CSE322 Computer Network

NS2 Term Project

Implementation and Evaluation of TCP HyStart in NS2

[Taming the elephants: New TCP slow start - ScienceDirect]

Static 802.11

Mobile 802.15.4

HAZ Sameen Shahgir

1805053

Group A2

Motivation:

When operating under large bandwidth-delay product (BDP) networks, the conventional slow start technique proves to be ineffective. Standard TCP doubles the congestion window (cwnd) for every round-trip time (RTT) during slow start. However, the exponential growth ofcwnd results in burst packet losses. Since the cwnd overshoots the path capacity an amount as large as the entire BDP, in large BDP networks, this overshoot causes strong disruption in the networks.

The selective acknowledgement (SACK) option relieves this problem to some extent. As SACK informs the sender the blocks of packets successfully received, the sender can be more intelligent about recovering from multiple packet losses. However, for a large BDP network where a large number of packets are in flight, the processing overhead of SACK information at the end points can be overwhelming. This is because each SACK block invokes a search into the large packet buffers of the sender for the acknowledged packets in the block, and every recovery of a lost packet causes the same search at the receiver end. During fast recovery, every packet reception generates a new SACK packet. Given that the size of the cwnd can be quite large (sometimes, greater than 100,000), the overhead of such a search can be overwhelming.

As such, it is necessary to find a better alternative to the usual technique of waiting for a packet loss to exit slow start. Hystart is an addon to the typical slow start algorithm that exits slow start based on two heuristics, namely, how close the sum of the time intervals between acknowledgements is to the minimum delay time and on whether RTT is abruptly increased or not.

Description of Hystart Algorithm:

First Heuristic

ACK Train Detection

Let the unused available bandwidth of the forward path, the minimum forward path one-way delay and available buffer space of the forward path be B, D_{min} and S respectively. Then a safe exit point must be less than:

$$C = B \times D_{min} + S$$

$$C < B \times D_{min} = BDP$$

$$\Delta N = \sum_{1}^{N} t_k \text{ , } where\ \ t = time\ internal\ between\ packets,$$
$$N = Number\ of\ packets$$

$$estimated\ BDP = b(N) = \frac{(N-1) \times L}{\Delta N}$$

$$d_{min} = estimated\ D_{min}$$

$$estimated\ C = C^* = b(N) \times d_{min}$$

$$C^* = \frac{(N-1) \times L}{\Delta N} \times d_{min}$$

$$When\ the\ upperbound\ of\ the\ network\ has\ been\ reached\text{:}\ \ (N-1) \times L \to C^* \approx BDP$$

$$then\ \Delta N \to\ d_{min}$$

Second Heuristic

Abrupt Increase in RTT

The first n packets/their their ACK counterparts are sampled. If $RTT_k = RTT_{k-1} + \eta$ (where η is a constant), then another safe exit point has been found.

Code Modification:

tcp_cubic.c

ns2-2.35/tcp/linux.src

My Modifications over TCP Hystart

1. hystart_ack_delta_us was doubled (to 4000us) in order to trigger Hystart more often. This parameter controls the upper bound on the time difference between two consecutive ACK such that they are considered a part of an ACK train.

2. Current RTT is not bound by smoothed RTT (constant 100ms) since the 2nd heuristic checks whether the Current RTT is greater than Last RTT + constant

```c
static void hystart(struct sock *sk)
{
    struct tcp_sock *tp = tcp_sk(sk);
    struct bictcp *ca = inet_csk_ca(sk);
    u32 threshold;
    u32 now = bictcp_clock_us(sk);

    if (after(tp->snd_una, ca->end_seq)){
        hystart_reset(sk);
        return;
    }

    measure_delay(sk); //recalc ca->delay_min

    if ((s32)(now - ca->last_ack) <= hystart_ack_delta_us) {
        ca->last_ack = now;


        threshold = ca->delay_min>>1;


        if ((s32)(now - ca->round_start) > threshold) {
            printf("heu1\n");
            ca->found = 1;
            tp->snd_ssthresh = tp->snd_cwnd;
            return;
        }
    }

    if (ca->sample_cnt < HYSTART_MIN_SAMPLES) {
        ca->sample_cnt++;
    }
    else {

        if (ca->curr_rtt > ca->delay_min +
        HYSTART_DELAY_THRESH(ca->delay_min >> 3)) {
            printf("hue2\n");
            ca->found = 1;
            tp->snd_ssthresh = tp->snd_cwnd;
        }
    }
}
```

```
static inline u64 bictcp_clock_us(const struct sock *sk)
{
    return ktime_get_real/1000;
}

static inline void hystart_reset(struct sock *sk)
{
    struct tcp_sock *tp = tcp_sk(sk);
    struct bictcp *ca = inet_csk_ca(sk);
    ca->last_ack = bictcp_clock_us(sk);
    ca->round_start = bictcp_clock_us(sk);
    ca->end_seq = tp->snd_nxt;
    ca->curr_rtt = ~0U;
    ca->sample_cnt = 0;
}
```

```
static void bictcp_acked(struct sock *sk, u32 cnt,
                                        ktime_t last)
{
    ...

    // change 1
    if (!ca->found &&
    tp->snd_cwnd <= tp->snd_ssthresh){
        hystart(sk);
    }

}
```

Topology Setup:

Static

MAC Type: 802.11

DSDV

Single Random Source, Multiple Sink

Area: 500x500 m

Default Number of Nodes: 40

Default Number of Flows: 20

Default TX Multiplier: 1

Default Packets per Second: 100

Measurement Units:

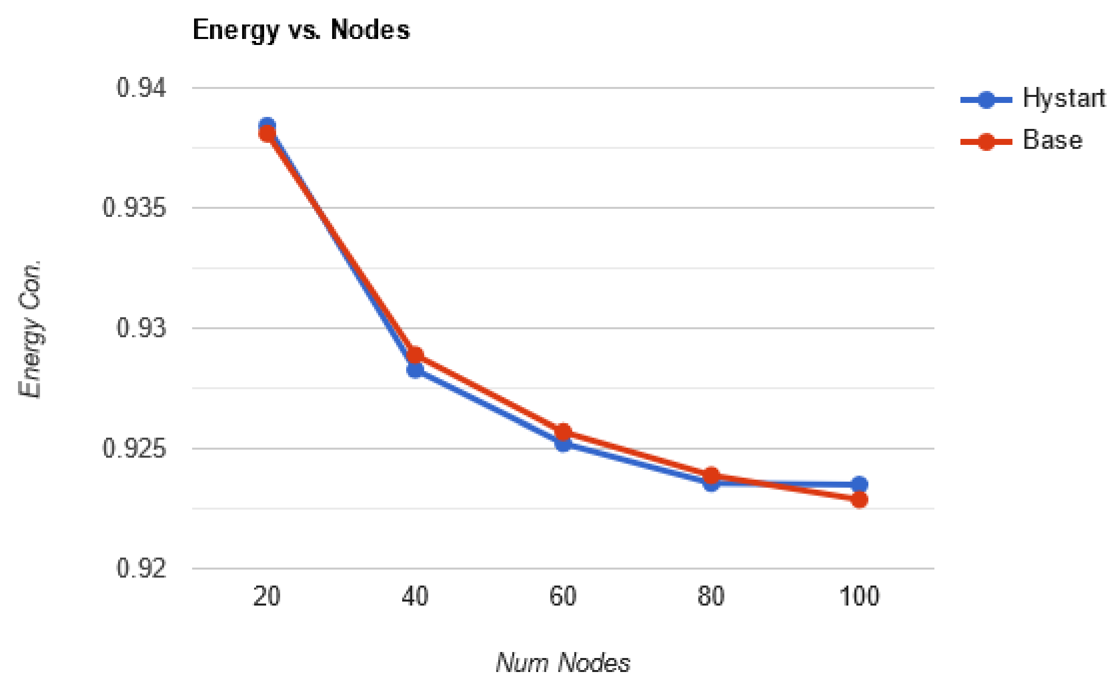
Energy Consumption: Js^{-1}

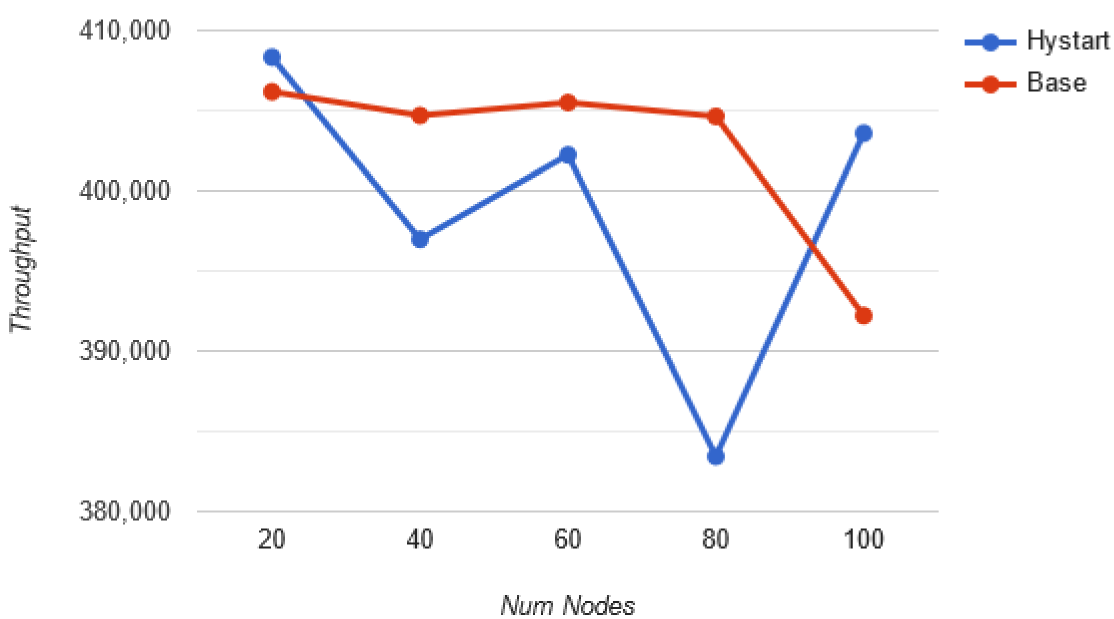
Throughput: bps

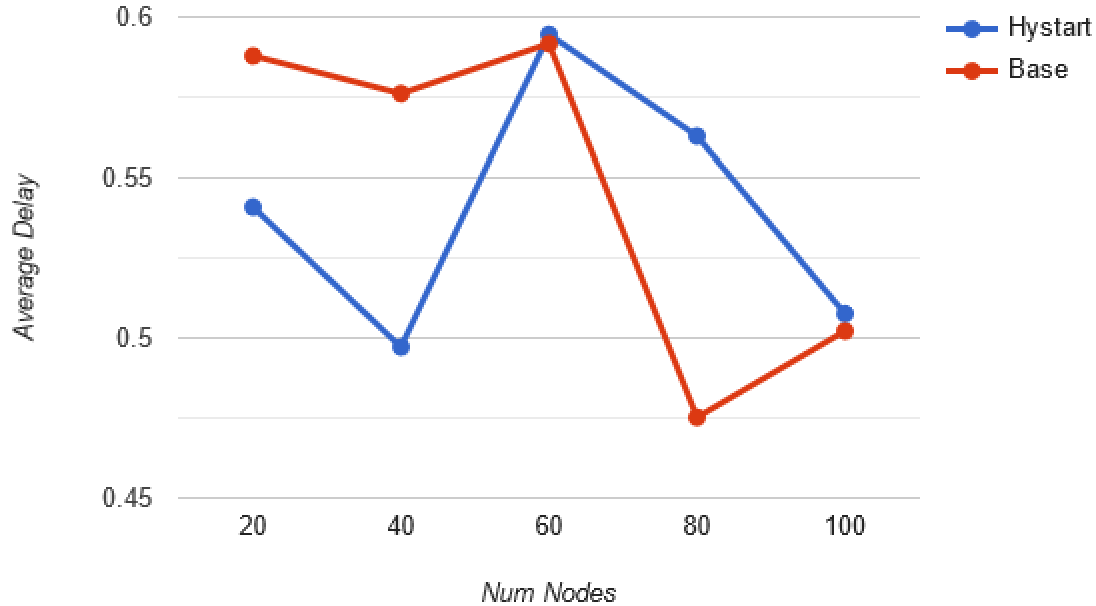
Avg. Delay: s

TX Range: (multiplier)

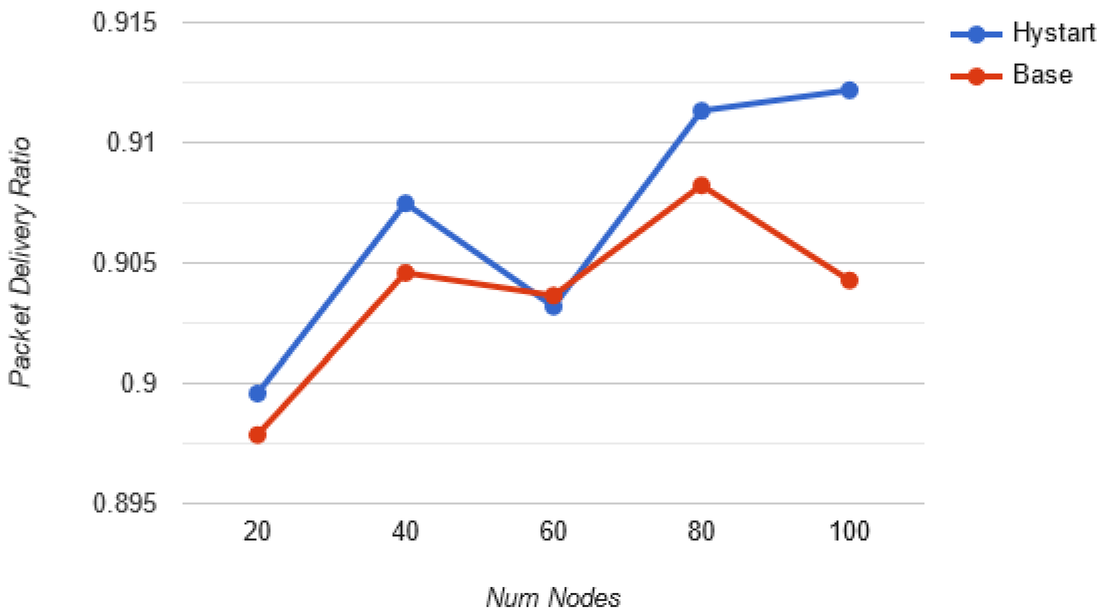
1. Metrics vs. Number of Nodes



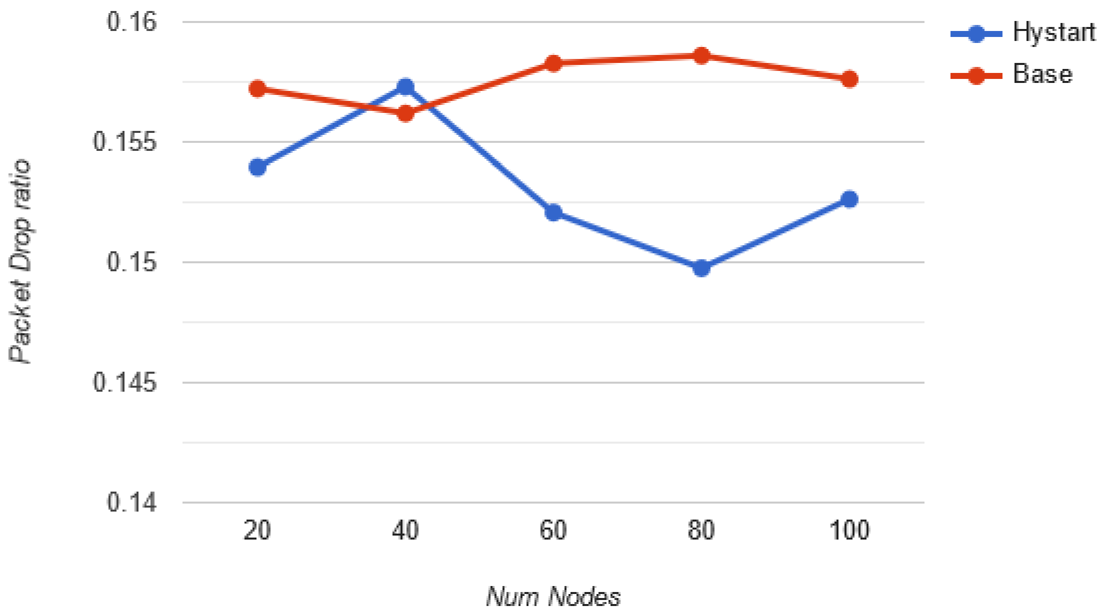




0.6
0.55
0.5
0.45
Average Delay
20
40
60
80
100
Num Nodes
Hystart
Base

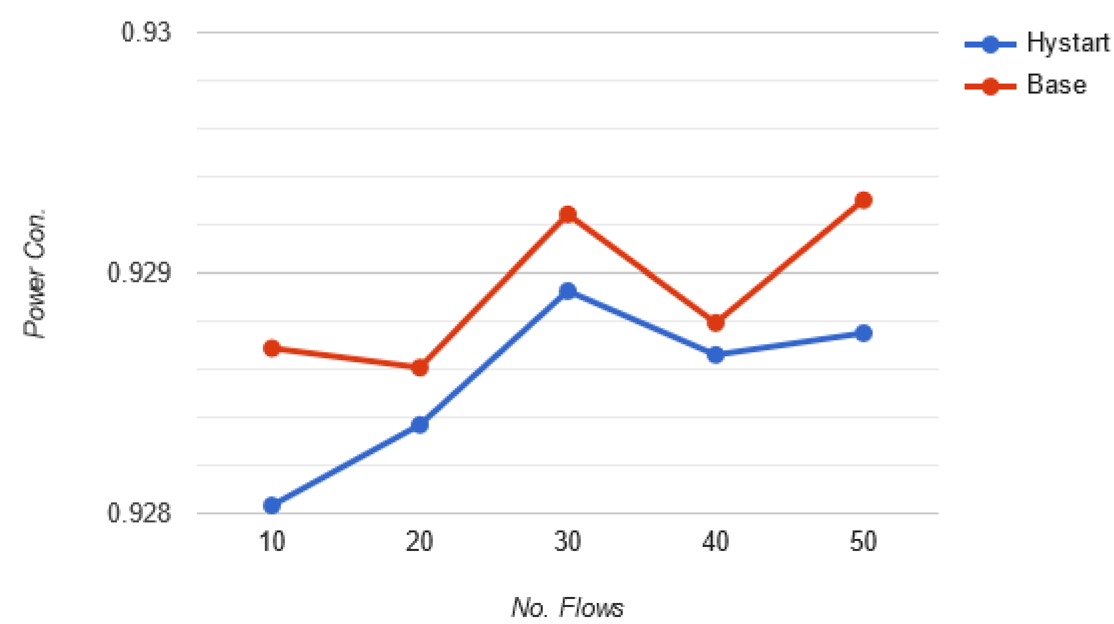


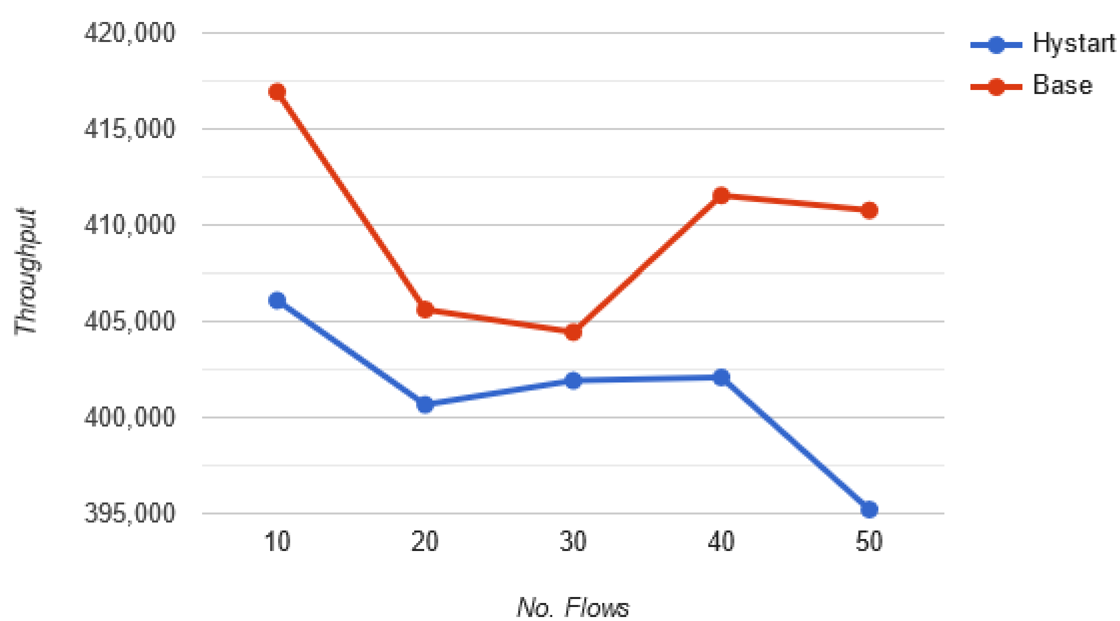
0.915
0.91
0.905
0.9
0.895
Packet Delivery Ratio
20
40
60
80
100
Num Nodes
Hystart
Base

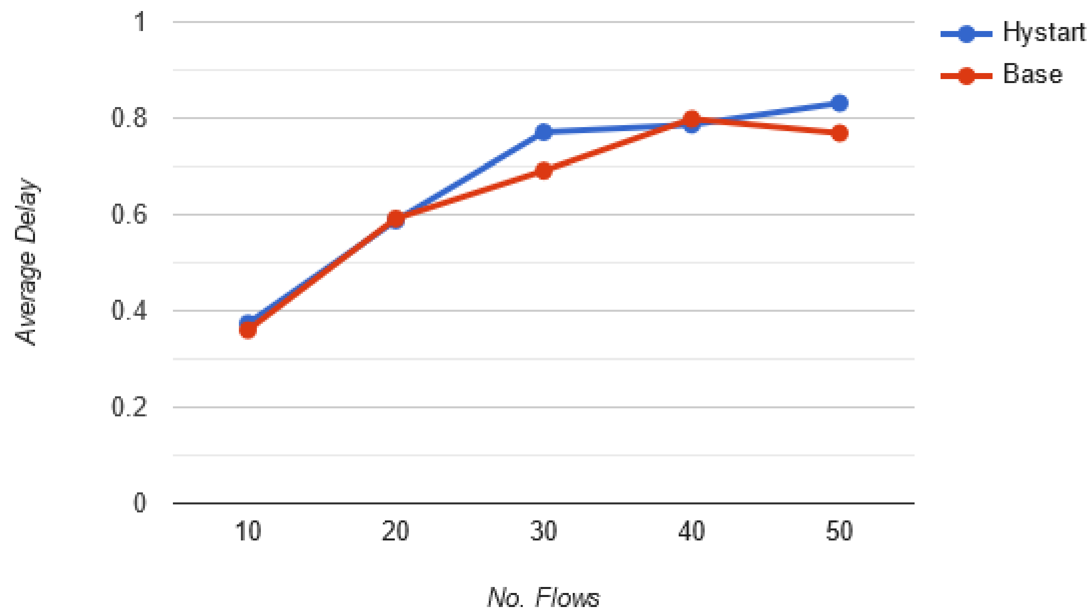


0.16
0.155
0.15
0.145
0.14
Packet Drop ratio
20
40
60
80
100
Num Nodes
Hystart
Base

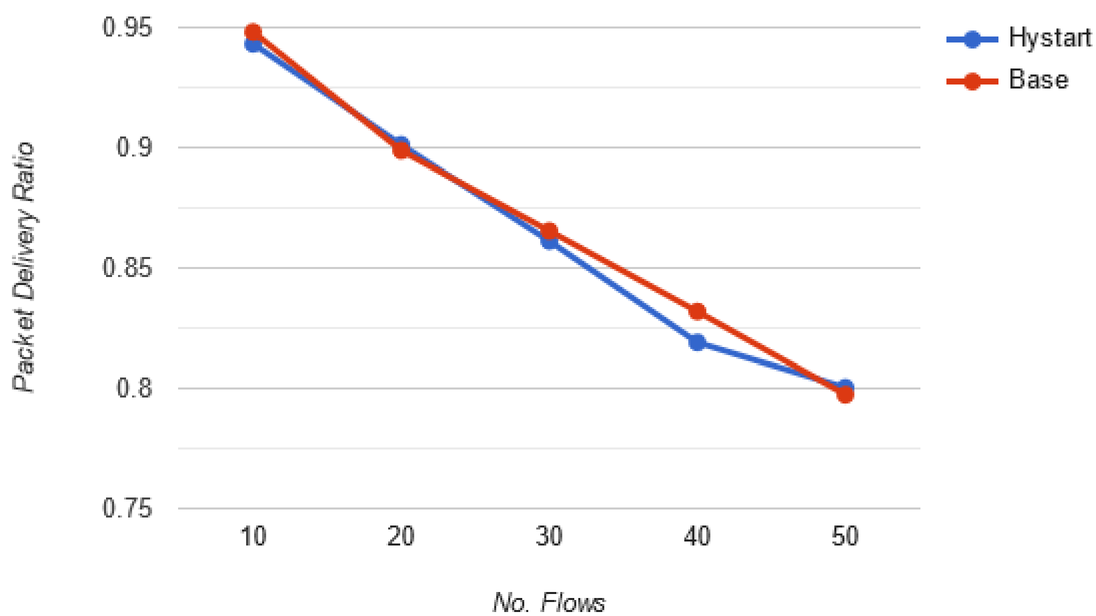
2. Metrics vs. Number of Flows



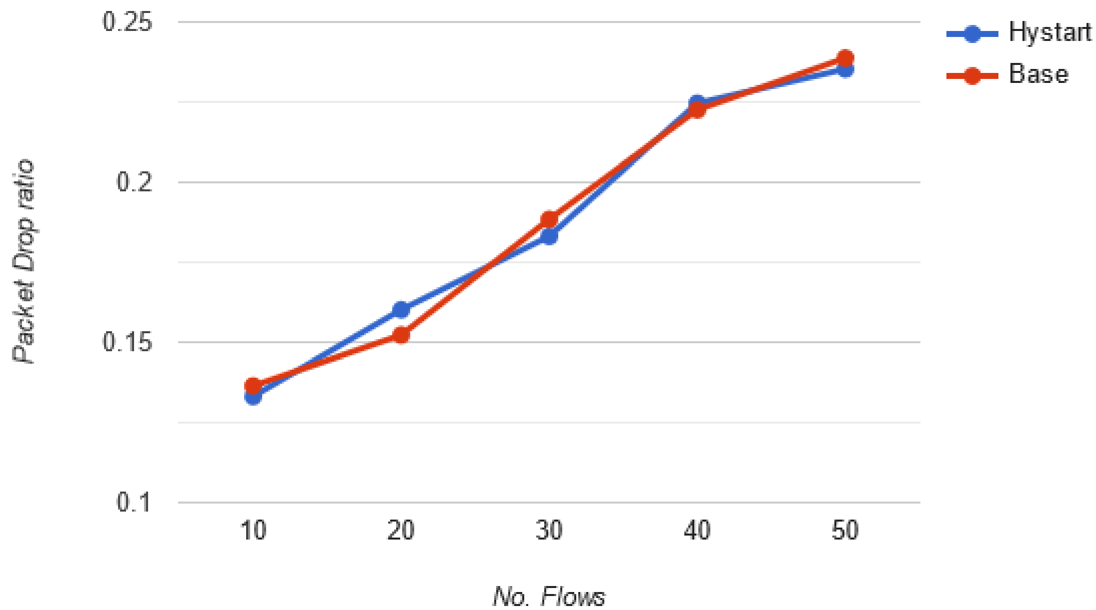




Average Delay
1
0.8
0.6
0.4
0.2
0
10
20
30
40
50
No. Flows
Hystart
Base

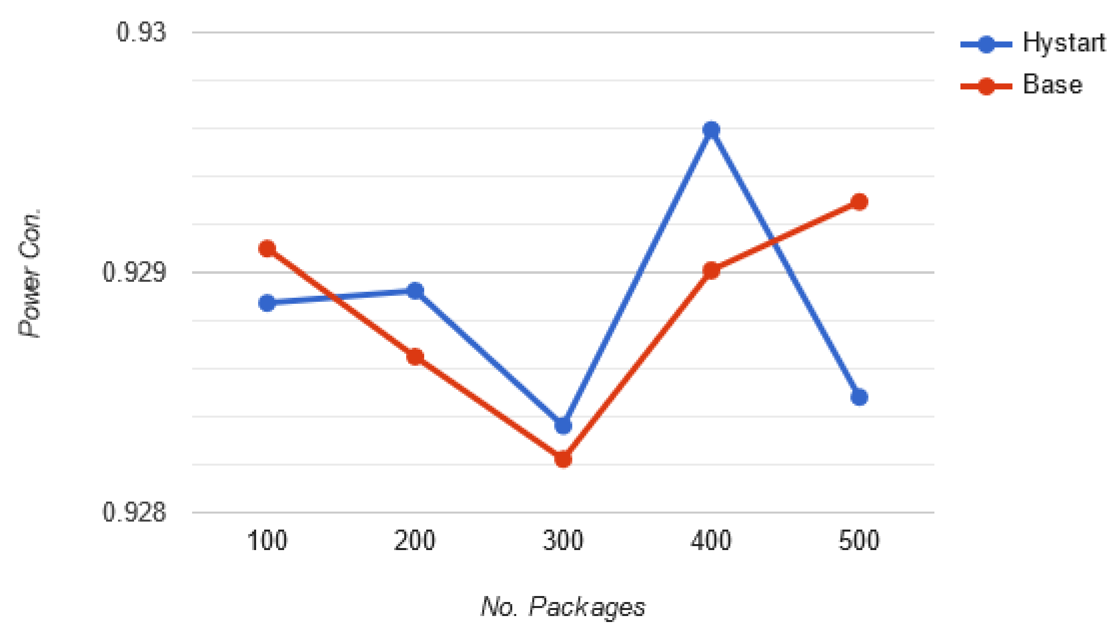


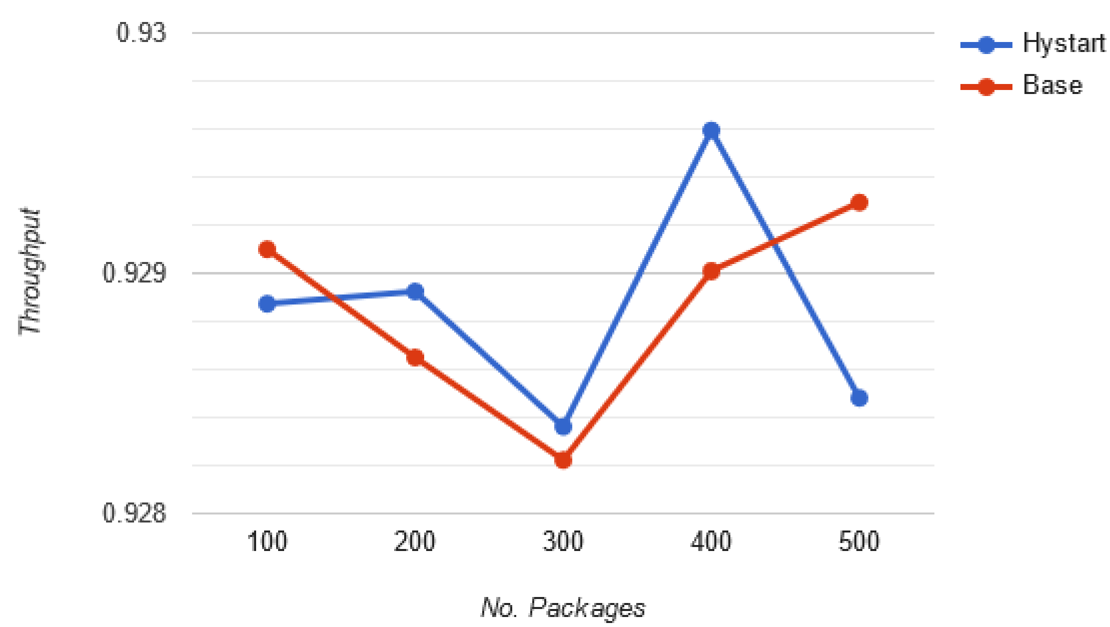
Packet Delivery Ratio
0.95
0.9
0.85
0.8
0.75
10
20
30
40
50
No. Flows
Hystart
Base

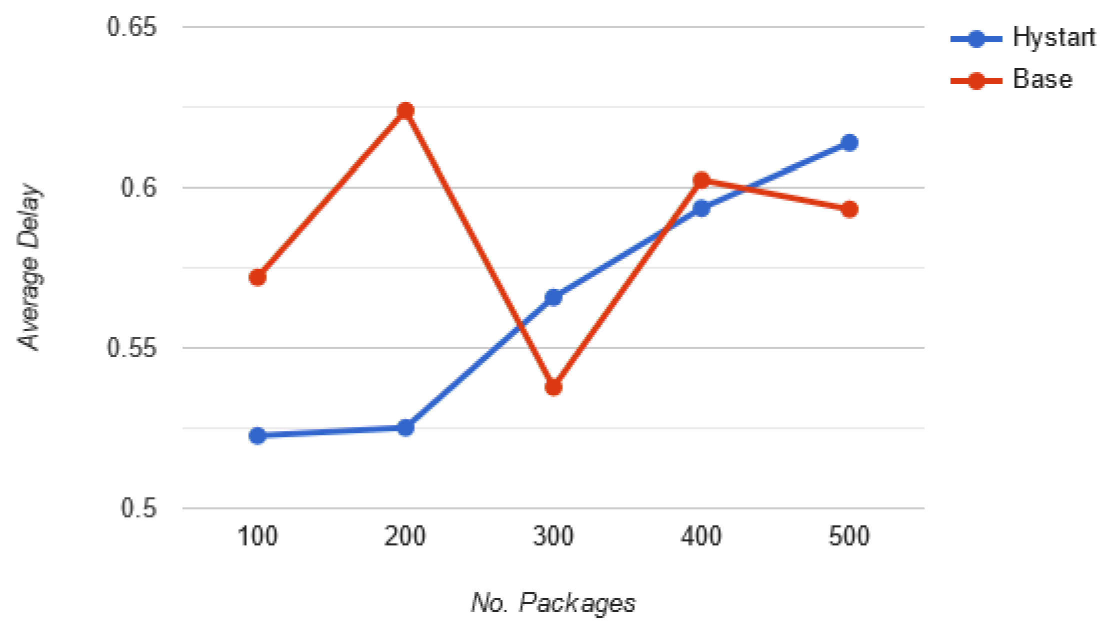


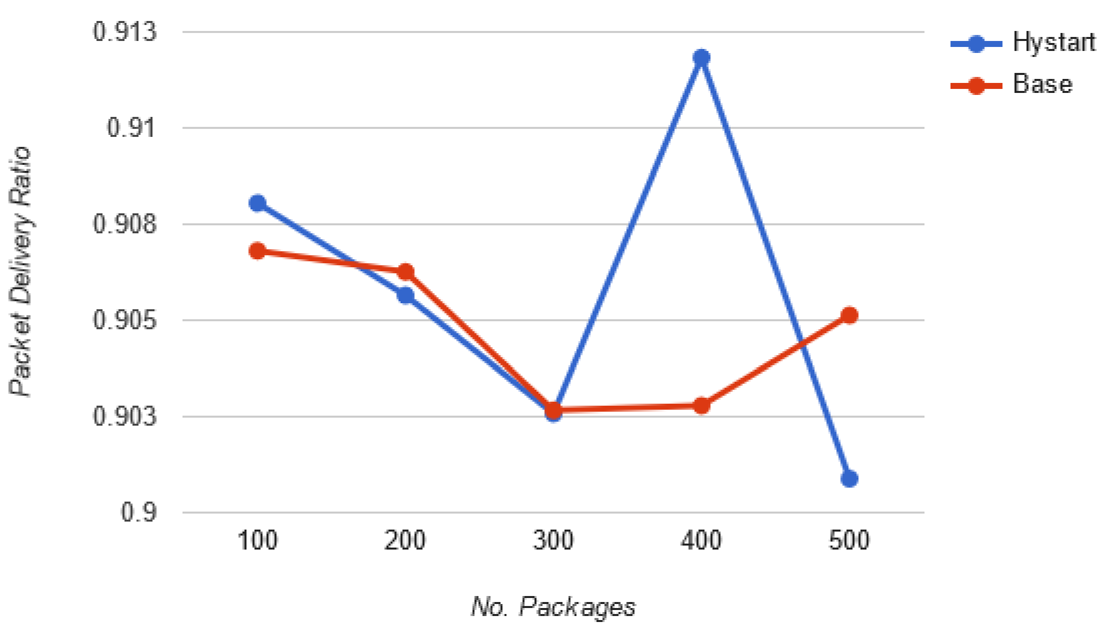
Hystart
Base
0.25
0.2
0.15
0.1
Packet Drop ratio
10
20
30
40
50
No. Flows

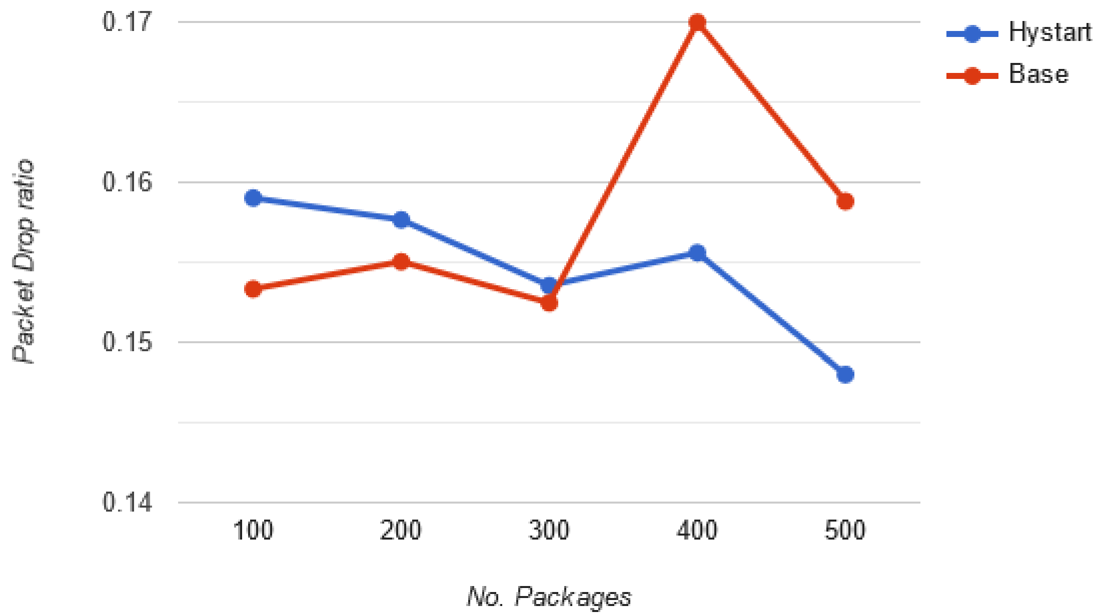
3. Metrics vs. Number of Packets per second





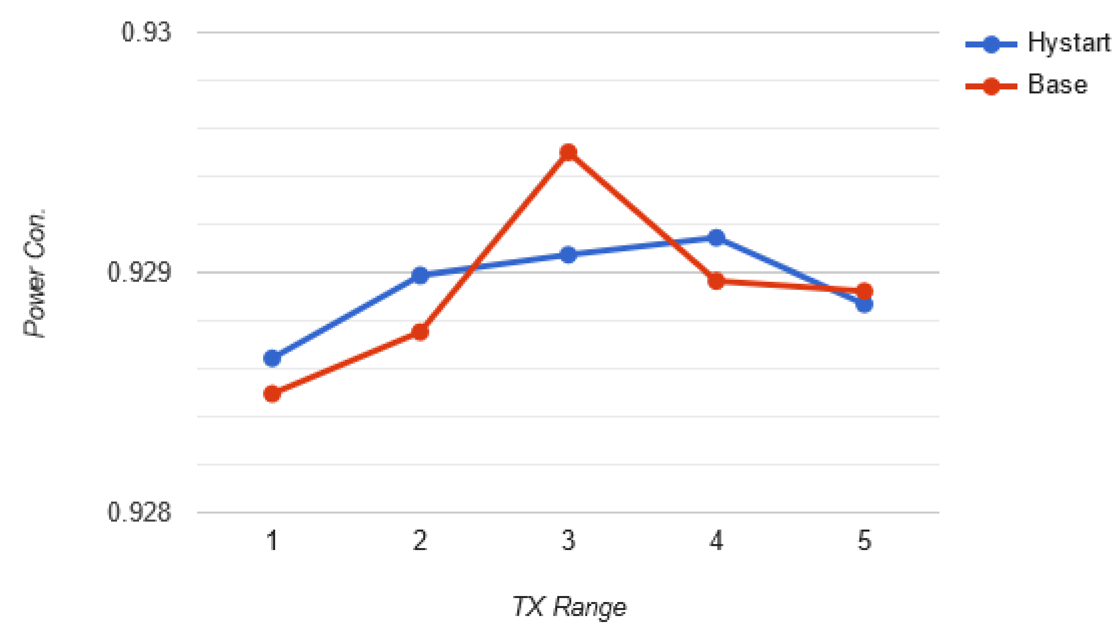

0.65
0.6
0.55
0.5
Average Delay
100
200
300
400
500
No. Packages
Hystart
Base

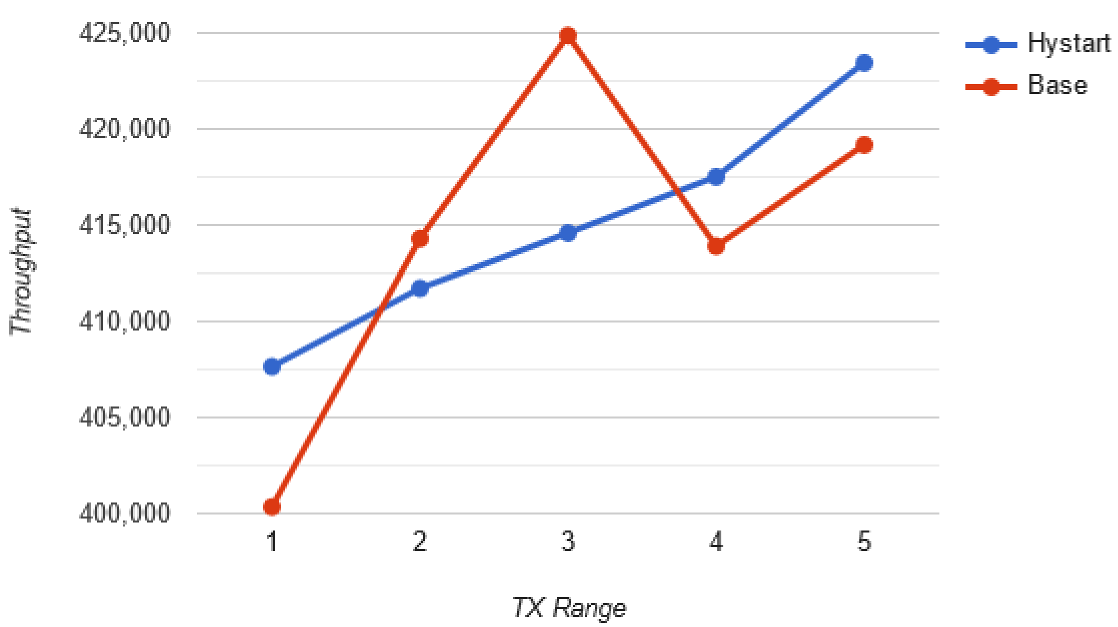

0.913
0.91
0.908
0.905
0.903
0.9
Packet Delivery Ratio
100
200
300
400
500
No. Packages
Hystart
Base

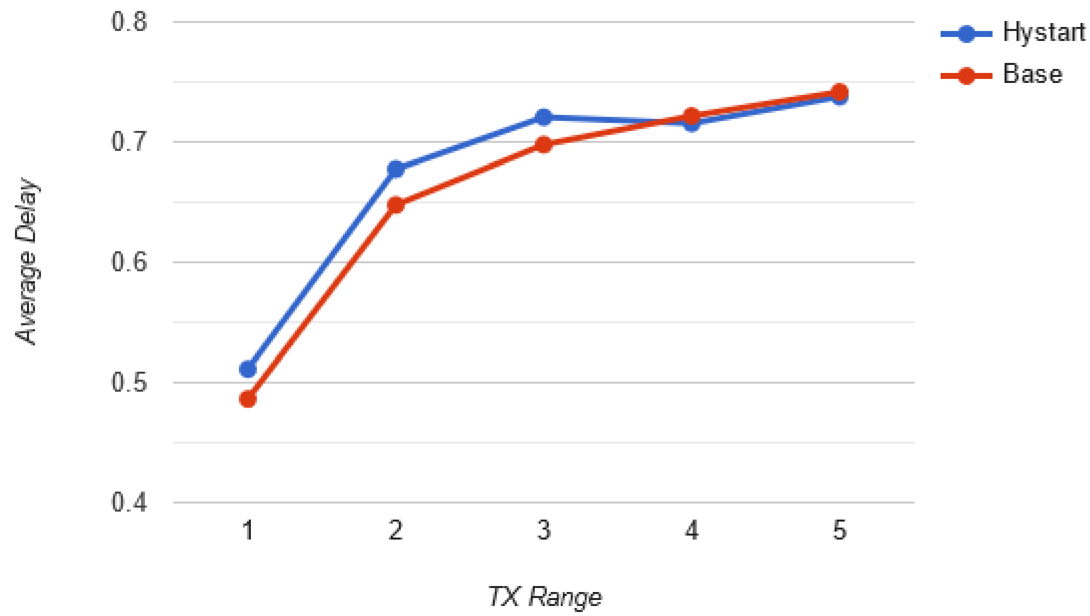


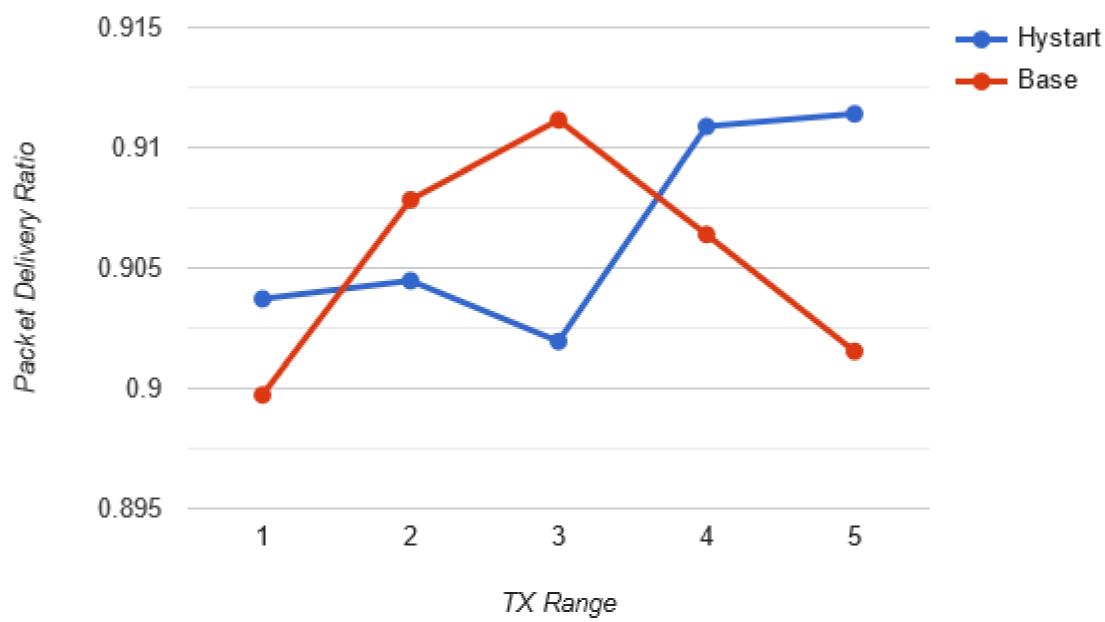
0.17
0.16
0.15
0.14
Packet Drop ratio
100
200
300
400
500
No. Packages
Hystart
Base

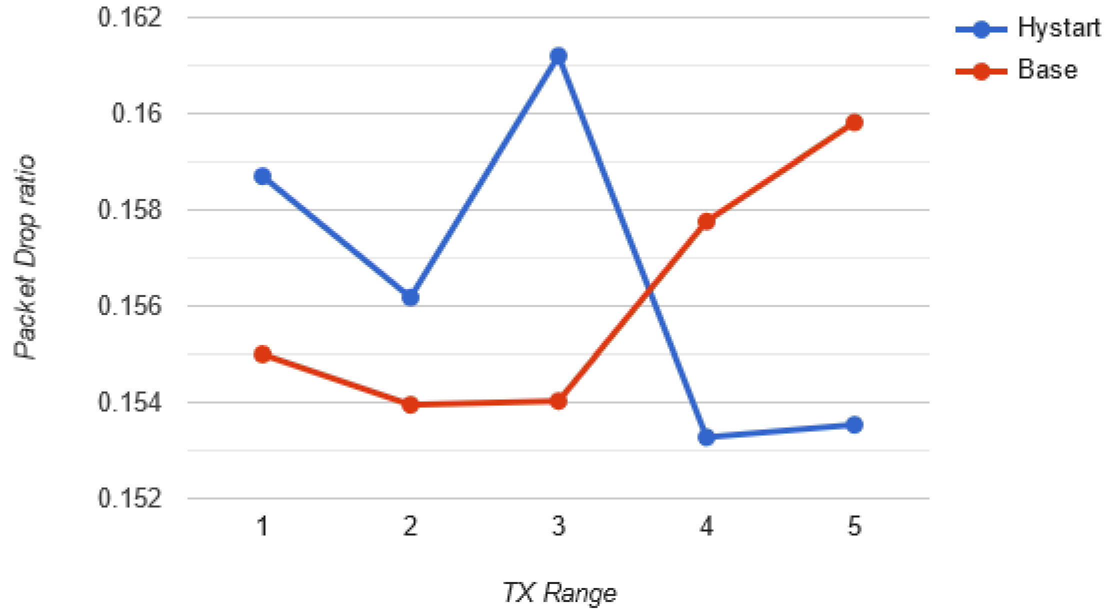
4. Metrics vs. TX Range






0.8
0.7
0.6
0.5
0.4
Average Delay
1
2
3
4
5
TX Range
Hystart
Base


0.915
0.91
0.905
0.9
0.895
Packet Delivery Ratio
1
2
3
4
5
TX Range
Hystart
Base



Hystart
Base
0.162
0.16
0.158
0.156
0.154
0.152
1
2
3
4
5
TX Range
Packet Drop ratio

Topology Setup:

Mobile

MAC Type: 802.15.4

DSDV

Single Random Source, Multiple Sink

Area: 500x500 m

Default Number of Nodes: 40

Default Number of Flows: 20

Default Speed: 10

Default Packets per Second: 100

Measurement Units:

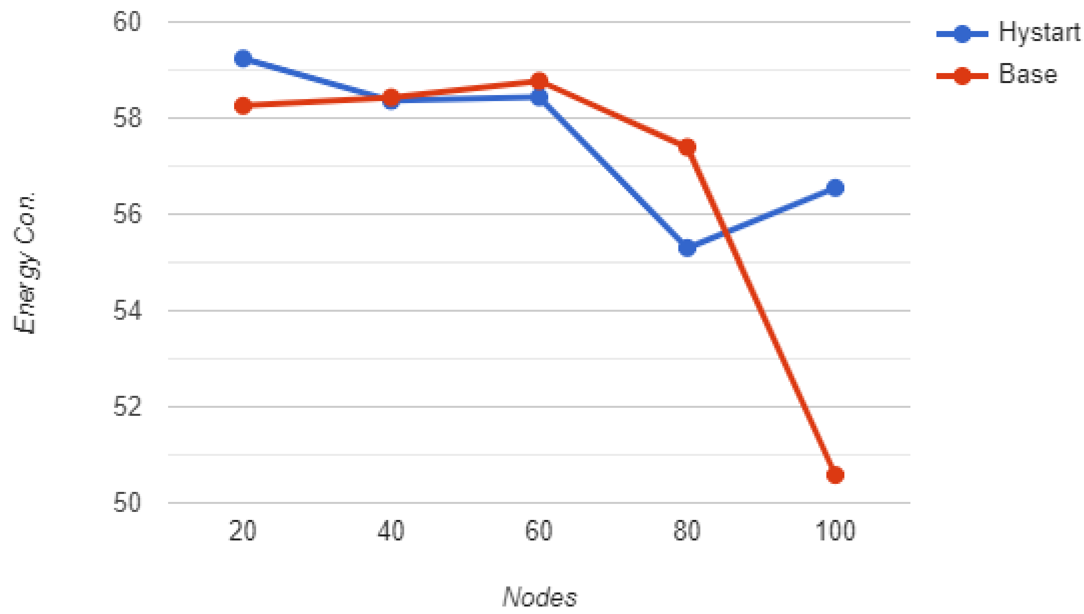
Energy Consumption: Js^{-1}

Throughput: bps

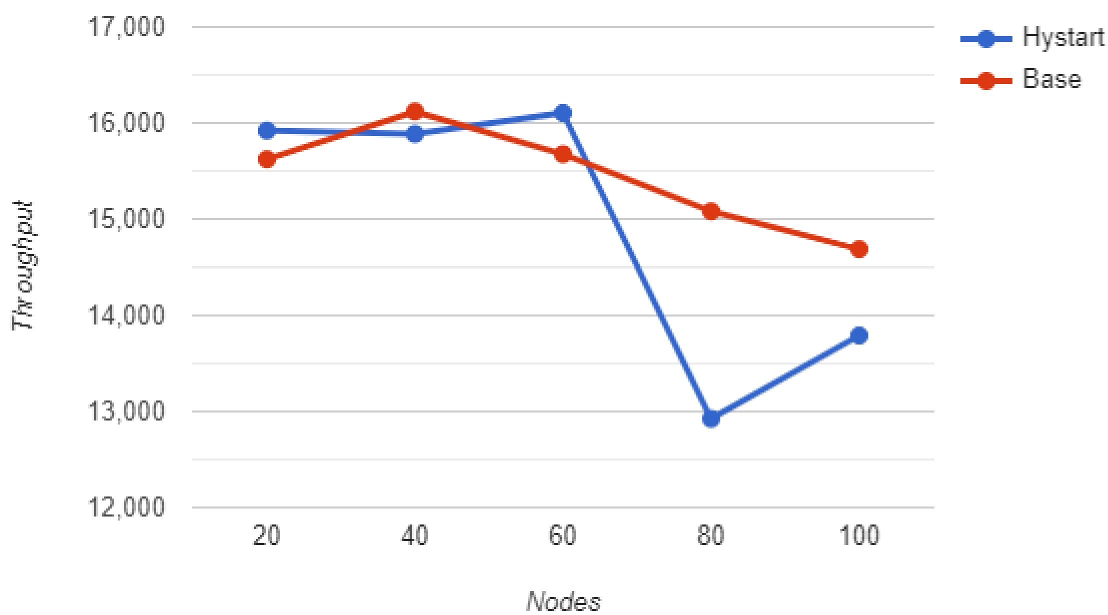
Avg. Delay: s

Speed: ms^{-1}

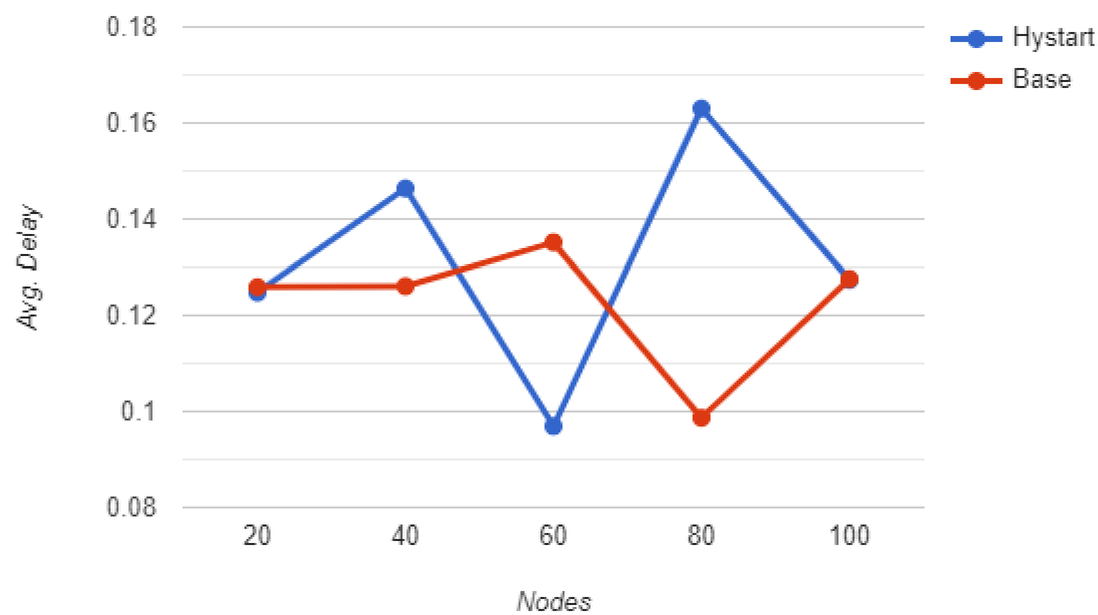
1. Metrics vs. Number of Nodes

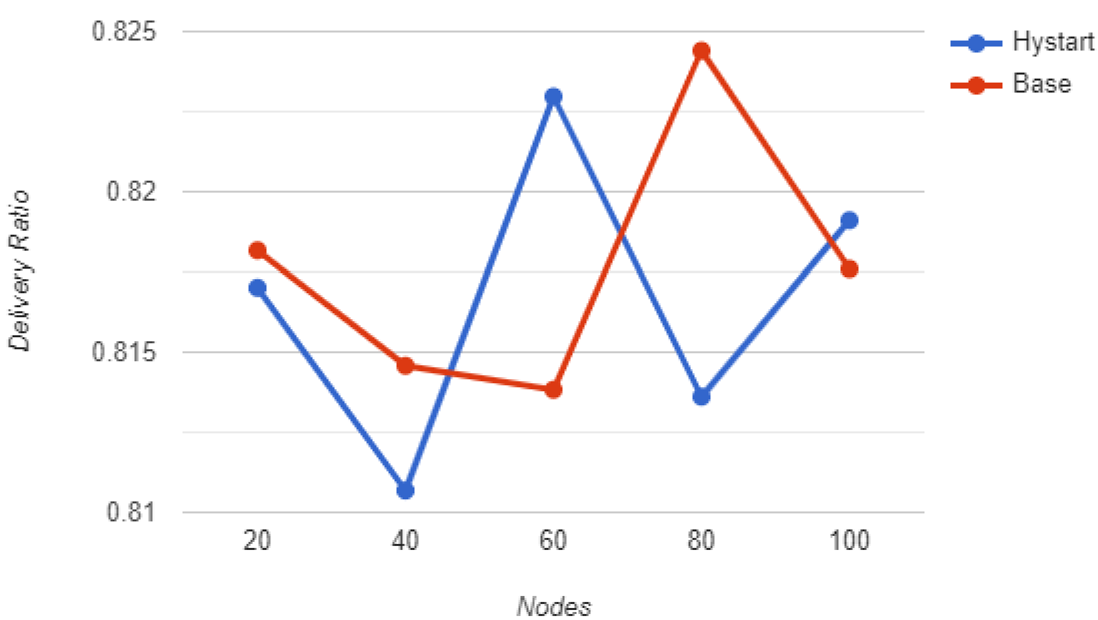


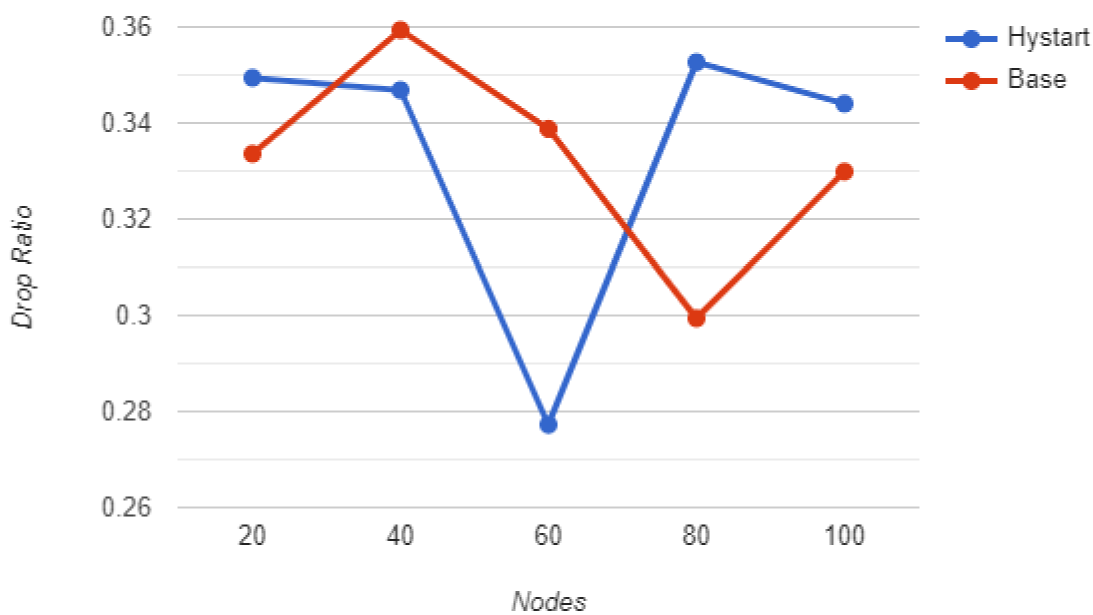
60
58
56
54
52
50
Energy Con.
20
40
60
80
100
Nodes
Hystart
Base



17,000
16,000
15,000
14,000
13,000
12,000
Throughput
20
40
60
80
100
Nodes
Hystart
Base

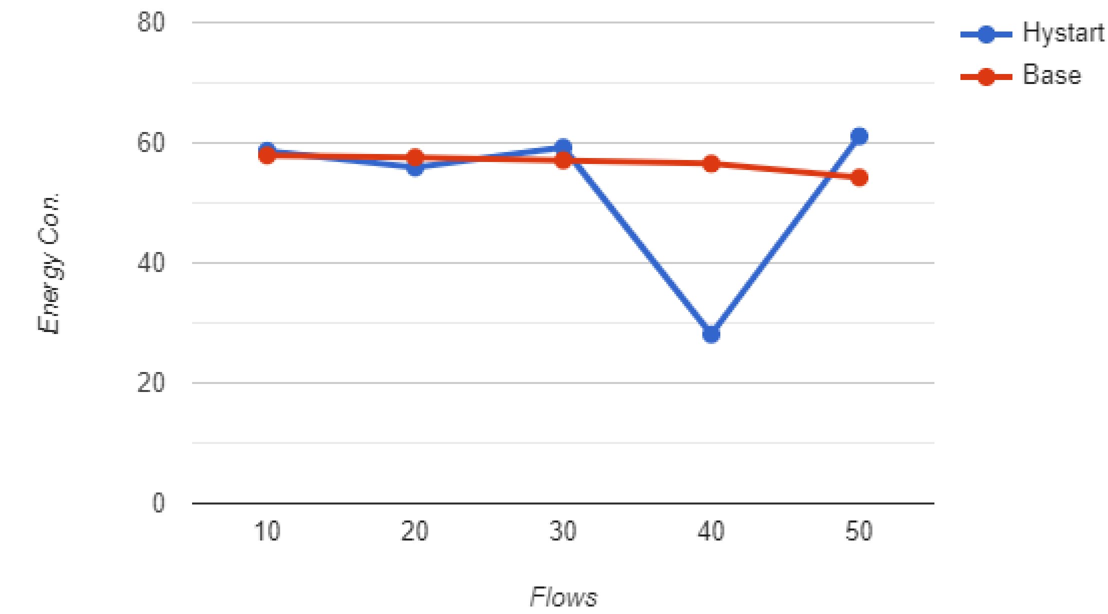

0.18
0.16
0.14
0.12
0.1
0.08
Avg. Delay
20
40
60
80
100
Nodes
Hystart
Base

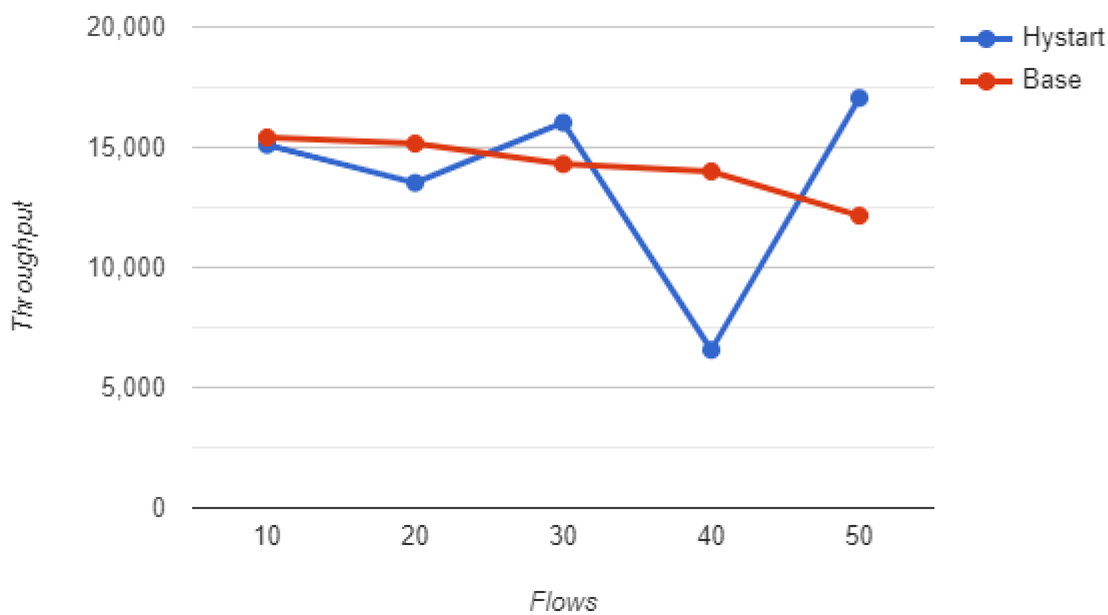

0.825
0.82
0.815
0.81
Delivery Ratio
20
40
60
80
100
Nodes
Hystart
Base

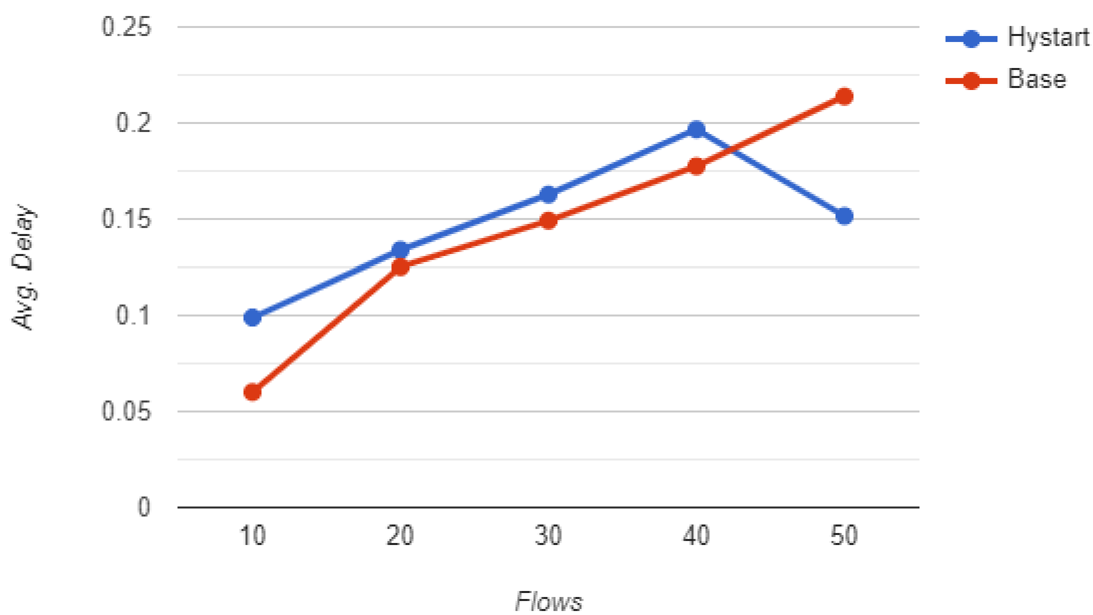


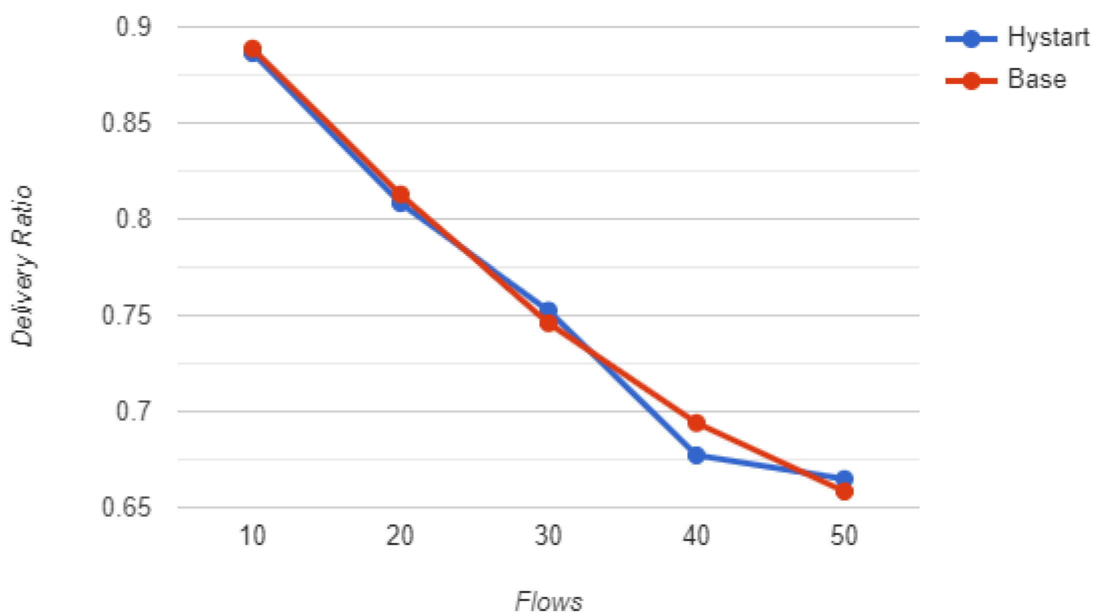
0.36
0.34
0.32
0.3
0.28
0.26
Drop Ratio
20
40
60
80
100
Nodes
Hystart
Base

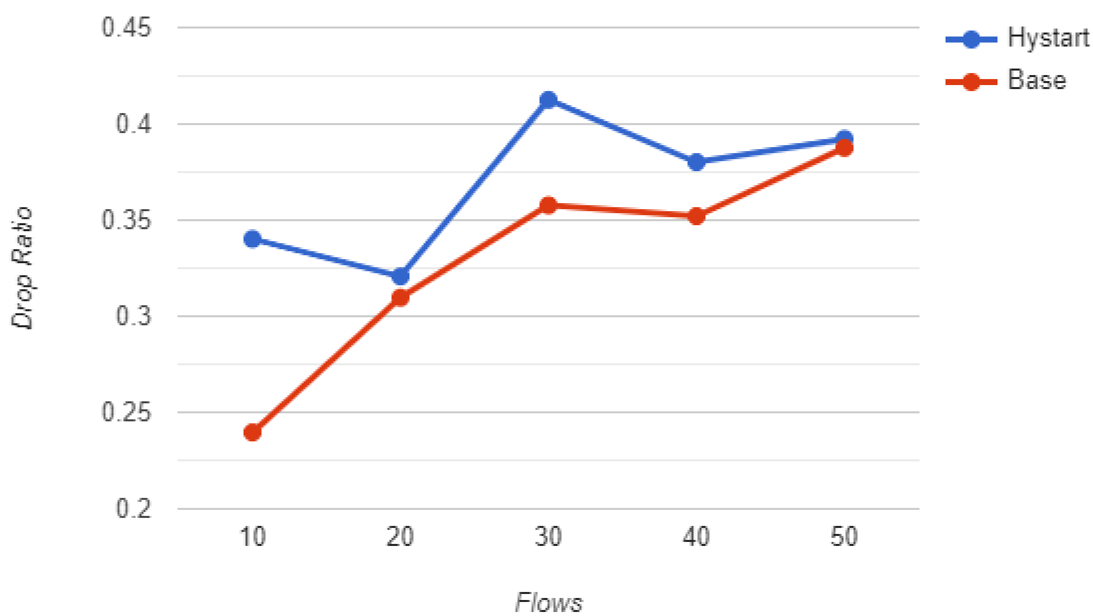
2. Metrics vs. Number of Flows





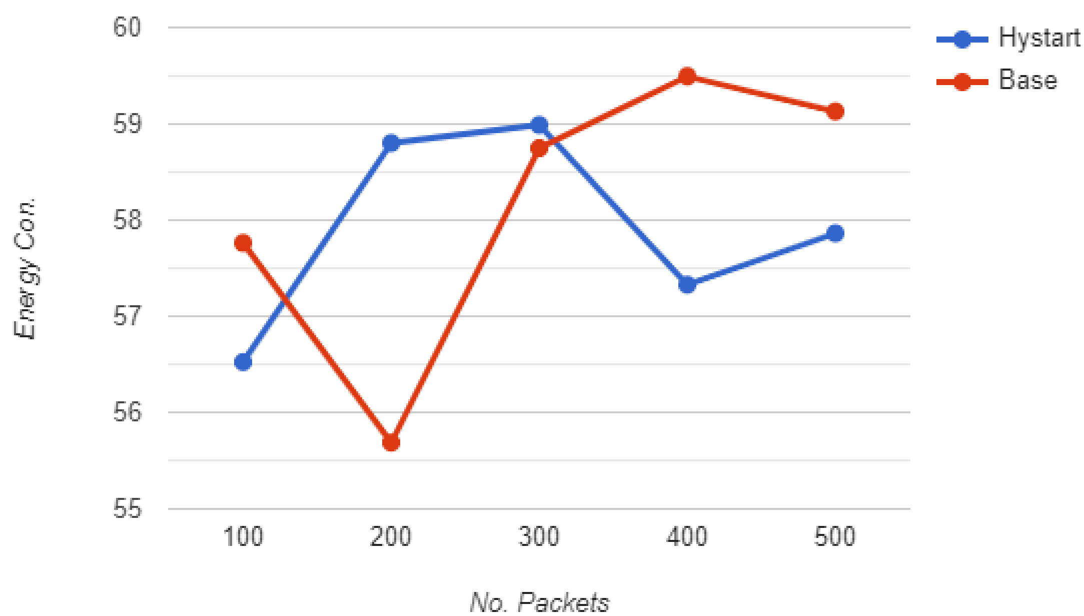

0.25
0.2
0.15
0.1
0.05
0
Avg. Delay
10
20
30
40
50
Flows
Hystart
Base

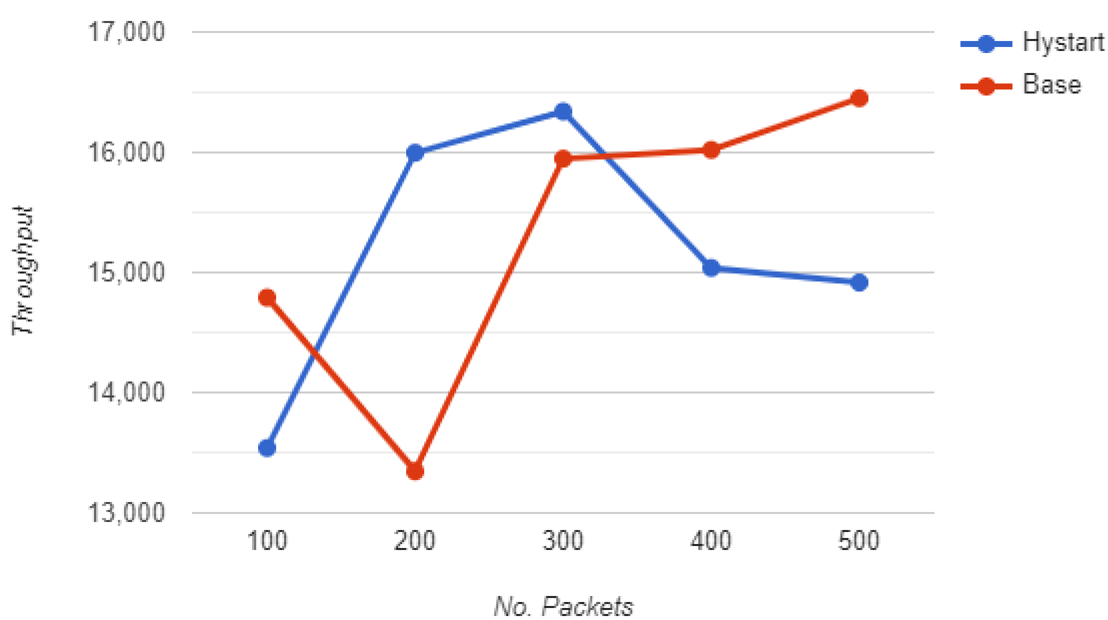

0.9
0.85
0.8
0.75
0.7
0.65
Delivery Ratio
10
20
30
40
50
Flows
Hystart
Base

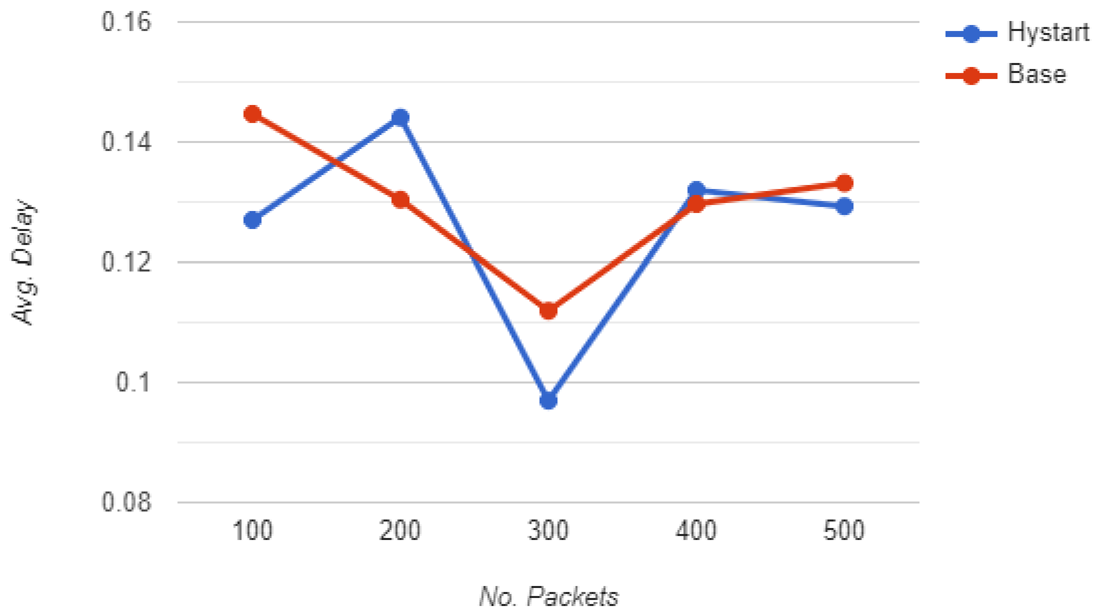


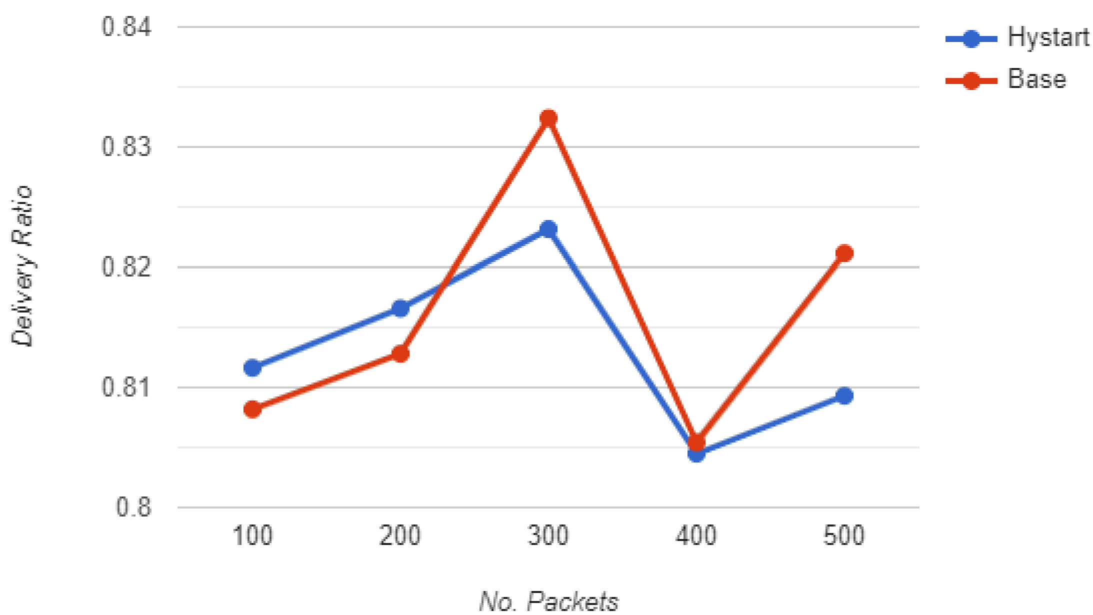
Hystart
Base
Drop Ratio
Flows
0.45
0.4
0.35
0.3
0.25
0.2
10
20
30
40
50

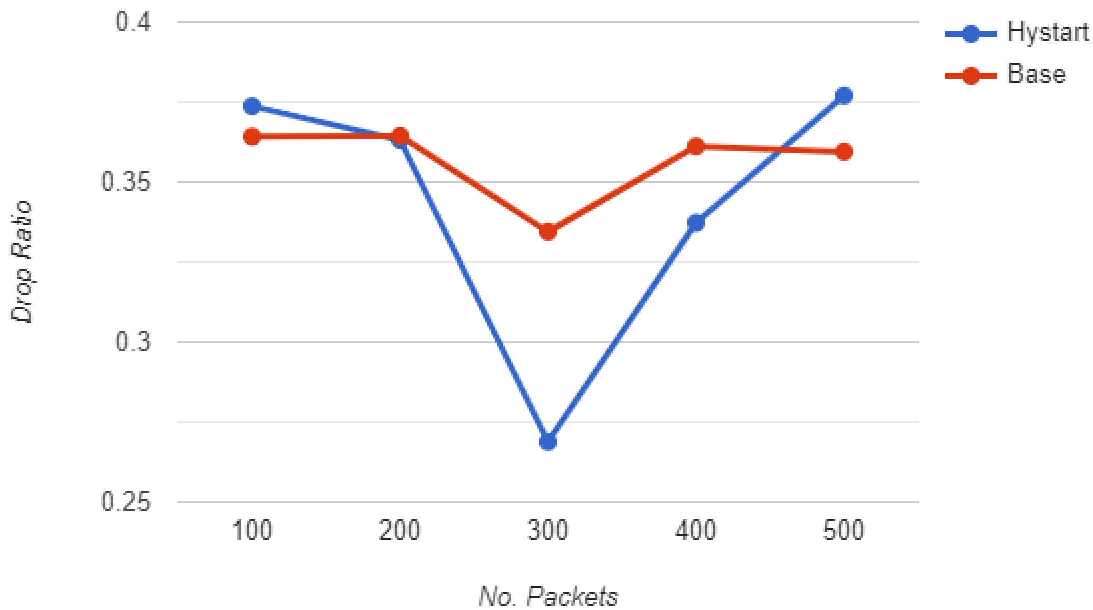
3. Metrics vs. Number of Packets Per Second





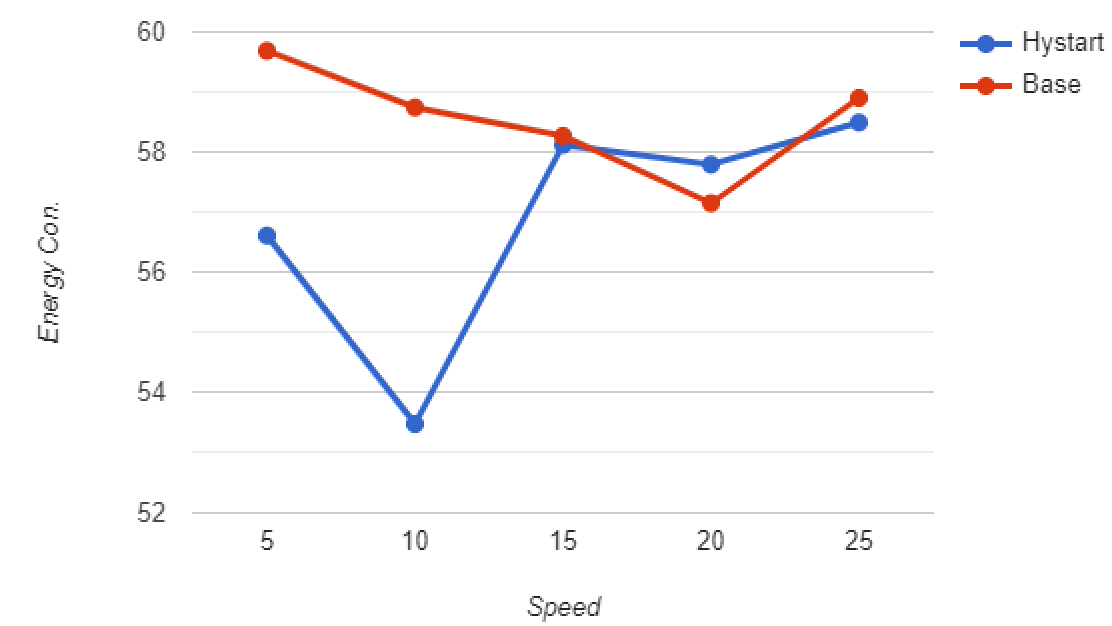

0.16
0.14
0.12
0.1
0.08
Avg. Delay
100
200
300
400
500
No. Packets
Hystart
Base

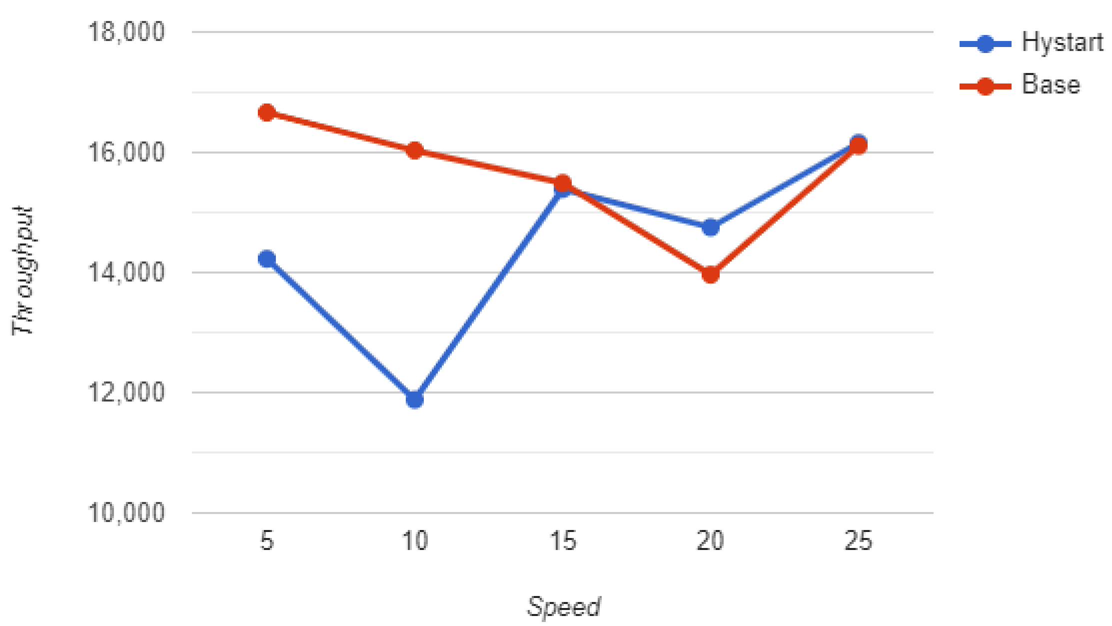

0.84
0.83
0.82
0.81
0.8
Delivery Ratio
100
200
300
400
500
No. Packets
Hystart
Base

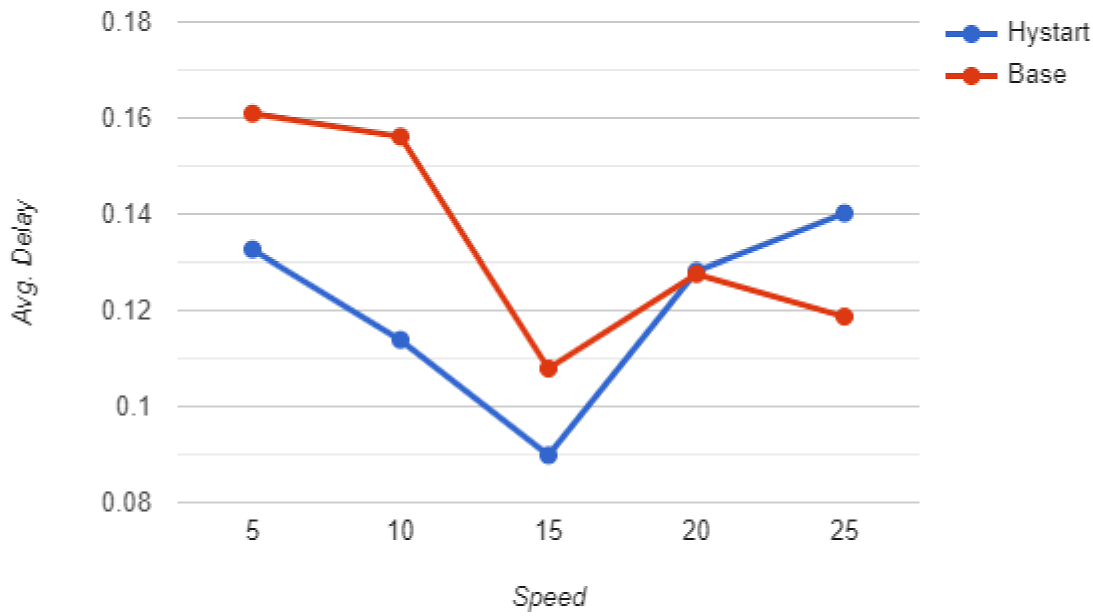


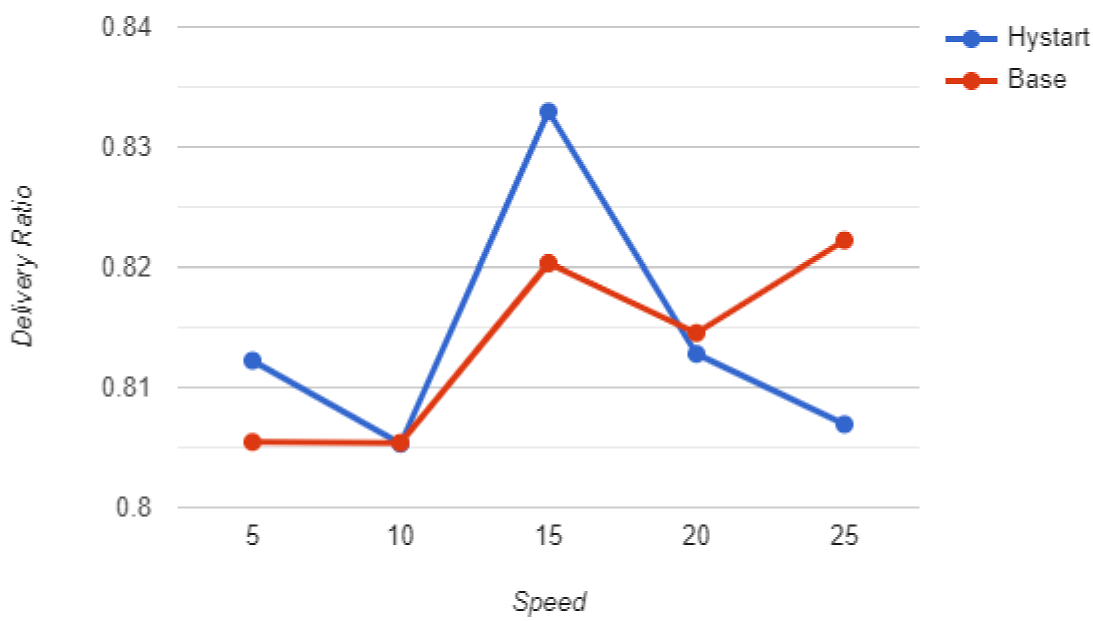
Hystart
Base
Drop Ratio
0.4
0.35
0.3
0.25
100
200
300
400
500
No. Packets

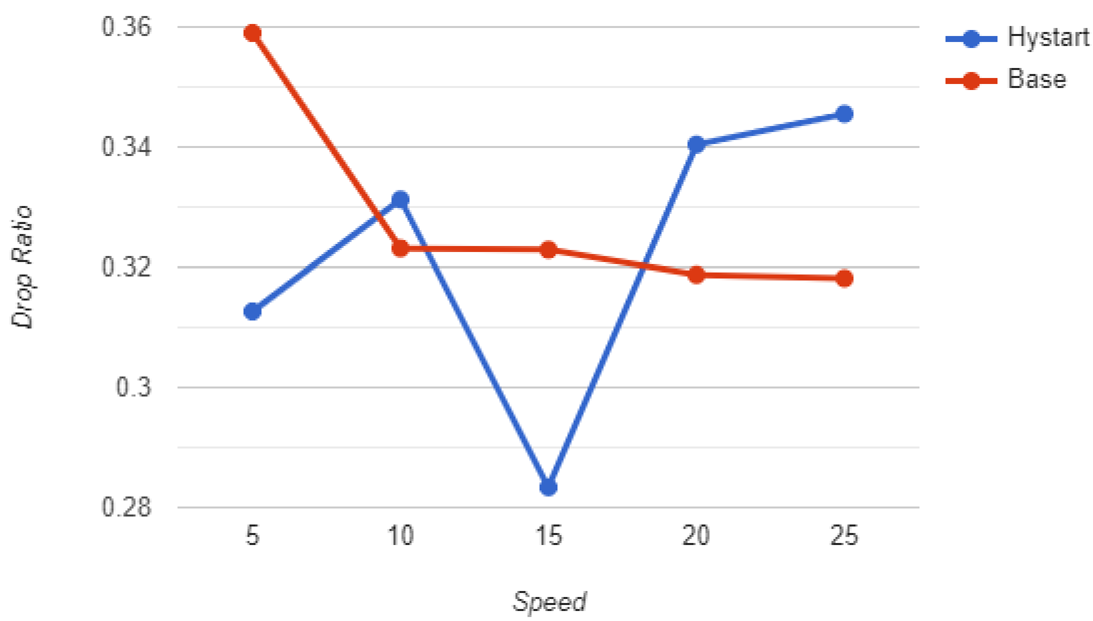
4. Metrics vs. Speed






0.18
0.16
0.14
0.12
0.1
0.08
Avg. Delay
5
10
15
20
25
Speed
Hystart
Base


0.84
0.83
0.82
0.81
0.8
Delivery Ratio
5
10
15
20
25
Speed
Hystart
Base



0.36
0.34
0.32
0.3
0.28
Drop Ratio
5
10
15
20
25
Speed
Hystart
Base

Summary Findings:

In 802.11

1. Energy consumption of TCP Hystart and base TCP Cubic are comparable in all cases.
2. Throughput is slightly but consistently lower in TCP Hystart when measured against Number of Nodes and Number of Flows. In other cases, it's comparable to base TCP Cubic.
3. Average Delay is comparable.
4. Delivery Ratio is improved noticably when compared.
5. There is a large improvement in Drop Ratio.
6. The comparative results of 50 iterations with the following values:

   Iterations = 50
   Area = 500x500 m
   Nodes = 40
   Flow = 20
   MAC = 802.11
   TX Range Multiplier = 3
   Application = FTP
   Packet Size = 1024 bytes
   Packet Interval = 0.005 sec.

Criteria	**Base**	**Hystart**	**%Difference**
Throughput	418016.244	417643.959	-0.08906
Average Delay	0.80988	0.798080	-1.4576
Delivery Ratio	0.81554	0.86891	**6.545**
Drop Ratio	0.2396	0.1906	**-20.434**

In 802.15.4

Setting MAC type to 802.15.4 in NS2 leads to packet_length_invalid_error. To remedy it, the following Energy Mode settings were necessary:

```
set val(initialenergy_15)  300.0
set val(idlepower_15)      40
set val(rxpower_15)        75
set val(txpower_15)        75
set val(sleeppower_15)     40
```

Even still, communication between nodes could not be guaranteed. Due to the random placement of nodes, the nodes were sometimes too far away to communicate. All readings were hence taken with the median of 10 observations, since taking the mean would misrepresent the actual findings.

There seems to be no apparent improvement by TCP Hystart when it comes to 802.15.4. This is likely due to the above-mentioned case where communication via 802.15.4 fails for distant nodes. This also has a knock-on effect on whether Hystart is triggered at all. Since the communication is so sparse, there is seldom an ACK train and hence the Hystart heuristics are rarely triggered. It was observed that a Hystart slow start exit was triggered about 1 in 10 simulations.

Bonus Work

1. New Measurement:
   per-node-throughput
2. Cross-Transmission (Attempted):
   wired_wireless.tcl
3. New Network:
   Satelite_iridium.tcl

Acknowledgement

Ha, Sangtae, and Injong Rhee. "Taming the elephants: New TCP slow start." *Computer Networks* 55.9 (2011): 2092-2110.