Adaptive Backoff Algorithm for Contention Window for Dense IEEE 802.11 WLANs

https://doi.org/10.1155/2016/8967281

CSE 322

Computer Networks Sessional

Name: MD. ZARZEES UDDIN SHAH CHOWDHURY

StudentID: 1805009

Lab Group: A1

Email: princezarzees5075@gmail.com

Supervisor: NAVID BIN HASAN, Lecturer, Computer Science and Engineering, BUET

Current implementation

- Increasing contention window exponentially when detecting a collision (DCF/Binary Exponential Backoff (BEB))
- Resetting contention window after a successful transmission

In mac-802_11.h of default NS2

```
inline void inc_cw()
{
        cw_ = (cw_ << 1) + 1;
        if(cw_ > phymib_.getCWMax())
          cw_ = phymib_.getCWMax();

    //update_cw();
}
inline void rst_cw()
{
        cw_ = phymib_.getCWMin();
    //update_cw();
}
```

Modification made in NS2

Calculating Probability

$$P(X \geq 2) = \binom{n}{l} \left(\frac{1}{N}\right)^{l} \left(1 - \frac{1}{N}\right)^{n-l}, \quad \text{where } l \geq 2. \quad (5)$$

Note: In the article, they took l as the contention number of a time slot. But in ns2 calculating contention number was not feasible. So I modified it in binomial distribution to calculate collision probability by summing over the probability of transmitting more than one node from l = 2 to l = n.

Here is the formula I used

$$P_{coll} = \sum_{l=2}^{n} \binom{n}{l} \left(\frac{1}{N}\right)^{l} \left(1 - \frac{1}{N}\right)^{n-l}$$

```
inline double p_coll()
{
    if (num_of_time_slots<=1)
        return num_of_time_slots;
    double temp=num_of_nodes;
    double p=1/num_of_time_slots;
    double t=p/(1-p);
    double C=temp*p*pow(1-p,temp-1);
    double s = 0.0;
    for (int i = 2; i <= num_of_nodes; i++)
    {
        temp--;
        C = C * (temp / i)*t;
        s=s+C;
    }
    return s;
}
```

$$n_{\text{est}} = \frac{\log\left(1 - \left(2/\left(\text{CW}_{\min} + 1\right)\right)\right) + \log\left(1 - P_{\text{coll}}\right)}{\log\left(1 - \left(2/\left(\text{CW}_{\min} + 1\right)\right)\right)}, \qquad (11)$$

Step 1: Set CW_{min}, and set the threshold value for the collision probability Th_{Coll}.
Step 2: Calculate P_{Coll} using (5).
Step 3: Compare P_{Coll} with Th_{Coll}.
while ($P_{Coll} \geq Th_{Coll}$)
Estimate the number of active stations using (11).
Calculate CW_{min} obtained by (10) using n_{est}.
Update the P_{Coll} value using (5).
End
Step 4: Estimate the number of active stations using the updated P_{Coll} and CW_{min} obtained by (11).

ALGORITHM 1: Estimation of the number of active stations.

$$CW^* = \left(\frac{CW_{min}}{2}\right) \times n - 1, \tag{10}$$

while (contention)
A station observes the channel for backoff period.
Calculate P_{Coll} using (5).
Estimate n_{est}, following the procedure in Algorithm 1.
Calculate CW^* using (10).
Choose a random backoff value from the range $[0, CW^* - 1]$.
Decrement backoff timer by one when the channel is idle as DCF protocol.
Transmit a packet when the backoff timer expires.
End

ALGORITHM 2: Adaptive backoff algorithm of contention window.

All the other formulas are as it is as in the research article. Here is the code for updating contention window.

```
inline void update_cw()
{

   double p_col = p_coll();
   int c=0;
    while (p_col > 0.21 && cw_>1 && p_col<=0.999)
    {
      double estimate = (log(fabs(1 - 2.0 / (cw_ + 1))) + log(fabs(1 - p_col))) / (log(fabs(1 - 2.0 / (cw_ + 1))));
      num_of_nodes = floor(estimate)+2;
      p_col = p_coll();
      c++;
      if (c>1000)
          break;
    }
   if (cw_>1 && p_col<=0.999)
   {double estimate = (log(fabs(1 - 2.0 / (cw_ + 1))) + log(fabs(1 - p_col))) / (log(fabs(1 - 2.0 / (cw_ + 1))));
   num_of_nodes = floor(estimate)+2;
   }
   cw_ = floor((cw_ / 2.0) * (num_of_nodes - 1));
   if (cw_ > phymib_.getCWMax())
      cw_ = phymib_.getCWMax();
   else if (cw_ < phymib_.getCWMin())
      cw_ = phymib_.getCWMin();
  //printf("%d %lf\n",num_of_nodes,p_col);
}
```

Wherever **inc_cw()** and **rst_cw()** is called, I replaced it with **update_cw()**.

```
inline void inc_cw()
{
    //    cw_ = (cw_ << 1) + 1;
    //    if(cw_ > phymib_.getCWMax())
    //        cw_ = phymib_.getCWMax();

    update_cw();
}
inline void rst_cw()
{
    //cw_ = phymib_.getCWMin();
    update_cw();
}
```

In channel.cc a global variable “**num_of_nodes**” is declared to get total number of nodes.

channel.cc > ...

```
#include "ip.h"
#include "dsr/hdr_sr.h"
#include "gridkeeper.h"
#include "tworayground.h"
#include "wireless-phyExt.h"

int num_of_nodes=0; //edited by zarzees

static class ChannelClass : public TclClass {
public:
    ChannelClass() : TclClass("Channel") {}
    TclObject* create(int, const char*const*) {
        return (new Channel);
    }
} class_channel;
```

mac-802_11.h 4 | channel.cc 7

mac > channel.cc > removeNodeFromList(MobileNode *)

```
395
396   void
397   WirelessChannel::addNodeToList(MobileNode *mn)
398   {
399       MobileNode *tmp;
400
401       // create list of mobilenodes for this channel
402       if (xListHead_ == NULL) {
403           fprintf(stderr, "INITIALIZE THE LIST xListHead\n");
404           xListHead_ = mn;
405           xListHead_->nextX_ = NULL;
406           xListHead_->prevX_ = NULL;
407       } else {
408           for (tmp = xListHead_; tmp->nextX_ != NULL; tmp=tmp->nextX_);
409           tmp->nextX_ = mn;
410           mn->prevX_ = tmp;
411           mn->nextX_ = NULL;
412       }
413       numNodes_++;
414       num_of_nodes++;  //edited by ZARZEES
415
416
417   }
418
```

In mac-timers.cc a variable “**num_of_time_slots**” is used to get time slots at that time.

Network Topologies and Parameters Under Simulation:

1. Wireless (802.11 and 802.15.4)
2. Mobility: Static
3. Positioning of Nodes (Grid 500×500)
4. Number of Nodes: 20, 40, 60, 80, 100
5. Number of Flows: 10, 20, 30, 40, 50
6. Packets per second: 100, 200, 300, 400, 500
7. Coverage: 1, 2, 3, 4, 5 (×TxRange, 1 for 100m)
   When varying parameters, other parameters are taken as follows
   - Number of nodes 40
   - Number of Flows 20
   - Packets per second 200
   - Coverage 3×TxRange
8. Simulation time 50 seconds

Procedure:

A bash script(project.sh) was run including all the necessary commands for all the parameters. Output was redirected to a csv(temp1.csv) file and all the graphs were generated by Excel. Parsing was done by parse.awk file and the simulation is in project.nam.

Figure: A simulation snapshot of nam

Results:

Table 1 Varying Number of Nodes

Number of Nodes	Throughput OLD	Average Delay OLD	Delivery ratio OLD	Drop ratio OLD	Energy Consumption OLD	Throughput NEW	Average Delay NEW	Delivery ratio NEW	Drop ratio NEW	Energy Consumption NEW
20	160936	0.0268599	0.998016	0.00198413	470.72	168990	0.073627	0.99777	0.00222965	469.51
40	173546	0.0673481	0.999077	0.000922509	952.376	167997	0.0618989	1	0	941.628
60	180563	0.107372	1	0	1414.75	219425	0.136935	0.996967	0.00303337	1402.9
80	198701	0.158408	0.999197	0.000803213	1899.8	186958	0.141907	0.997732	0.00199402	1879.77
100	205884	0.101772	0.99845	0.00155039	2379.2	216029	0.173628	0.999163	0.00083682	2438.36

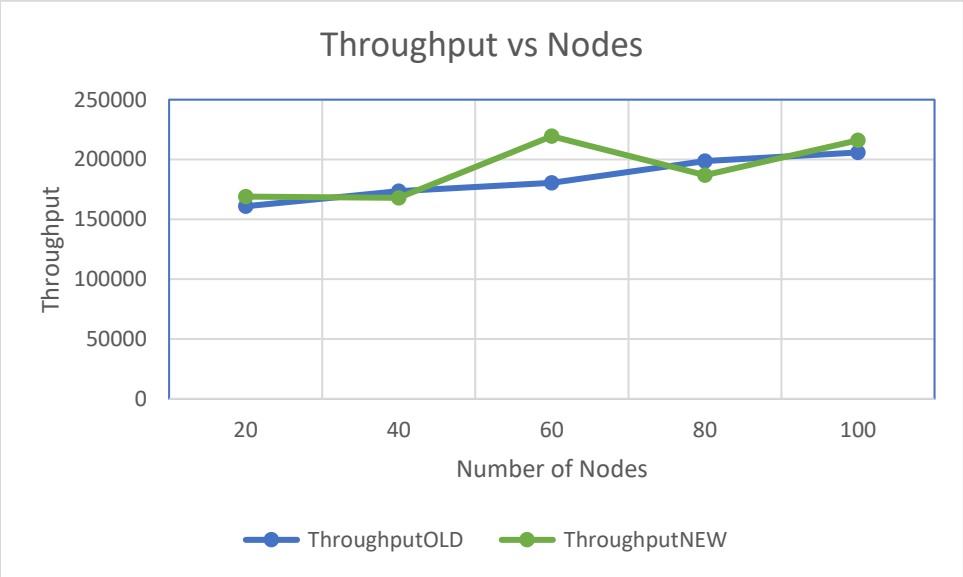


Figure 1.1: Throughput(bits/sec) vs Number of nodes

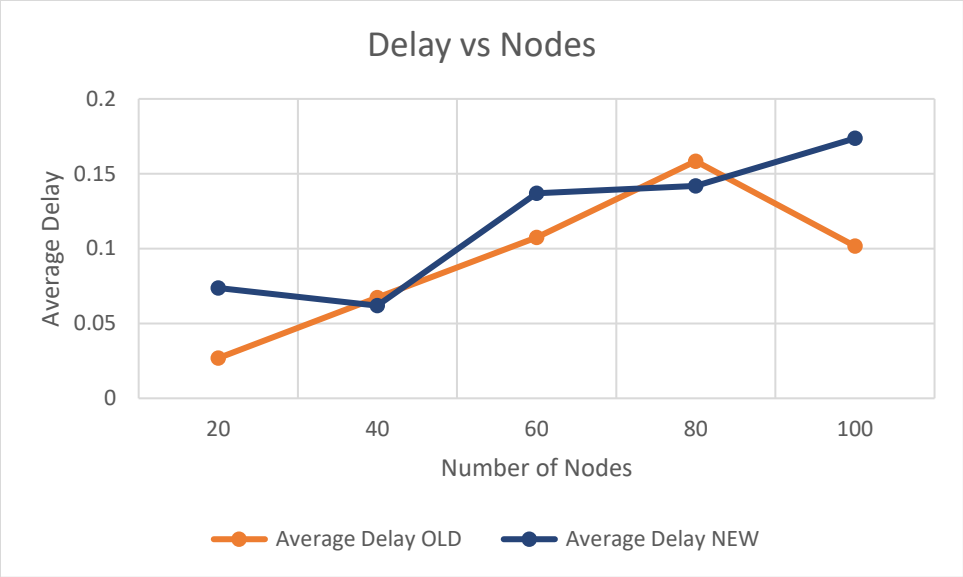


Figure 1.2: Average Delay(sec) vs Number of nodes

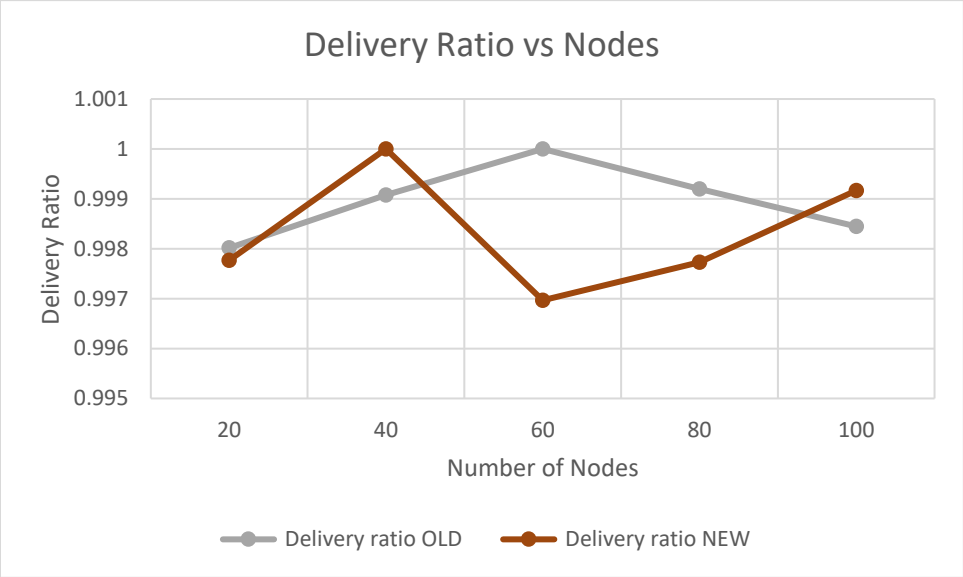


Figure 1.3: Delivery Ratio vs Number of nodes

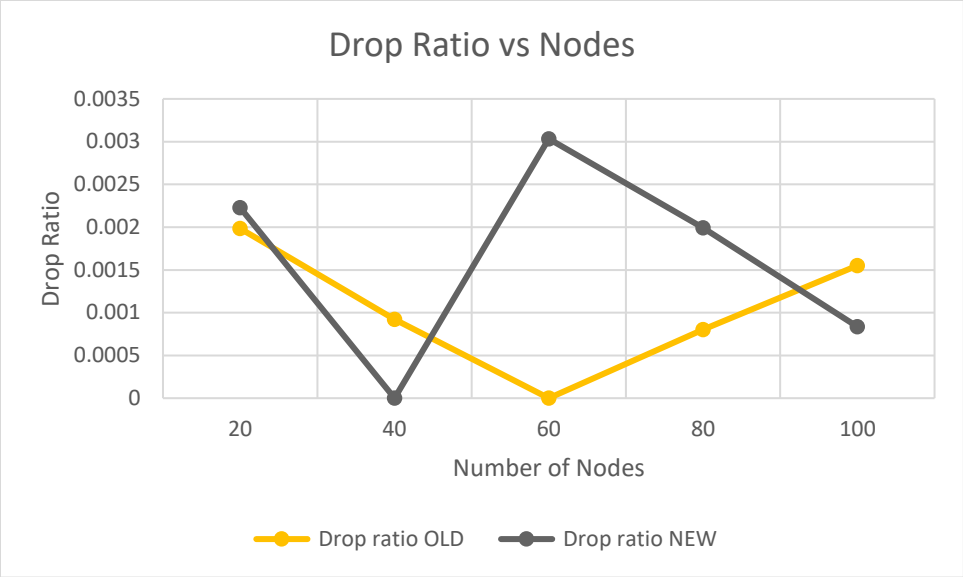


Figure 1.4: Drop Ratio vs Number of nodes

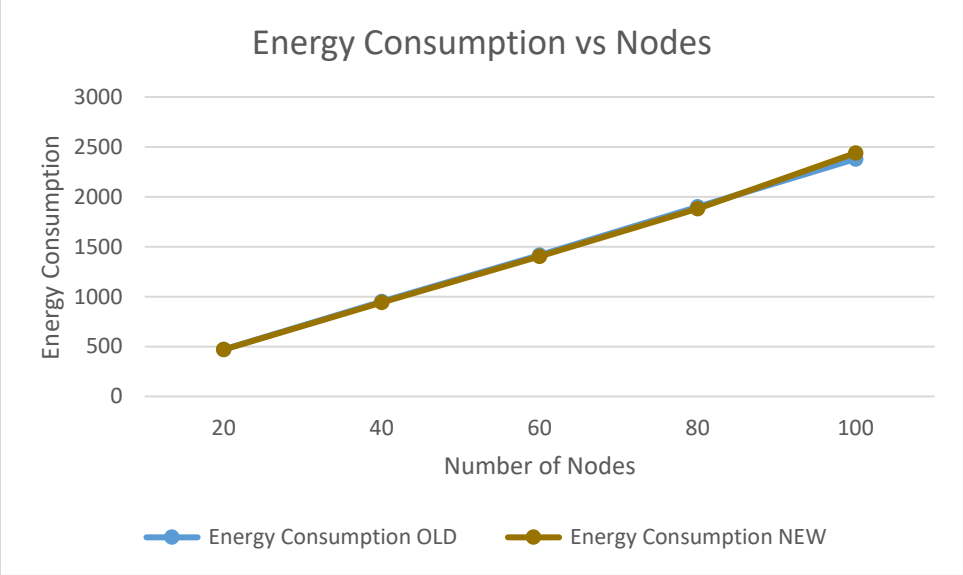


Figure 1.5: Energy Consumption(Joule) vs Number of nodes

Table 2 Varying Number of Flows

Number of Flows	Throughput OLD	Average Delay OLD	Delivery ratio OLD	Drop ratio OLD	Energy Consumption OLD	Throughput NEW	Average Delay NEW	Delivery ratio NEW	Drop ratio NEW	Energy Consumption NEW
10	85059.2	0.0196029	1	0	919.529	84896	0.0234881	1	0	917.788
20	195904	0.0477103	1	0	948.515	191504	0.0903508	1	0	939.893
30	302520	0.127417	0.995253	0.0021097	978.275	268289	0.17418	0.999395	0.000604961	962.52
40	390381	0.894675	0.976819	0.0039968	998.3	366580	0.991706	0.996114	0.00345423	994.543
50	392920	1.14036	0.948151	0.00724361	999.218	395194	1.43108	0.984974	0.00355872	995.695

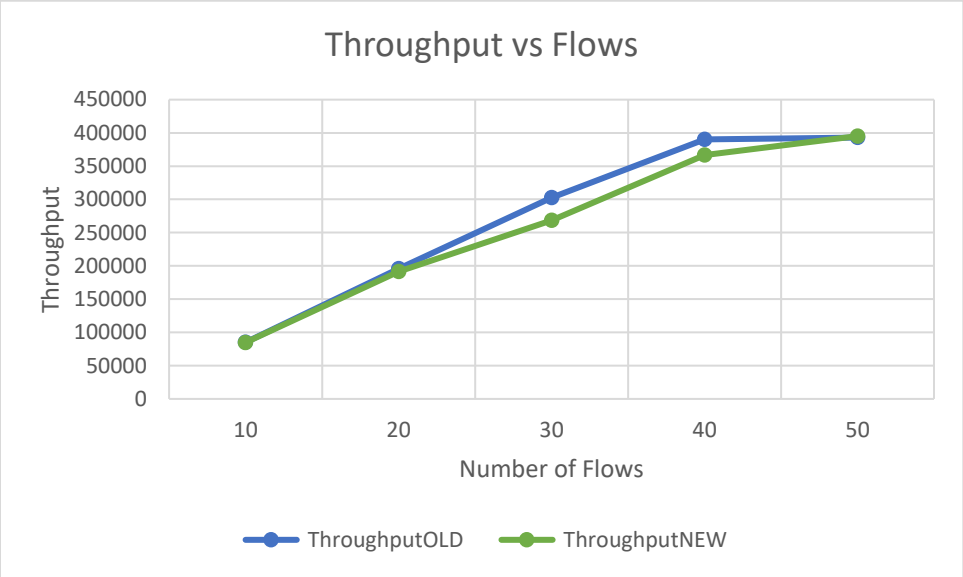


Figure 2.1: Throughput vs Number of flows

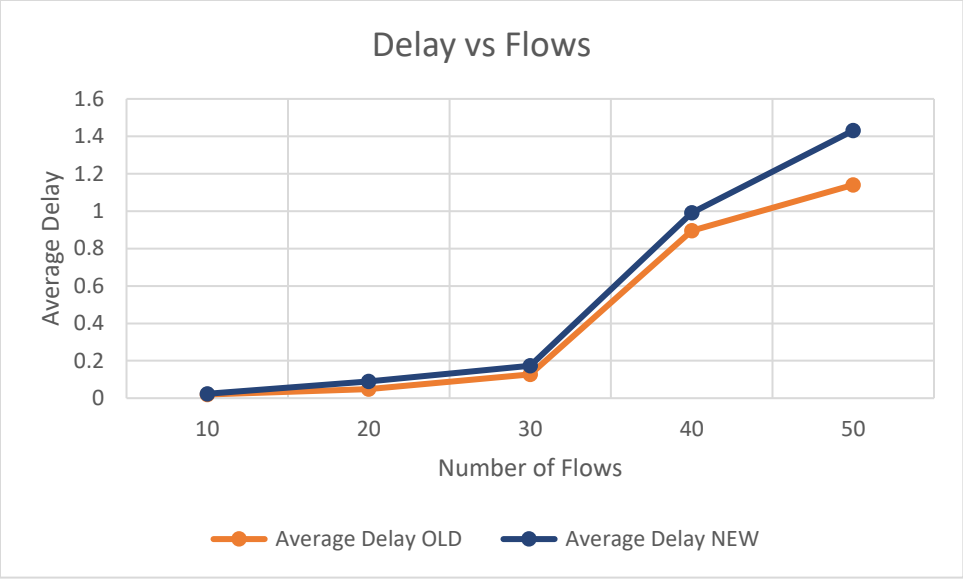


Figure 2.2: Average Delay vs Number of flows

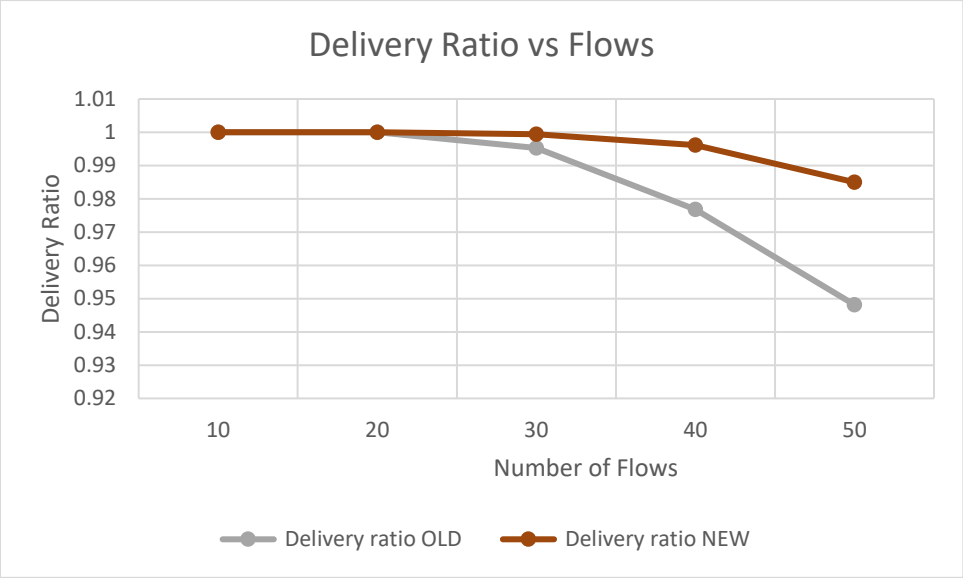


Figure 2.3: Delivery Ratio vs Number of flows

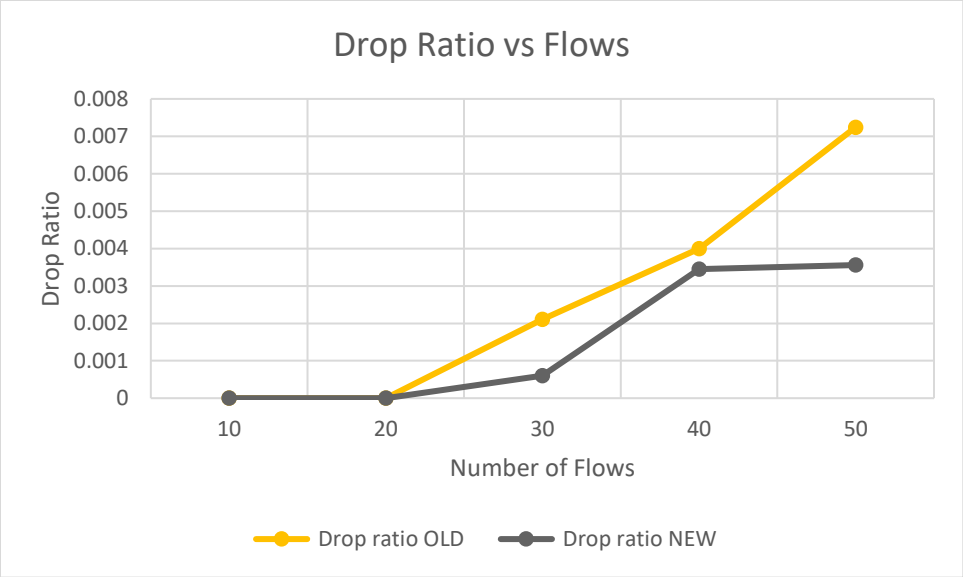


Figure 2.4: Drop Ratio vs Number of flows

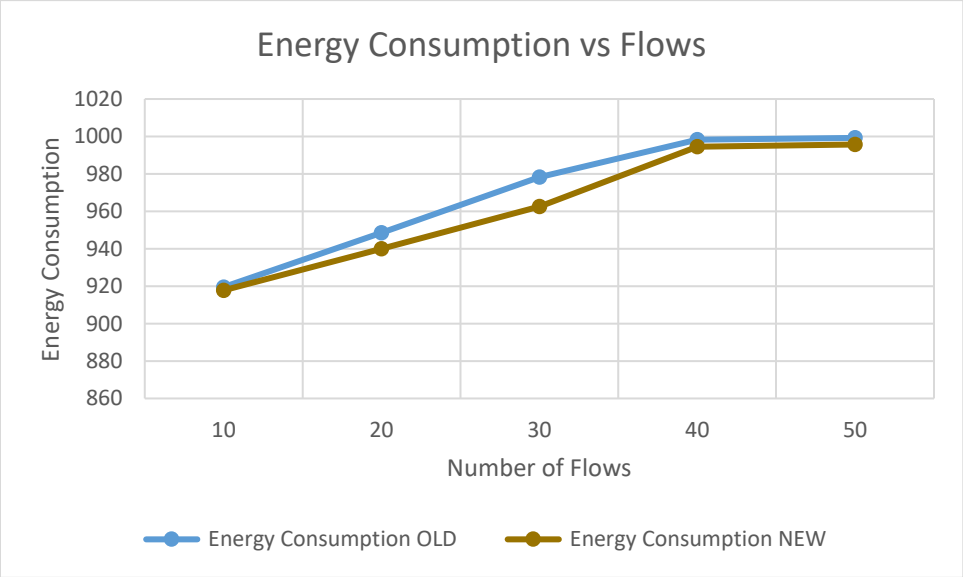


Figure 2.5: Energy Consumption vs Number of flows

Table 3 Varying Packets per second

Packets Per Second	Throughput OLD	Average Delay OLD	Delivery ratio OLD	Drop ratio OLD	Energy Consumption OLD	Throughput NEW	Average Delay NEW	Delivery ratio NEW	Drop ratio NEW	Energy Consumption NEW
100	144822	0.0192375	1	0	926.943	175833	0.0729386	1	0	944.706
200	206189	0.0333827	0.999222	0.0007 7821	949.629	187782	0.0734348	0.999146	0.00085 3971	952.02
300	212554	0.0340293	0.997738	0	952.677	152664	0.0758007	0.997917	0	949.692
400	152170	0.0212006	1	0	933.875	199331	0.0574319	1	0	946.965
500	153146	0.0300373	0.998957	0.0010 4275	941.256	176650	0.0621937	0.999094	0.00090 5797	951.256

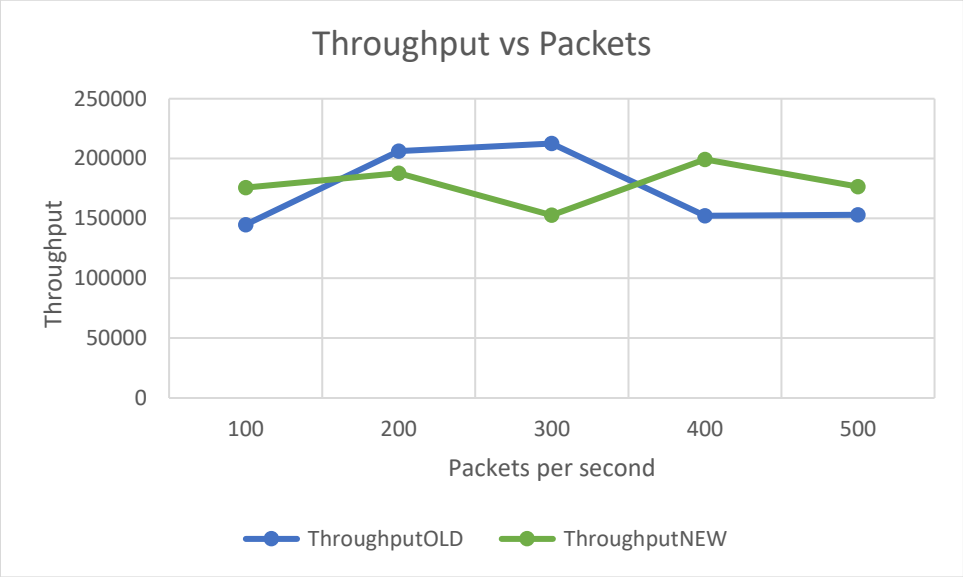


Figure 3.1: Throughput vs Number of Packets

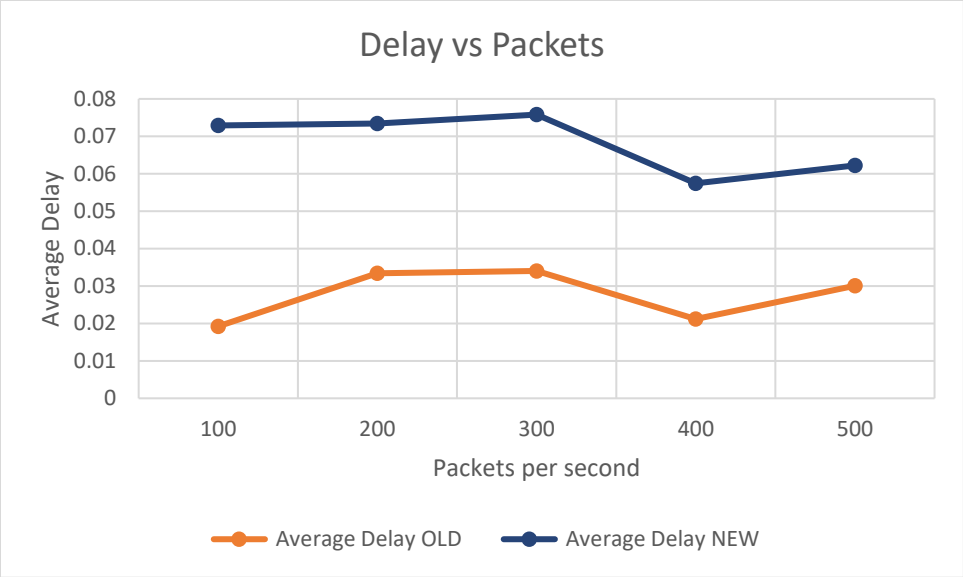


Figure 3.2: Average Delay vs Number of packets

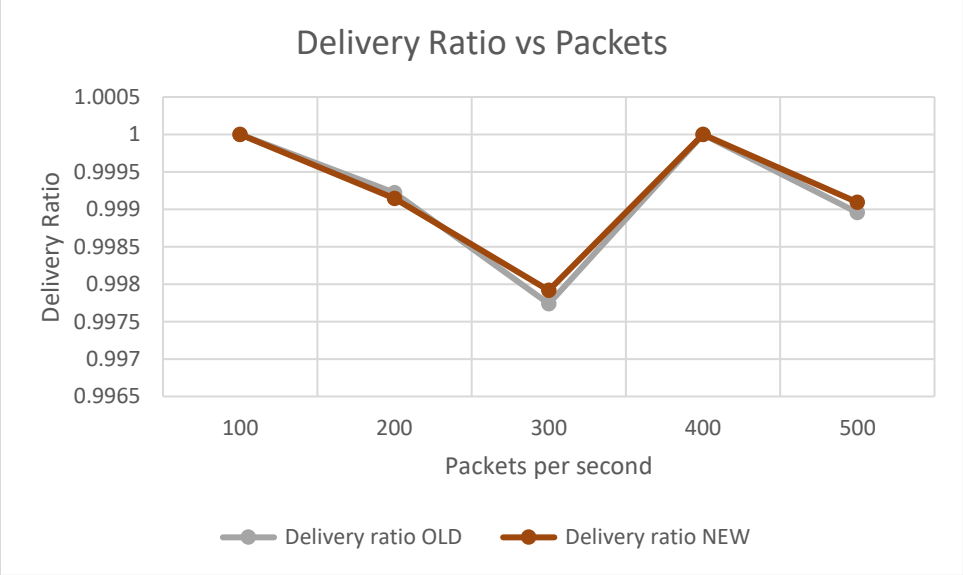


Figure 3.3: Delivery Ratio vs Number of packets

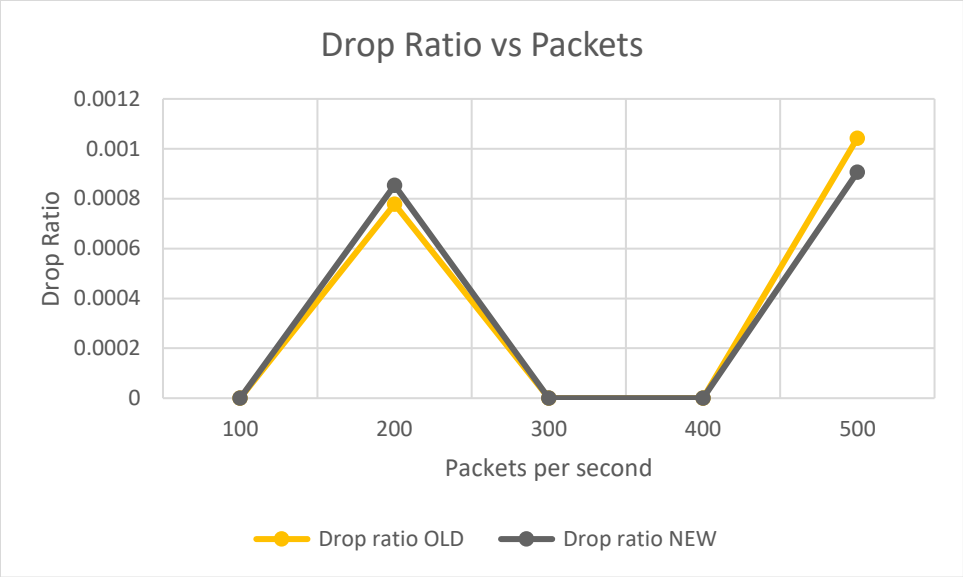


Figure 3.4: Drop Ratio vs Number of packets

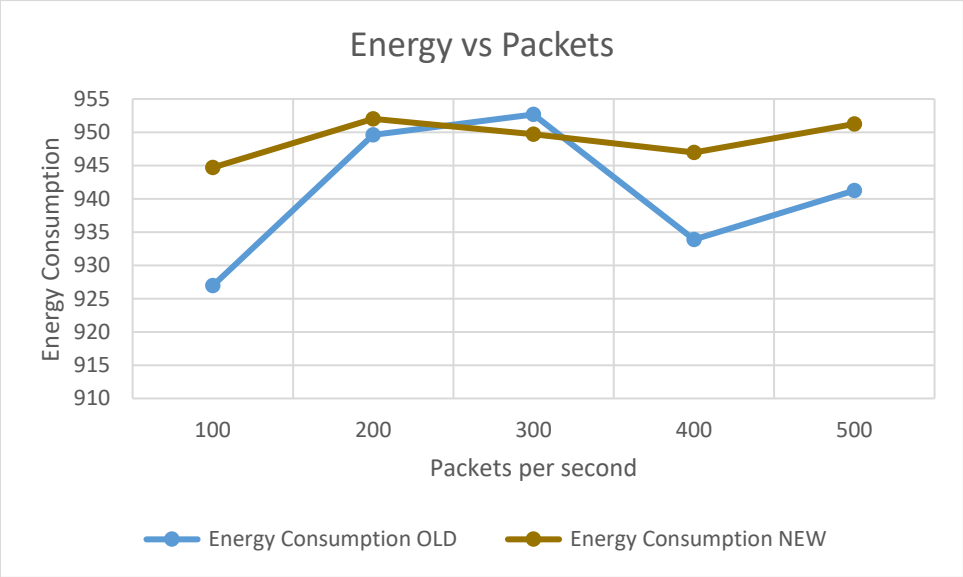


Figure 3.5: Energy Consumption vs Number of packets

Table 4 Varying TxRange

TxRange	Throughput OLD	Average Delay OLD	Delivery ratio OLD	Drop ratio OLD	Energy Consumption OLD	Throughput NEW	Average Delay NEW	Delivery ratio NEW	Drop ratio NEW	Energy Consumption NEW
1	198907	0.403669	0.991987	0.00560897	981.407	160903	1.34559	0.95351	0.00759013	986.351
2	168659	0.0511213	0.998108	0.00189215	955.069	178447	0.204903	0.999103	0.000896861	956.003
3	188397	0.0275268	0.997451	0.00169924	944.103	223155	0.0599929	0.997245	0.00183655	946.504
4	168816	0.0151073	0.998108	0.000946074	931.167	175992	0.0487446	0.99909	0	936.673
5	197702	0.0131114	1	0	934.827	210592	0.0220871	0.998476	0	937.417

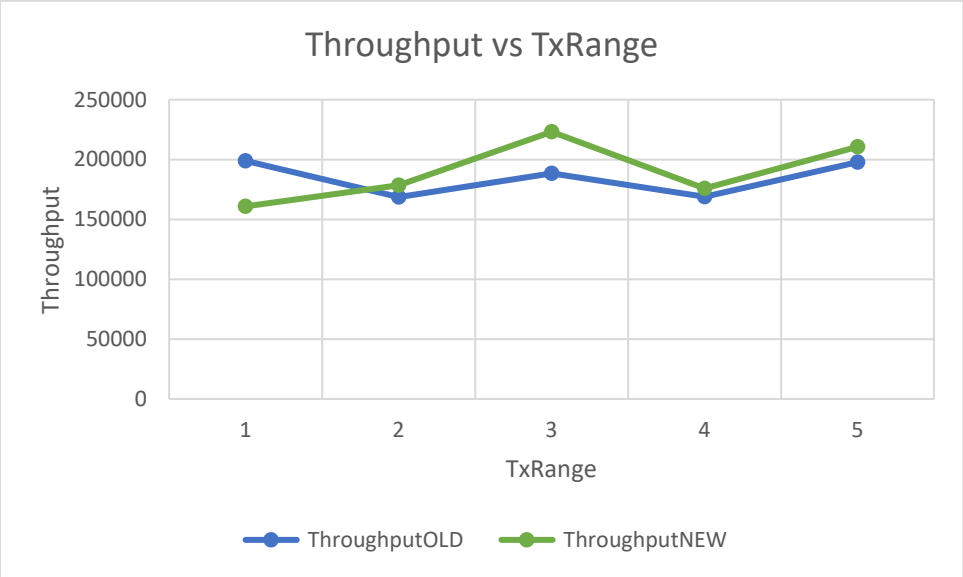


Figure 4.1: Throughput vs TxRange

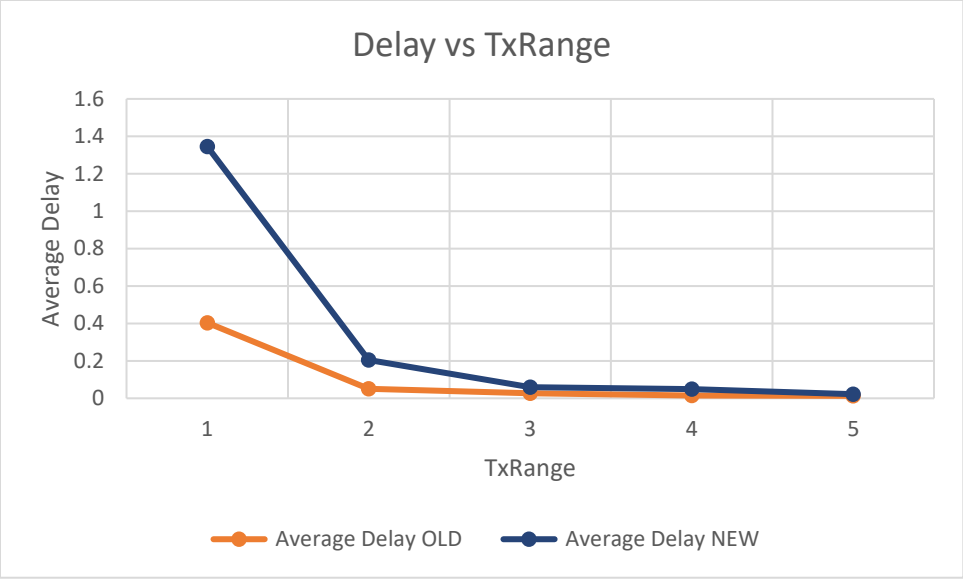


Figure 4.2: Average Delay vs TxRange

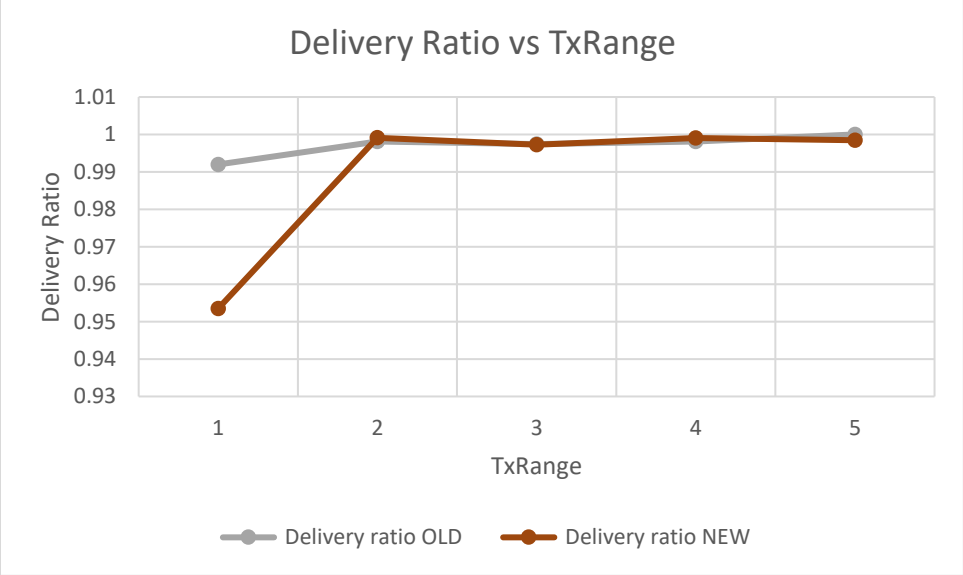


Figure 4.3: Delivery Ratio vs TxRange

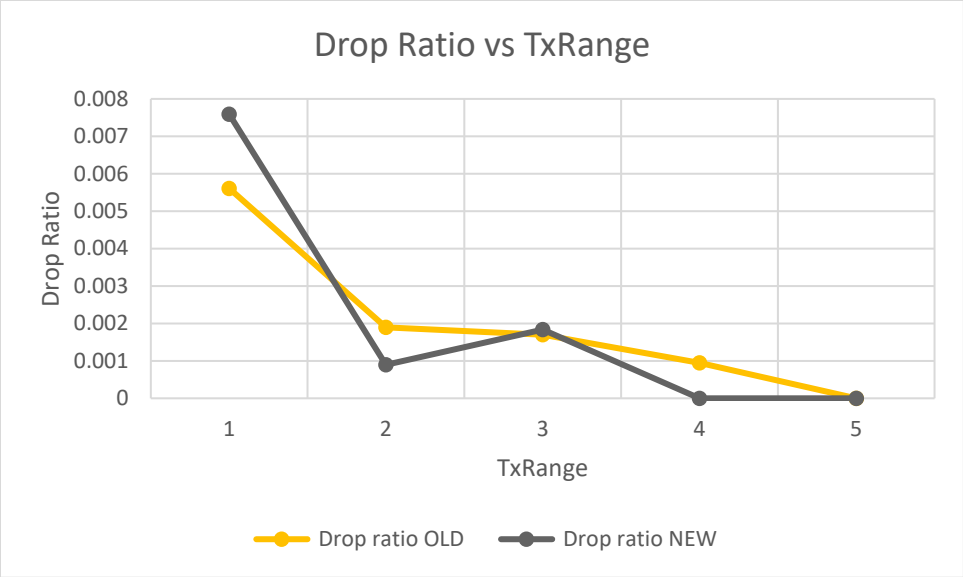


Figure 4.4: Drop Ratio vs TxRange

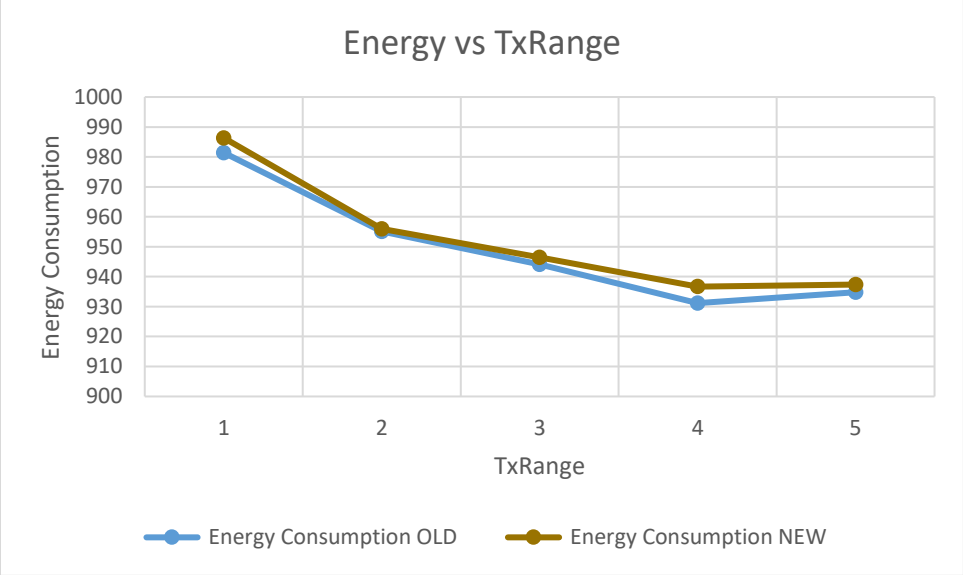


Figure 4.5: Energy Consumption vs TxRange

Summary Findings

According to my selected research article, throughput should be improved and collision rate should be reduced. But as I had to change the formula for probability of collision (finding contention number of a slot was not feasible in NS2) to binomial distribution, a vast improvement could not be found. Though Figure 1.1, 3.1, 4.1 shows some improvement over the built in method. Figure 2.1 gives almost same result.

Delay is more or less higher than the default, cause finding the perfect contention window through a while loop taking some time. See Figure 1.2, 2,2, 3.2 and 4.2.

Figure 1.3, 2.3 show some improvement in delivery ratio. 3.3 and 4.3 almost give same result as previous.

Drop ratio shows some clear improvement specially in Figure 2.4 and 4.4. Even it is clear from Figure 1.4 that when number of nodes increased, drop ratio decreased with new method.

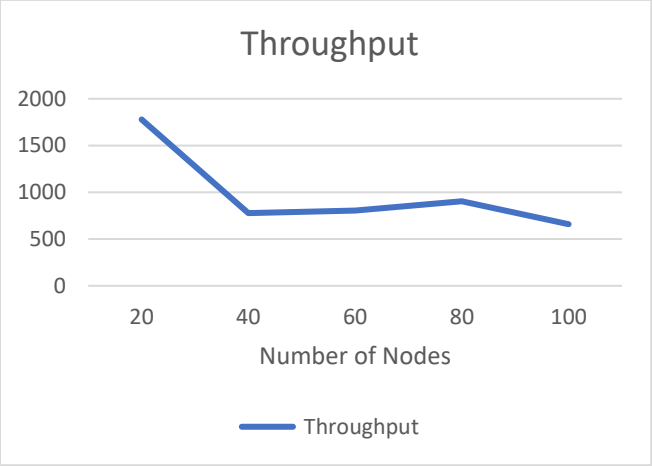
Surprisingly even increasing the overhead of calculating optimum contention window iteratively, does not consume that much energy. Sometimes it consumes less (Figure 2.5). Overall the energy consumption is same.

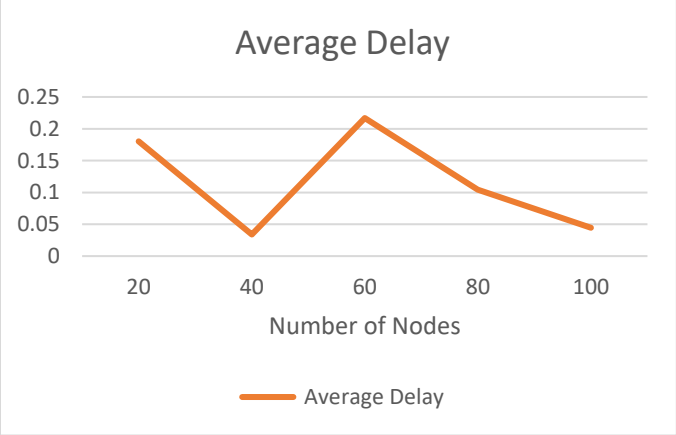
In a nutshell, there was not that much colossal improvement in all cases. But throughput, delivery ratio and drop ratio give better result overall in majority cases. It might happen due to changing the formula of collision probability. As the writers show using the original formula, there will be a much better result.

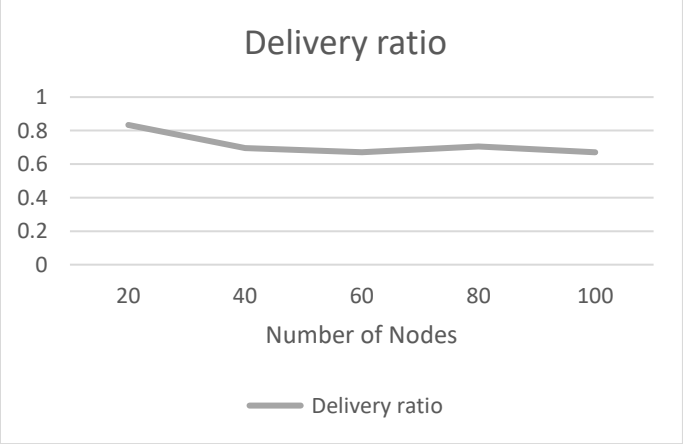
Since using naïve probability distribution like binomial, giving better result in throughput and drop ratio, it can be said that the authors' claim is correct about using adaptive back off algorithm for contention window.

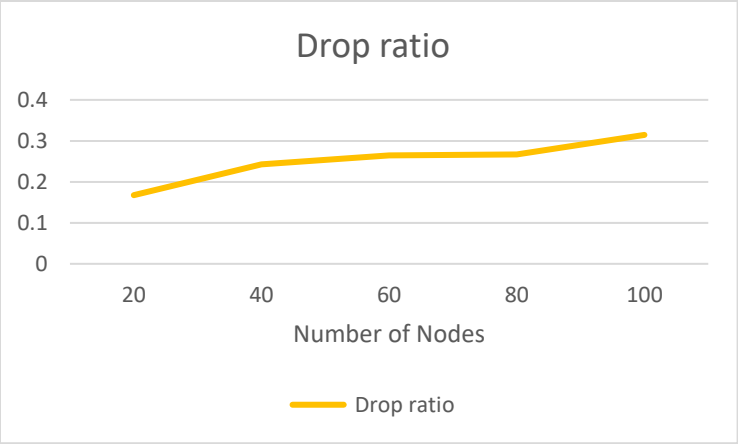
Additional Simulation (Not part of project)

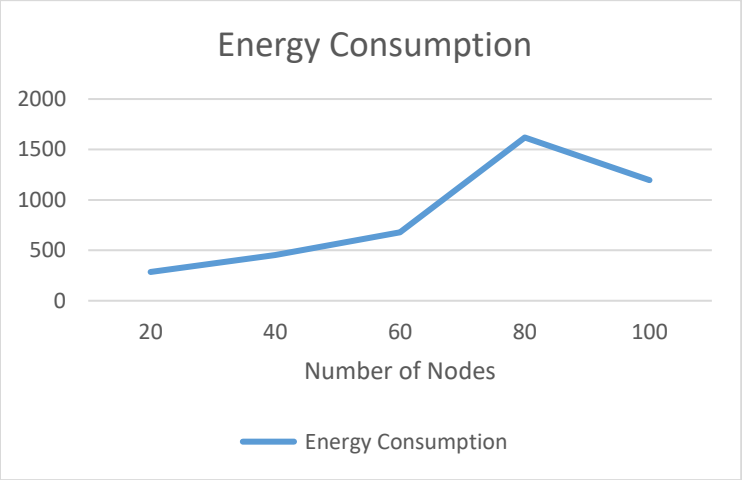
In 802.15.4

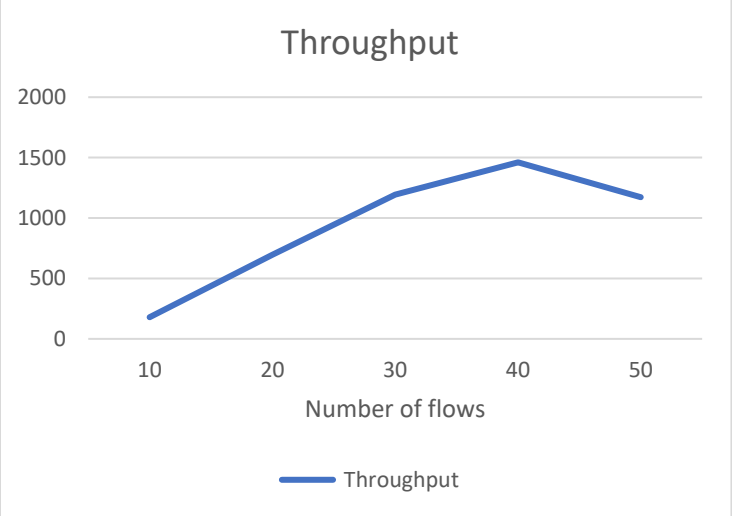


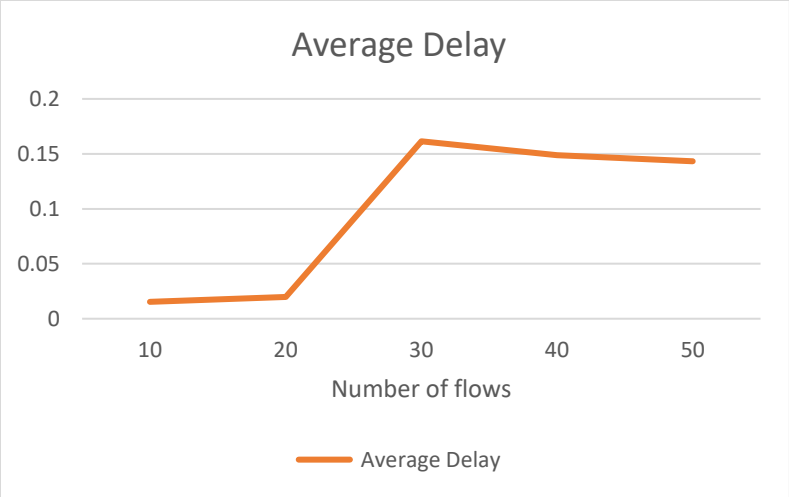


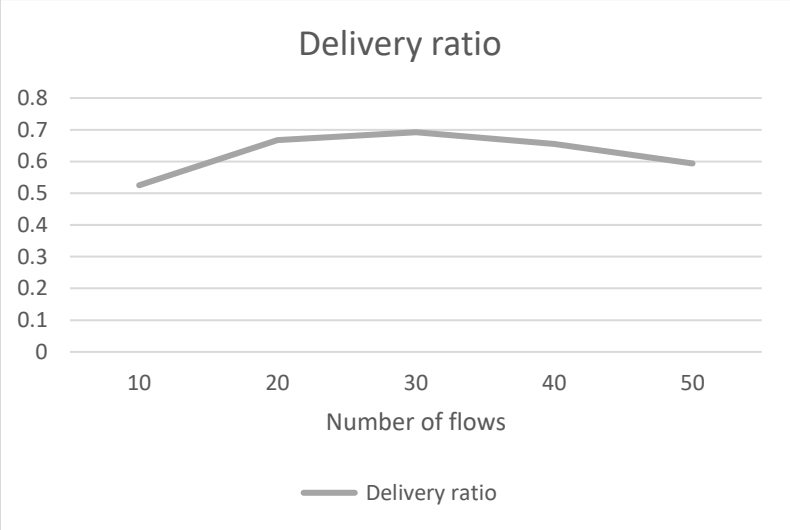


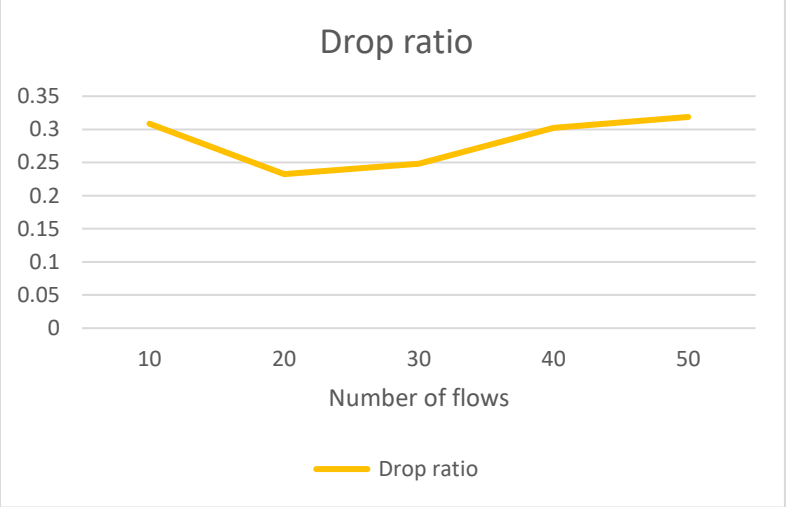

Drop ratio
0.4
0.3
0.2
0.1
0
20
40
60
80
100
Number of Nodes
Drop ratio

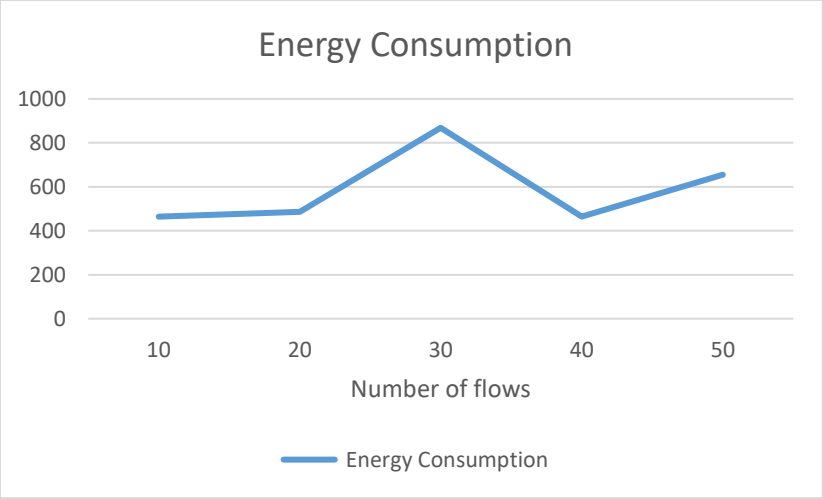

Energy Consumption
2000
1500
1000
500
0
20
40
60
80
100
Number of Nodes
Energy Consumption

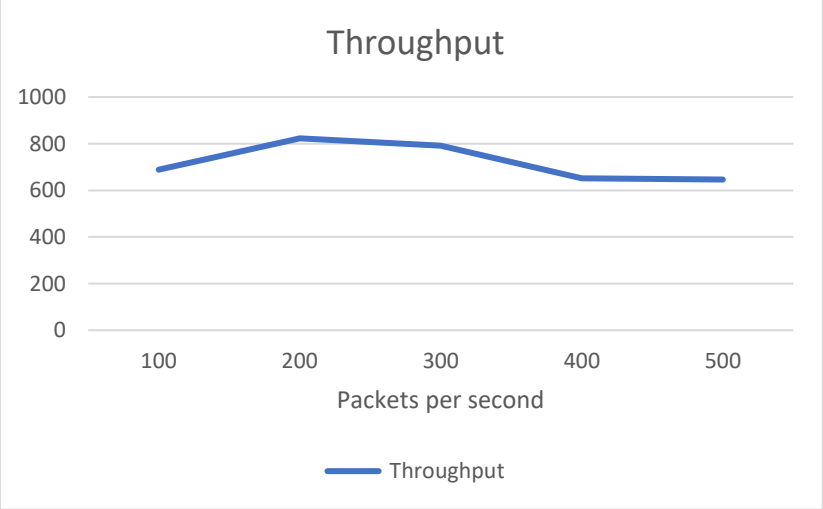

Throughput
2000
1500
1000
500
0
10
20
30
40
50
Number of flows
Throughput

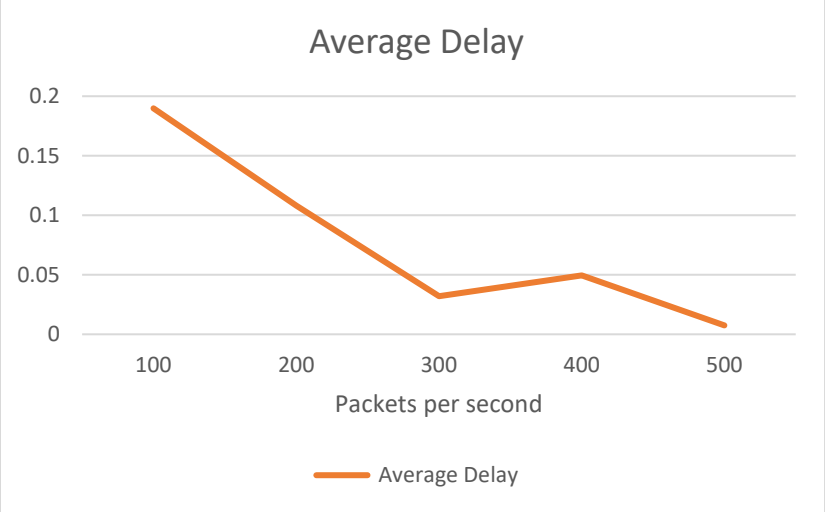

Average Delay
0.2
0.15
0.1
0.05
0
10
20
30
40
50
Number of flows
Average Delay


Delivery ratio
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
Number of flows
Delivery ratio

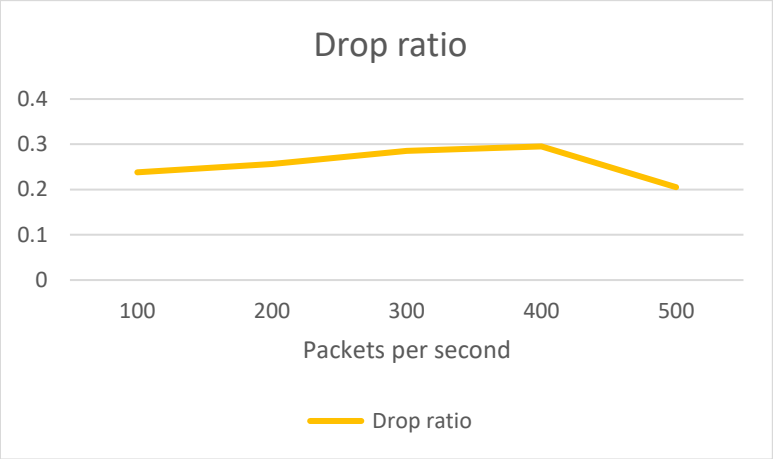

Drop ratio
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10
20
30
40
50
Number of flows
Drop ratio

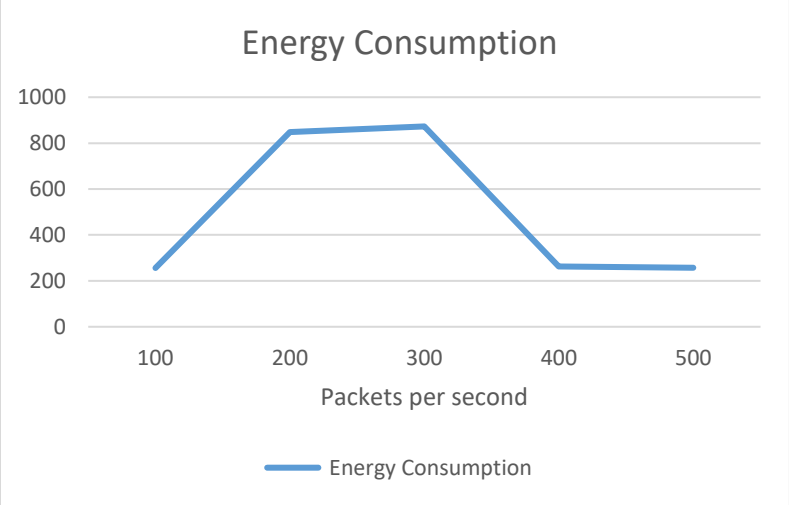

Energy Consumption
1000
800
600
400
200
0
10
20
30
40
50
Number of flows
Energy Consumption

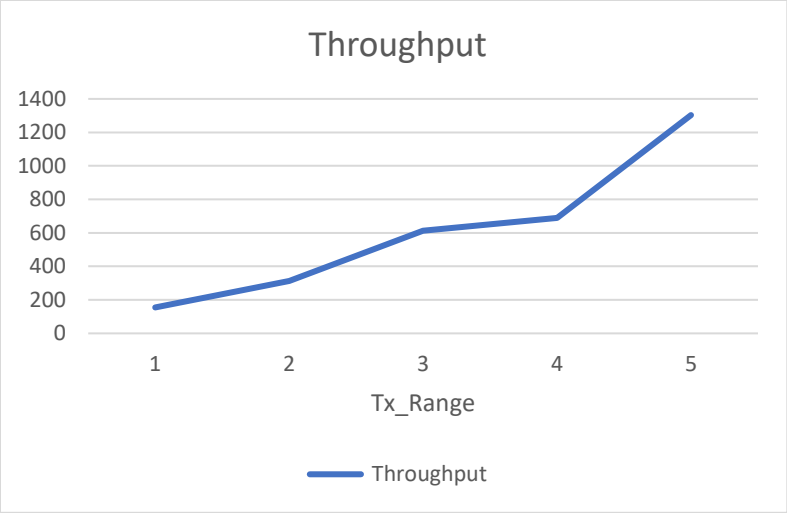

Throughput
1000
800
600
400
200
0
100
200
300
400
500
Packets per second
Throughput

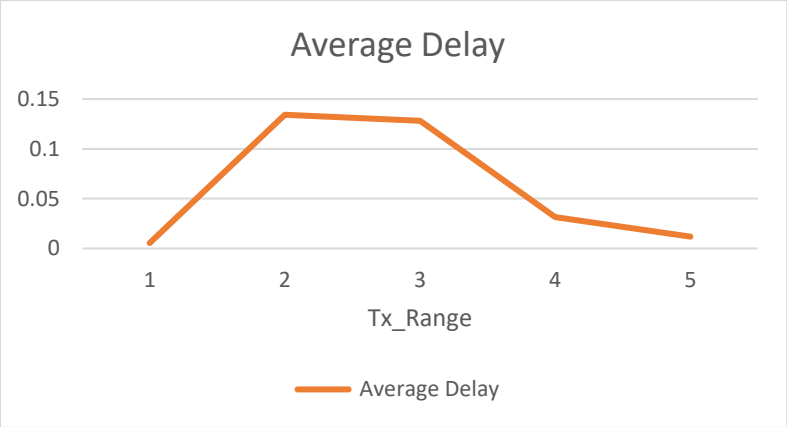

Average Delay
0.2
0.15
0.1
0.05
0
100
200
300
400
500
Packets per second
Average Delay

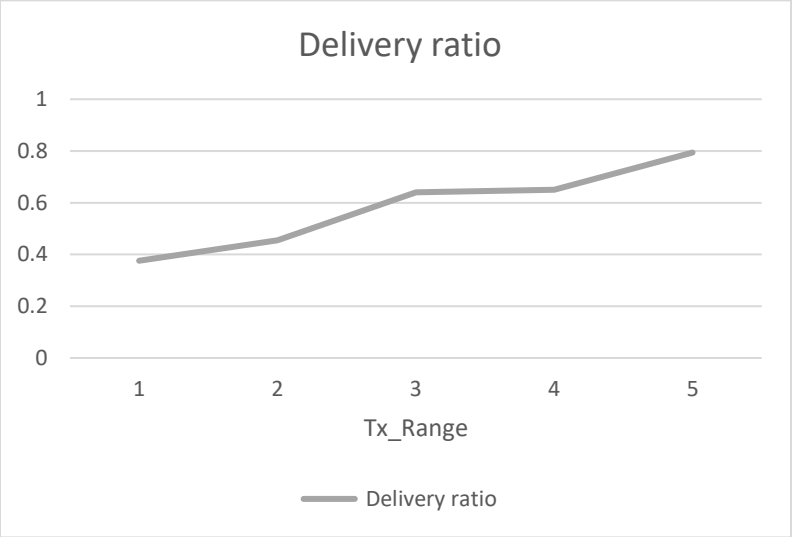

Delivery ratio
0.72
0.7
0.68
0.66
0.64
0.62
0.6
100
200
300
400
500
Packets per second
Delivery ratio

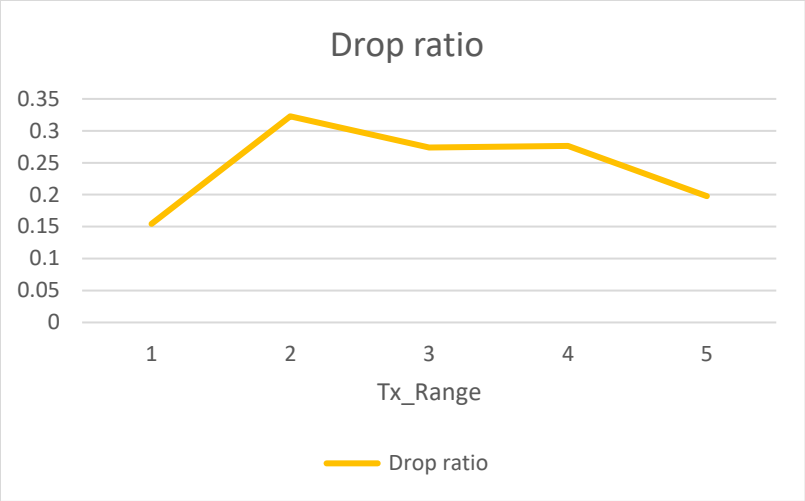

Drop ratio
0.4
0.3
0.2
0.1
0
100
200
300
400
500
Packets per second
Drop ratio


Energy Consumption
1000
800
600
400
200
0
100
200
300
400
500
Packets per second
Energy Consumption


Throughput
1400
1200
1000
800
600
400
200
0
1
2
3
4
5
Tx_Range
Throughput


Average Delay
0.15
0.1
0.05
0
1
2
3
4
5
Tx_Range
Average Delay


Delivery ratio
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
Tx_Range
Delivery ratio


Drop ratio
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
3
4
5
Tx_Range
Drop ratio