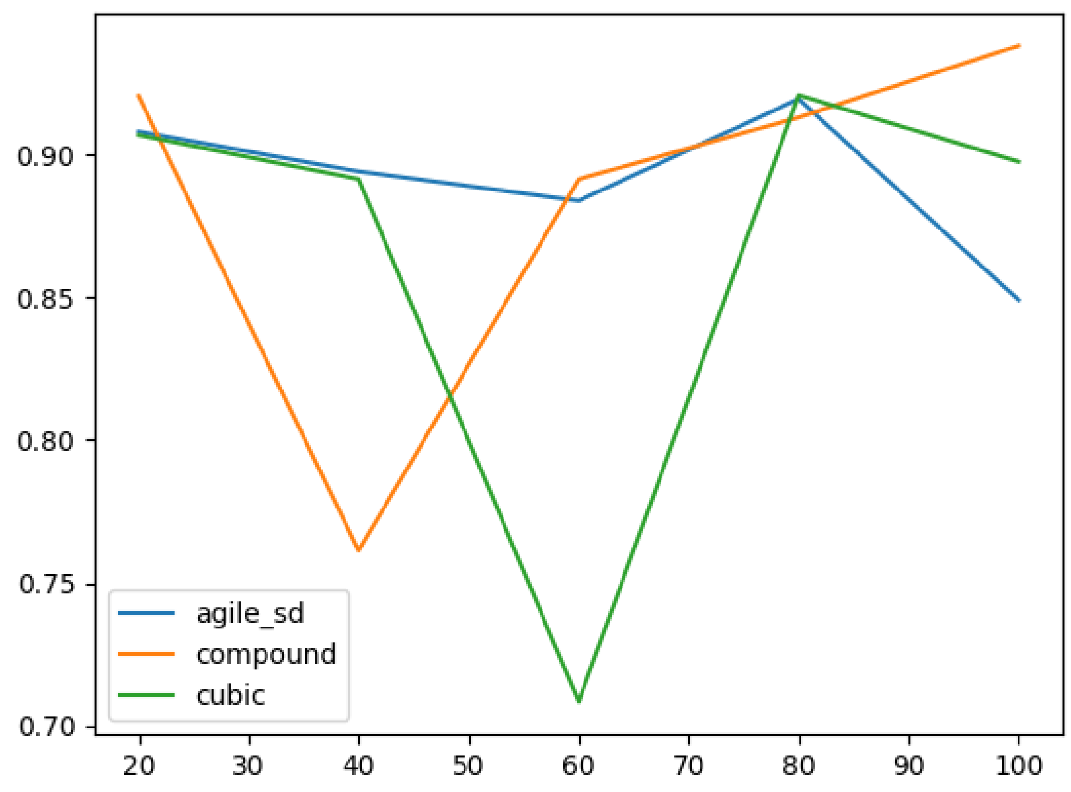
Plots

1.Varying Number of Nodes

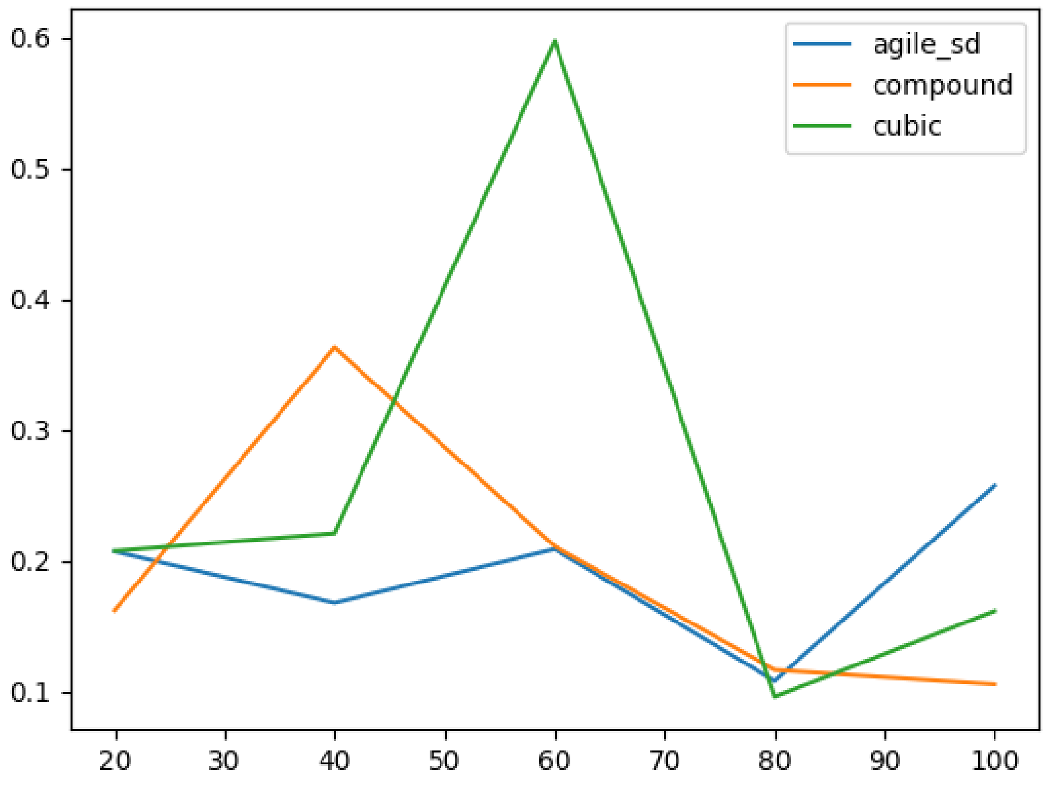
Throughput



Delivery Ratio

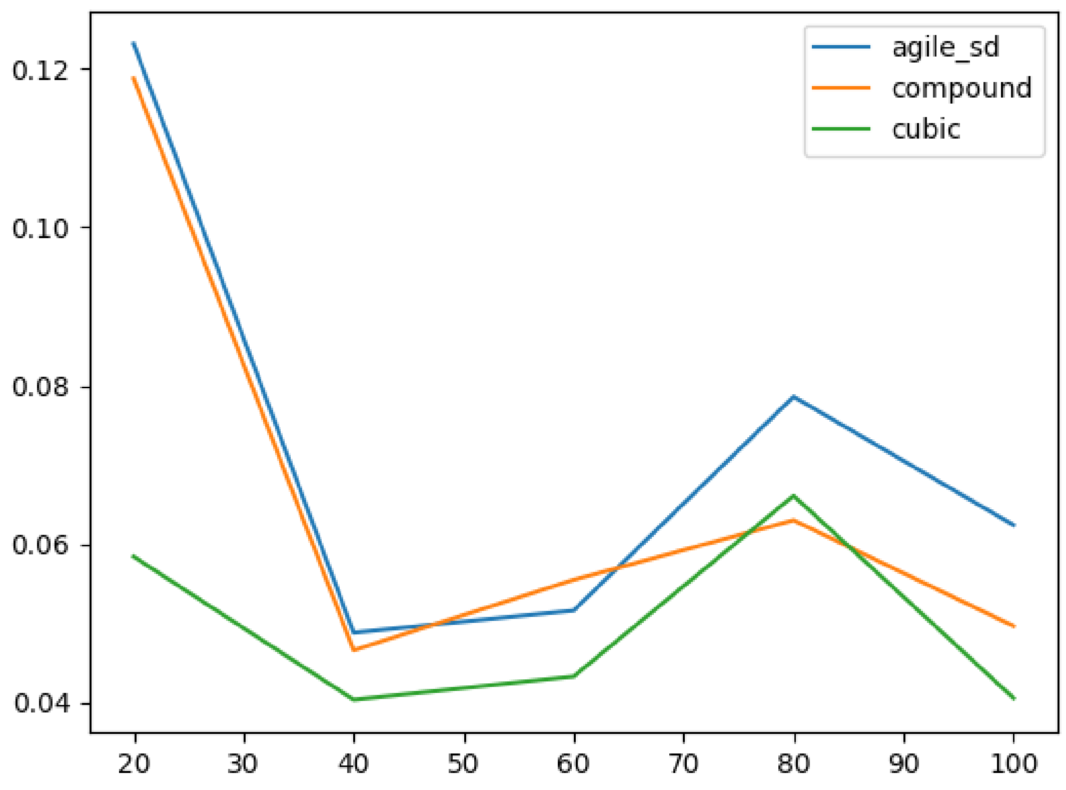


Drop Ratio



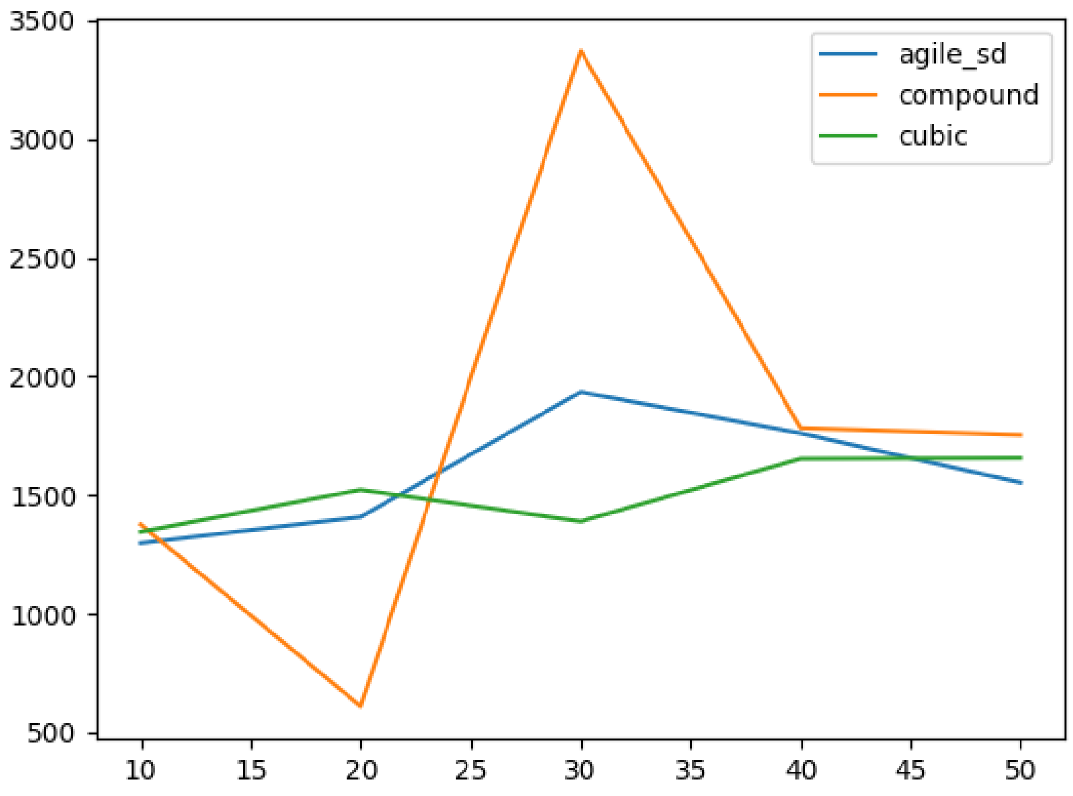
0.6
0.5
0.4
0.3
0.2
0.1
20
30
40
50
60
70
80
90
100
agile_sd
compound
cubic

Average Delay

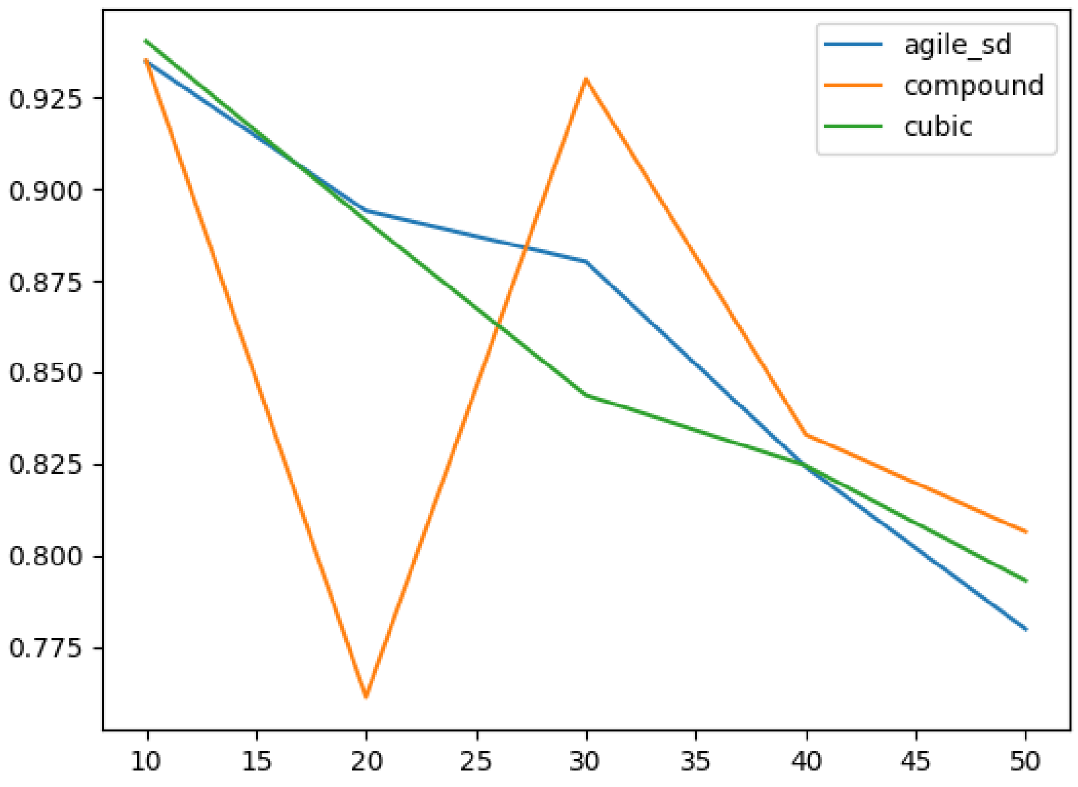


2.Varying Number of Flows

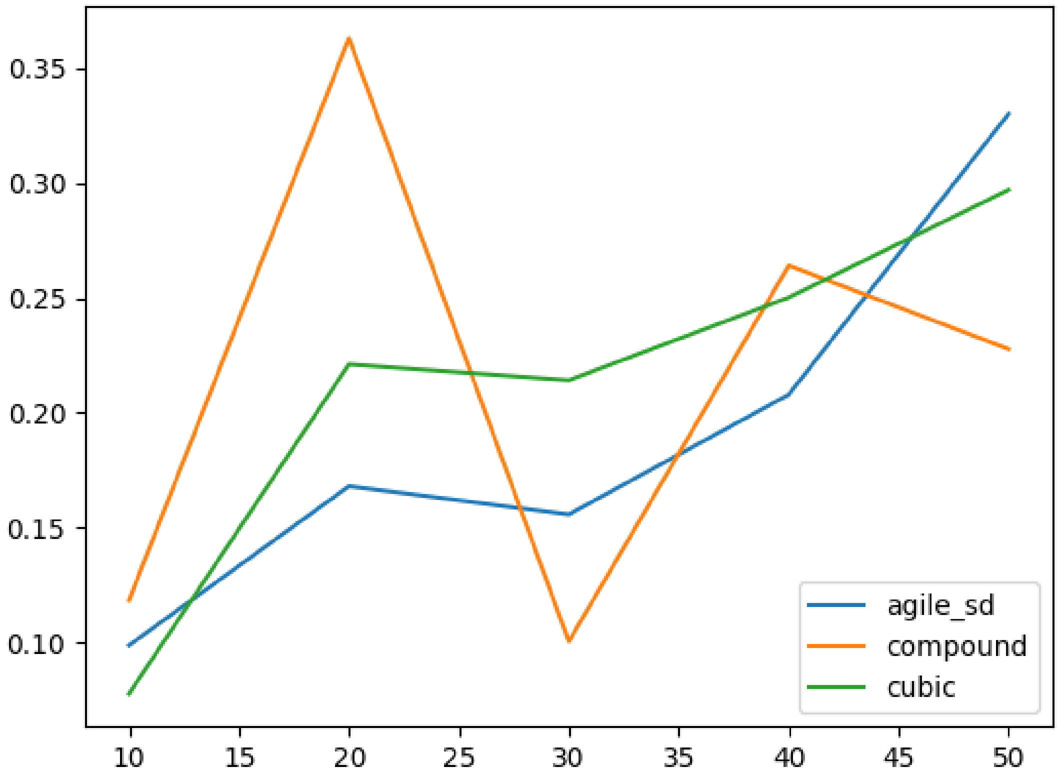
Throughput



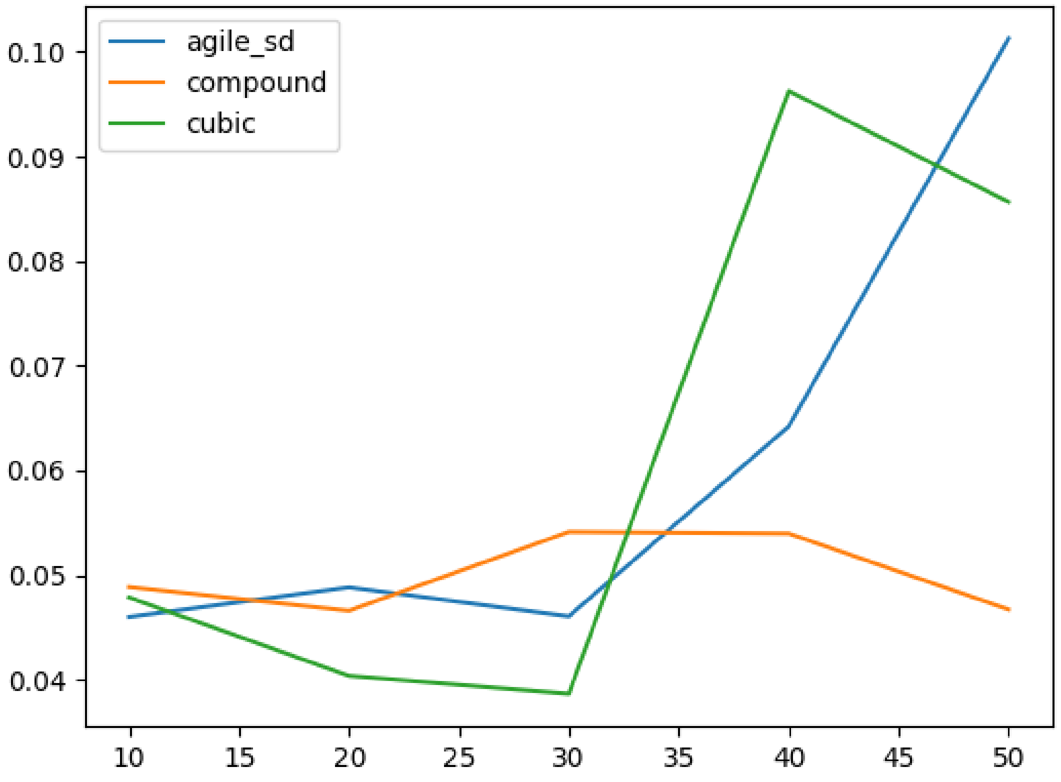
Delivery Ratio



Drop Ratio

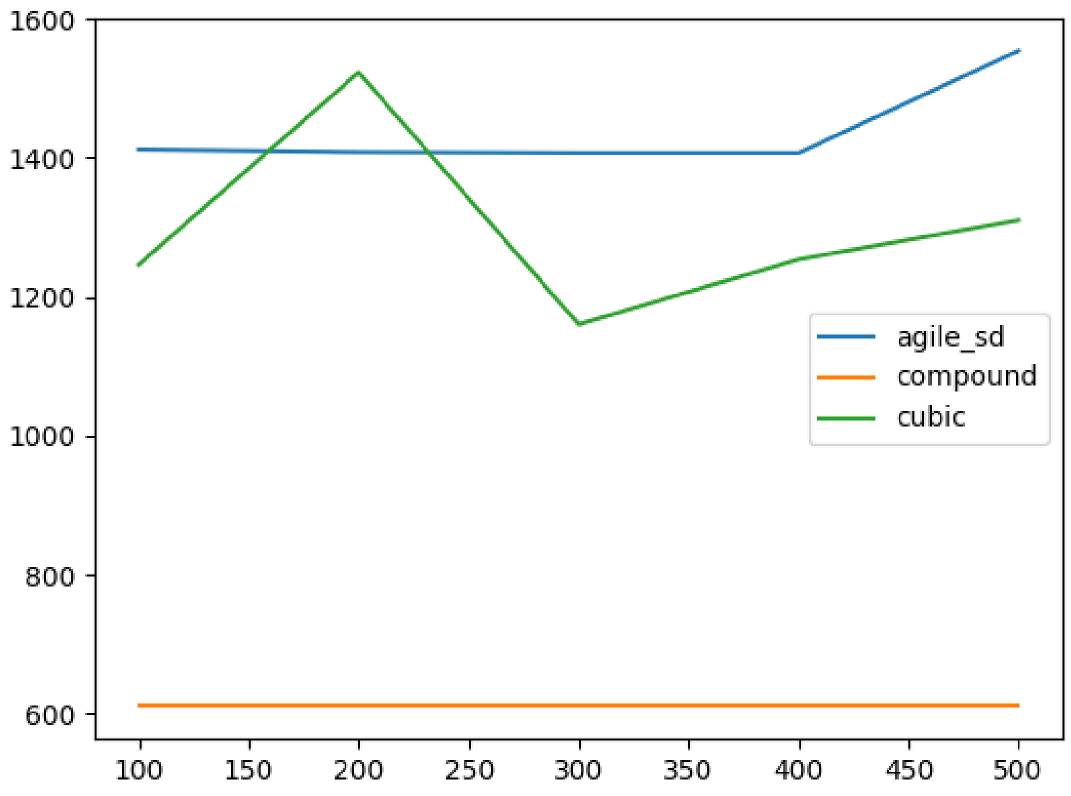

0.35
0.30
0.25
0.20
0.15
0.10
10
15
20
25
30
35
40
45
50
agile_sd
compound
cubic

Average Delay

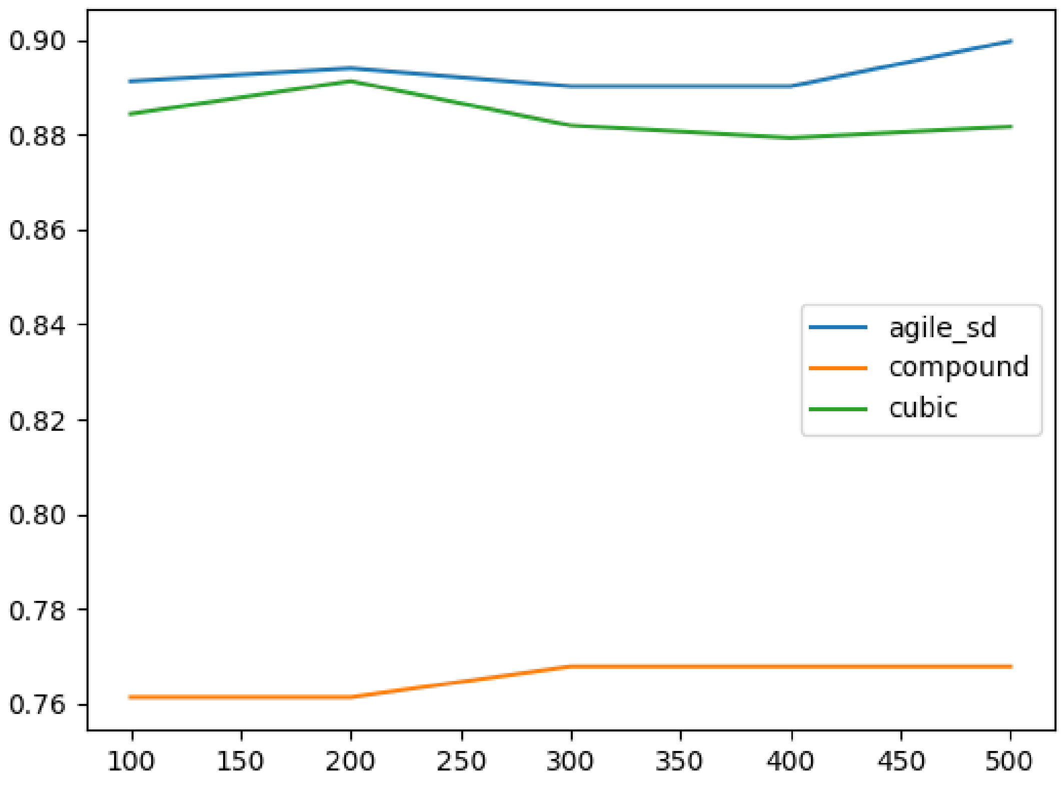


3.Varying Number of packet/sec

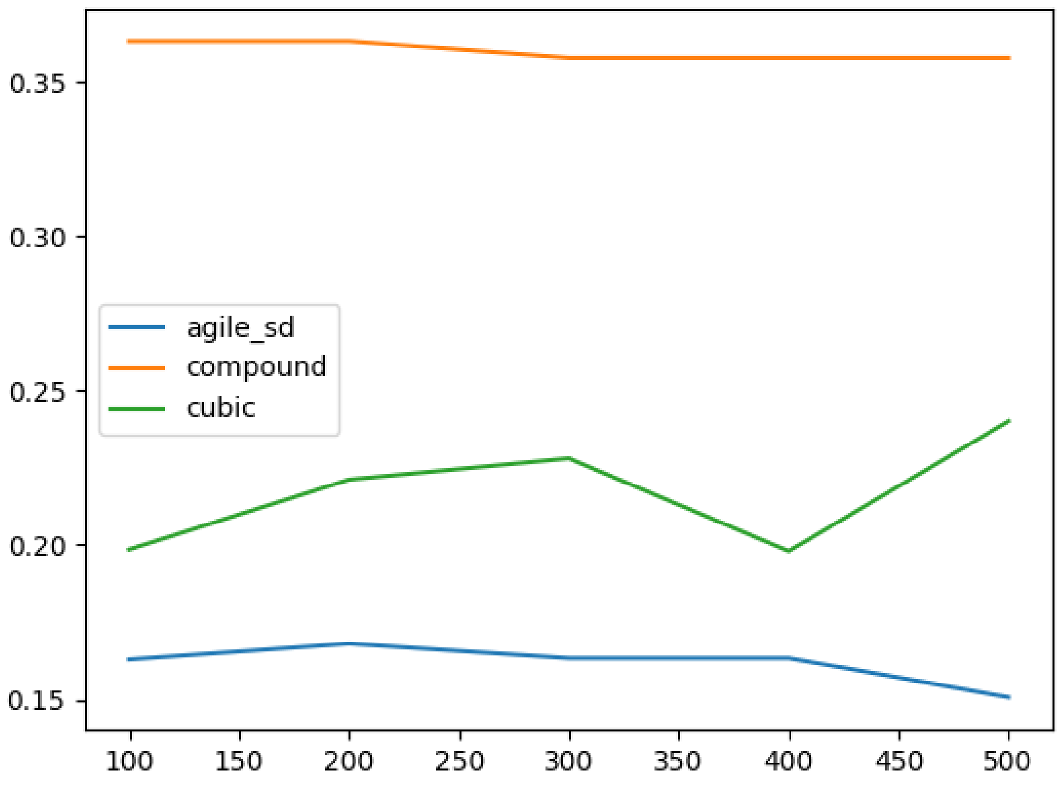
Throughput



Delivery Ratio

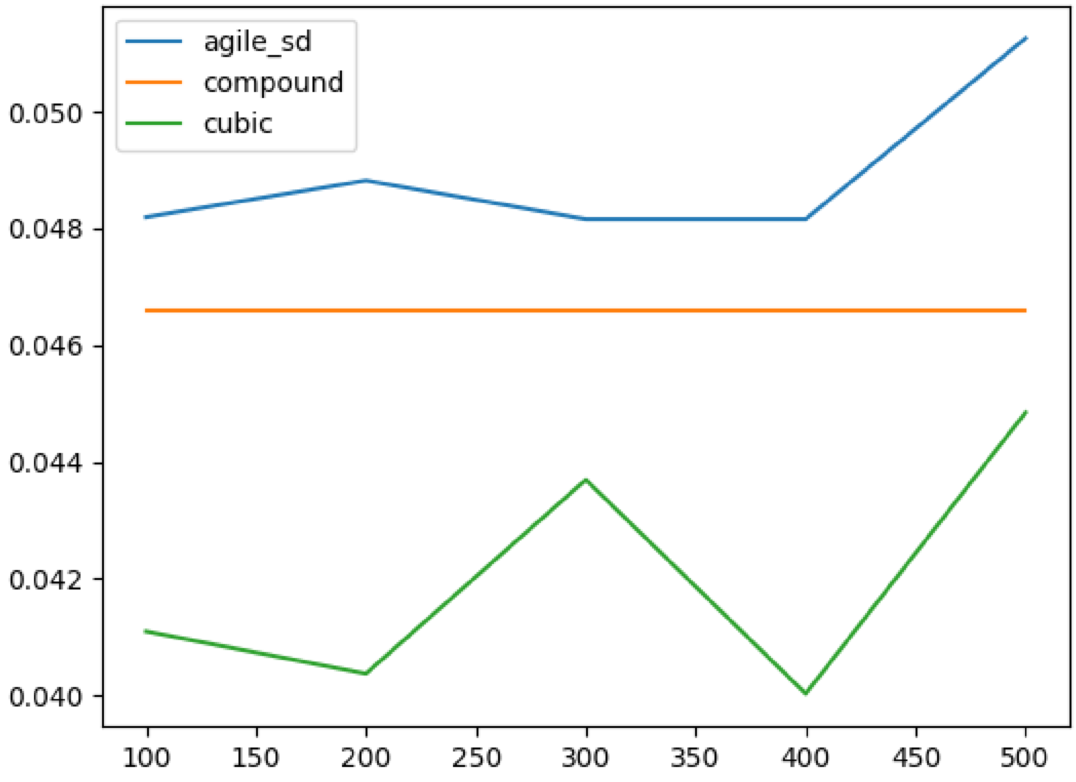


Drop Ratio



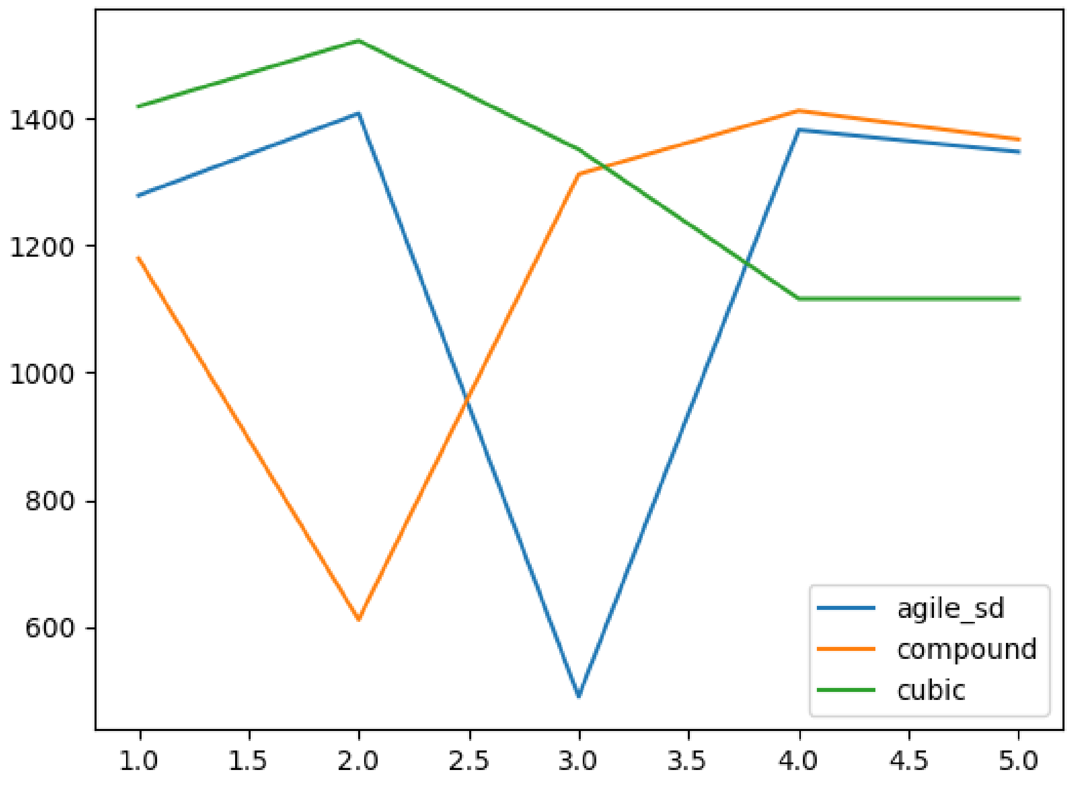
0.35
0.30
0.25
0.20
0.15
100
150
200
250
300
350
400
450
500
agile_sd
compound
cubic

Average Delay



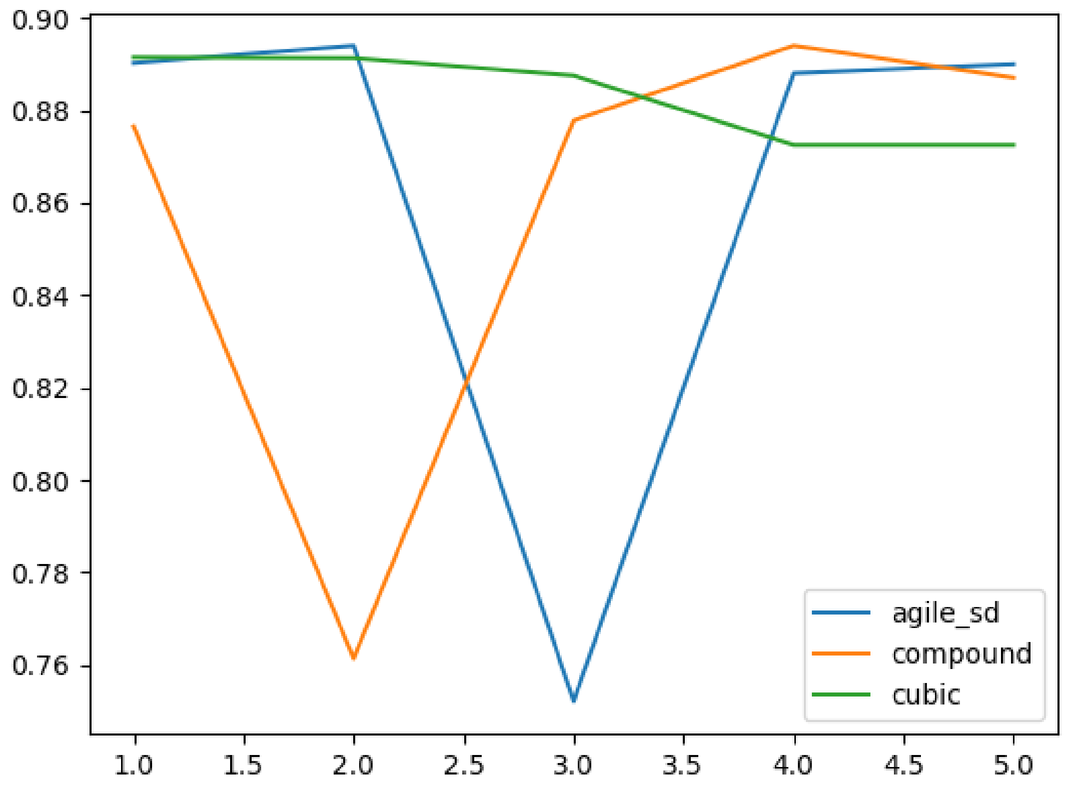
4.Varying Coverage Area

Throughput

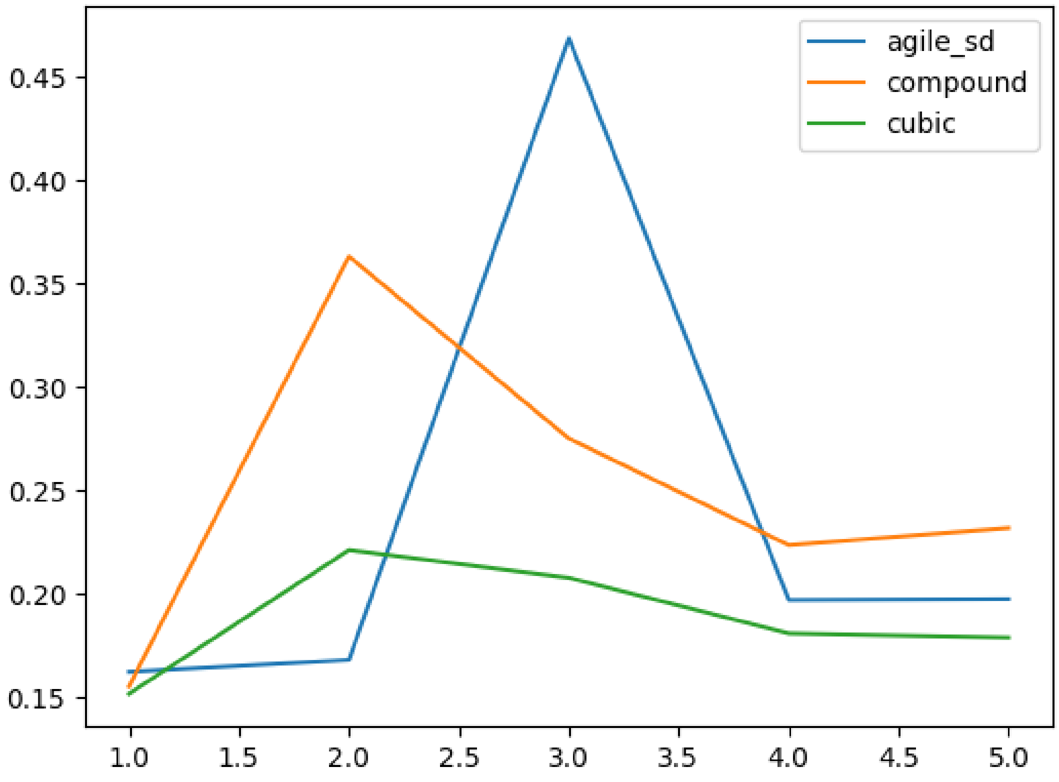


1400
1200
1000
800
600
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
agile_sd
compound
cubic

Delivery Ratio

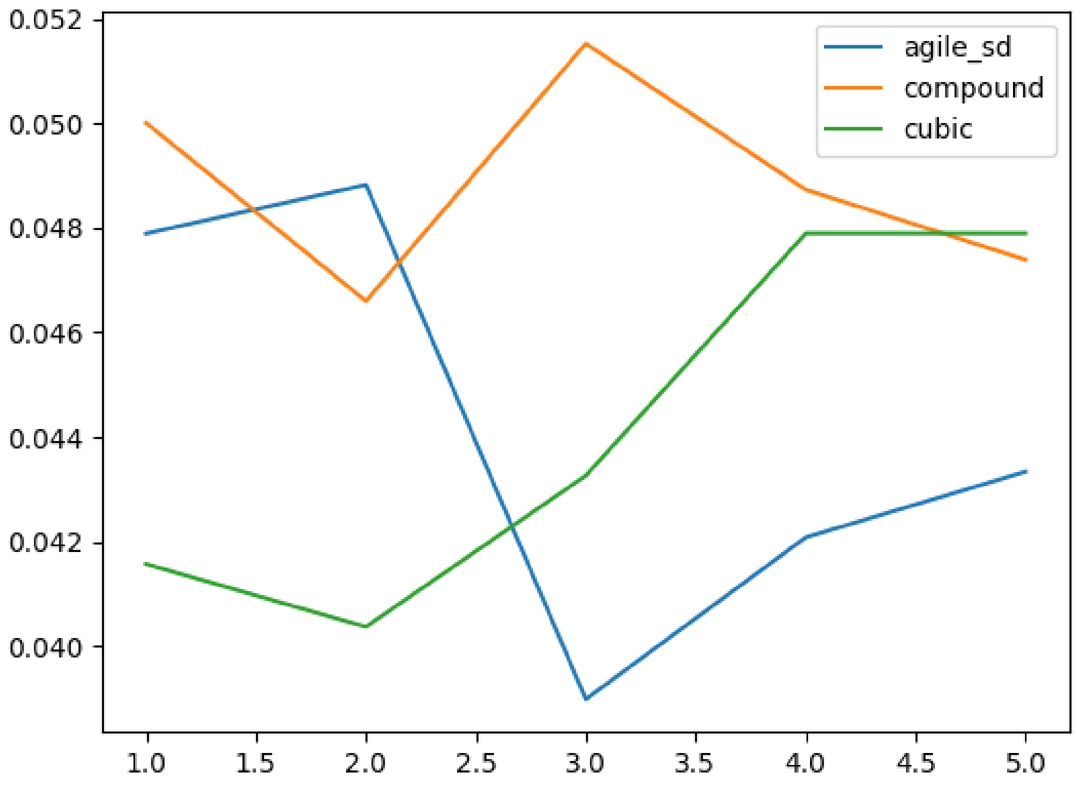


Drop Ratio



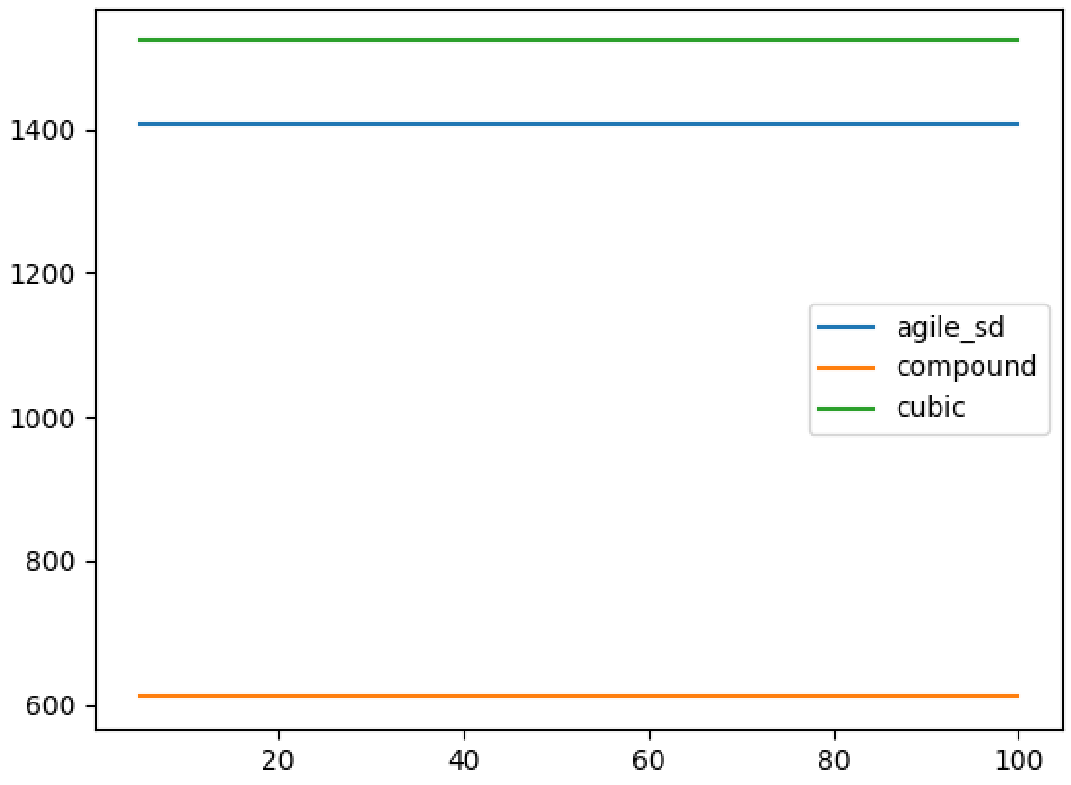
agile_sd
compound
cubic
0.45
0.40
0.35
0.30
0.25
0.20
0.15
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

Average Delay

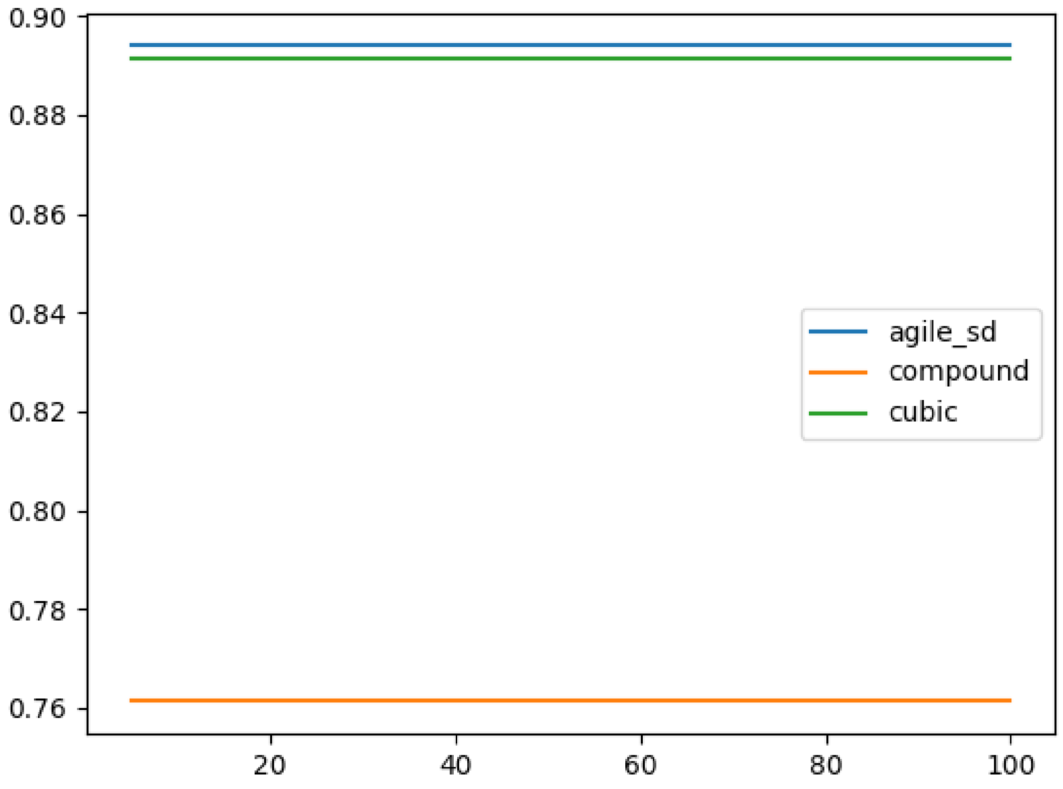


5.Varying number of Queue

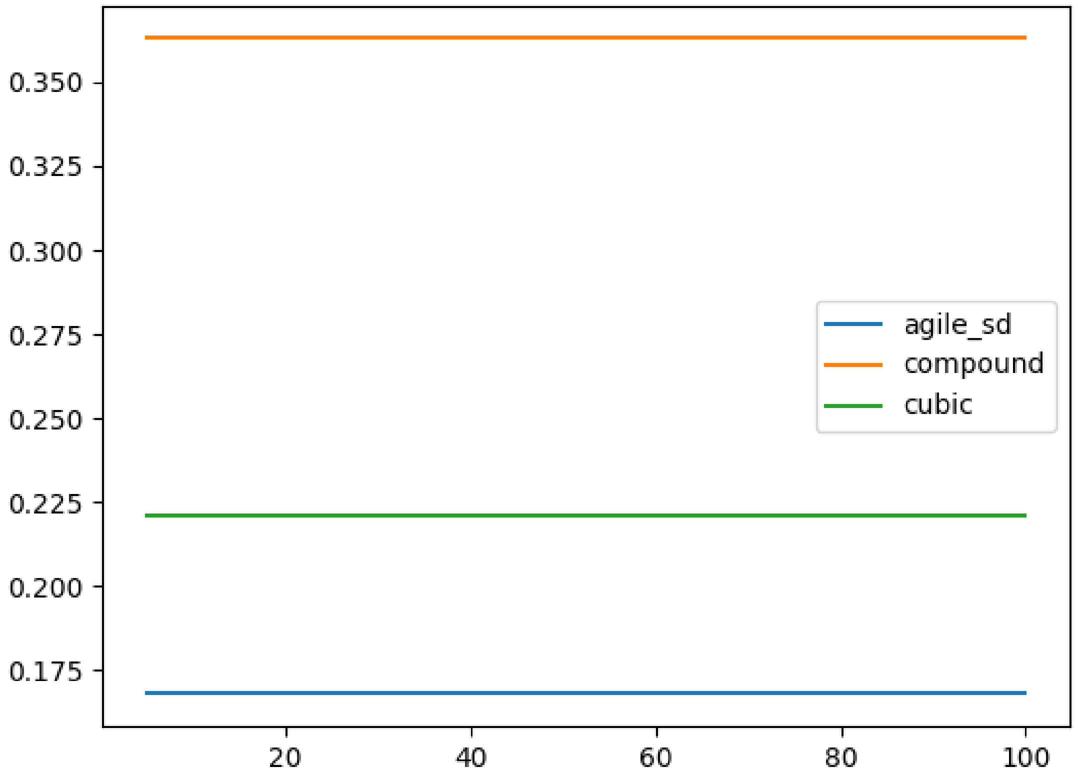
Throughput



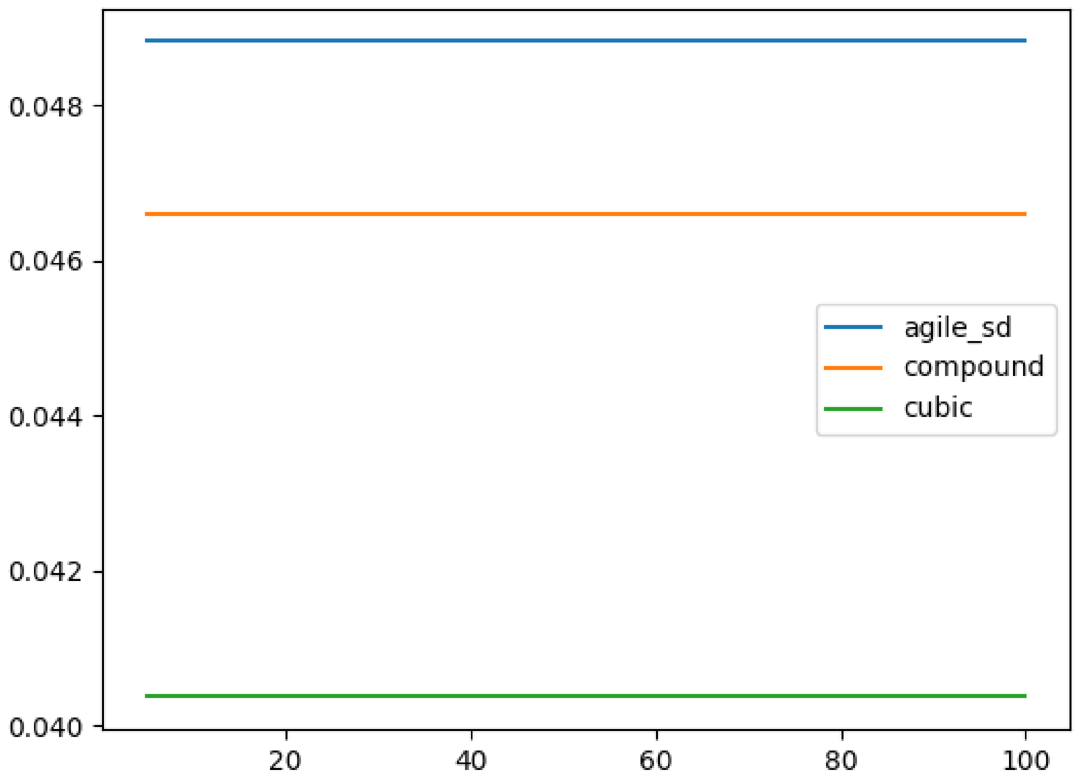
Delivery Ratio



Drop Ratio


0.350
0.325
0.300
0.275
0.250
0.225
0.200
0.175
20
40
60
80
100
agile_sd
compound
cubic

Average Delay



Summary :

1.According to the plots from very limited data set it is obvious that agile SD provides better throughput and of course better stability.

2.the proposed algorithm performed better than tcp compound in most of the cases

But it was tough to beat tcp cubic. Ofcourse in some cases it performed better than tcp cubic.
3.if we could add this agility factor to tcp cubic maybe there could be a better improvement of tcp cubic

4.the intention of this algorithm was to increase throughput. Which was somewhat served. But more study should be done on this algorithm.
5.Also this algorithm has drop ratio problem which should be fixed.

Source

Agile-SD

***A Linux-based TCP congestion control algorithm for supporting highspeed and short distance networks.**

Journal : Journal of Network and Computer Applications

Paper link

Time of Publish : June 2015