**Assignment**: To ​Develop a new example for PIE algorithm in ns-3

**Brief**: The implementation of PIE in ns-3 is being updated because the existing implementation is based on the original PIE paper and not on the RFC of PIE. The new implementation of the PIE will have many new components added in it. In this project, we need to develop a new example for PIE so that the user can utilize the components that are now being added in PIE.

Existing components are :

1. Random dropping
2. Drop probability calculation
3. Latency calculation
4. Burst tolerance

Add-ons components :

1. ECN support
2. Active-Inactive state
3. Dequeue rate estimation
4. Derandomization
5. Car drop adjustment

Value of knobs are also different in RFC.

The changed value are following:

|  |  |  |
| --- | --- | --- |
| KNOB | Value in Paper | Value in RFC |
| T\_Update | 30 ms | 16 ms |
| MAX\_burst | 200ms | 150ms |
| deque\_thres | 10kB | 16kB |
| qdelay\_ref | 20ms | 15 ms |
| averaging\_rate | 0.5 | 0.125 |

**ECN support:**-

PIE may enable marking ECN enable packets instead of dropping.A new threshold max\_ecnth is introduced (default value=0.1). If Drop\_prob is less than max\_ecnth marking is done else dropping state.

if PIE->drop\_prob\_ < mark\_ecnth && ecn\_capable\_packet == TRUE:  
 mark packet;  
 else  
 drop packet;

**Dequeue Rate estimation:**

If one chooses to implement the departure rate estimation, PIE uses a

counter to keep track of the number of bytes departed for the current interval. This counter is incremented per packet departure. Every

T\_UPDATE, PIE calculates latency using the departure rate, which can

be implemented using a single multiply operation.

\* Upon a packet dequeue:  
  
 if PIE->in\_measurement\_ == FALSE and queue.byte\_length() >=  
 DQ\_THRESHOLD:  
 PIE->in\_measurement\_ = TRUE;  
 PIE->measurement\_start\_ = now;  
 PIE->dq\_count\_ = 0;  
  
 if PIE->in\_measurement\_ == TRUE:  
 PIE->dq\_count\_ = PIE->dq\_count\_ + deque\_pkt\_size;  
 if PIE->dq\_count\_ >= DQ\_THRESHOLD then  
 weight = DQ\_THRESHOLD/2^16  
 PIE->avg\_dq\_time\_ = (now - PIE->measurement\_start\_) \*  
 weight + PIE->avg\_dq\_time\_ \*  
 (1 - weight);  
 PIE->dq\_count\_ = 0;  
 PIE->measurement\_start\_ = now  
 else  
 PIE->in\_measurement\_ = FALSE;

**Active /Inactive state:**

PIE has an optional feature of automatically becoming

active/inactive. To implement this feature, the PIE may choose to only

become active (from inactive) when the buffer occupancy is over a

certain threshold, which may be set to 1/3 of the tail drop

threshold. PIE becomes inactive when congestion ends; i.e., when the

drop probability reaches 0, current and previous latency samples are

all below half of QDELAY\_REF.

Upon packet arrival:  
  
 if PIE->active\_ == FALSE && queue\_length >= TAIL\_DROP/3:  
 PIE->active\_ = TRUE;  
 PIE->burst\_allowance\_ = MAX\_BURST;  
  
 if PIE->burst\_allowance\_ > 0  
 enqueue packet;  
 else  
 randomly drop a packet with a probability of  
 PIE->drop\_prob\_.  
  
 if (PIE->drop\_prob\_ == 0 and current\_qdelay < QDELAY\_REF/2 and  
 PIE->qdelay\_old\_ < QDELAY\_REF/2)  
 PIE->active\_ = FALSE;  
 PIE->burst\_allowance\_ = MAX\_BURST;  
  
 \* "Drop probability calculation" block: PIE does the following:  
 if PIE->active\_ == TRUE:  
 PIE->burst\_allowance\_ =  
 max(0,PIE->burst\_allowance\_ - T\_UPDATE);

**Derandomization:**

PIE support random dropping of packets, sometimes packets are dropped too close to each other or too far from each other. In certain

scenarios, such as a small number of simultaneous TCP flows, these deviations can cause significant deviations in link utilization and queuing latency.

A parameter called "PIE->accu\_prob\_" is reset to 0 after a drop. Upon packet arrival, PIE->accu\_prob\_ is incremented by the amount of drop probability, PIE->drop\_prob\_.

If PIE->accu\_prob\_< 0.85

the arriving packet is enqueued;

Else if( PIE->accu\_prob\_ >8.5),

and the queue is congested, the arrival packet is forced to be dropped;

else

The packet is randomly dropped.

**Cap Drop Adjustment:**

In the case of a single TCP flow, during the slow-start phase the

queue could quickly increase, which could result in a very rapid

increase in drop probability. In order to prevent an excessive

ramp-up that could negatively impact the throughput in this scenario,

PIE can cap the maximum drop probability increase in each step.

if (PIE->drop\_prob\_ >= 0.1 && p > 0.02) {  
 p = 0.02;  
 }