

Verification of Network Congestion Avoidance Algorithm like WRED using VMM

Puja Sethia

eInfochips
Ahmedabad, India

www.einfochips.com

ABSTRACT

Computer networks have experienced an explosive growth over the past few years and with that growth have come severe congestion problems. There are many different network congestion avoidance mechanisms. Weighted Random Early Detection (WRED) is one of them. This paper describes WRED verification techniques and covers the verification architecture for verifying WRED Algorithm. When verifying WRED algorithm, rather than using random based techniques the focus is more on generating specific types of stimulus. This paper covers the usage of VMM Single Stream Scenarios, VMM Multi-Stream Scenarios and VMM Scoreboard with-loss packet feature and explains how these VMM features can make the complex verification environment required for WRED verification, simple and reusable.

Table of Contents

1	Introduction	3
1.1	HOW WRED WORKS?	3
1.2	WRED ALGORITHM.....	3
1.3	WRED EXAMPLE.....	4
2	WRED Verification	5
2.1	WRED VERIFICATION APPROACH.....	6
2.1.1	<i>Stimulus Generation Requirements</i>	7
2.1.2	<i>Map WRED Stimulus Requirements with VMM Scenarios</i>	9
2.2	PREDICT AND VERIFY WRED RESULTS	14
2.2.1	<i>Prediction of WRED Results</i>	14
2.2.2	<i>Verify WRED Results</i>	15
2.3	DOES RESULTS MAKE SENSE?.....	17
3	Conclusions	18
4	References	19

Table of Figures

Figure 1	– WRED Algorithm Flow Diagram	3
Figure 2	– WRED Packet Drop Probability	4
Figure 3	– WRED Increment Drop Profile.....	5
Figure 4	– WRED Verification Environment.....	6
Figure 5	– WRED Pass and Drop Region	8
Figure 6	– Mapping WRED Stimulus requirements with VMM Components	10
Figure 7	– WRED Stimulus Scenarios.....	11
Figure 8	– WRED Packet Single Stream Scenario	12
Figure 9	– WRED Base Multi-Stream Scenario	13
Figure 10	– WRED Test Multi-Stream Scenario	14
Figure 11	– WRED Predicted Result	15
Figure 12	– WRED Predicted and Actual Result	16
Figure 13	– WRED Scoreboard.....	17
Figure 14	– WRED Result Graph – Drop Rate increasing linearly with queue size	18

Table of Tables

Table 1	– WRED Increment Drop Profile	5
Table 2	– WRED Pass and Drop Region	8

1 Introduction

Congestion avoidance techniques monitor network traffic loads in an effort to anticipate and avoid congestion at common network and inter-network bottlenecks before it becomes a problem. These techniques are designed to provide preferential treatment for premium (priority) class traffic under congestion situations while concurrently maximizing network throughput and capacity utilization and minimizing packet loss and delay. Congestion avoidance is achieved through packet dropping. Among the more commonly used congestion avoidance mechanisms is Weighted Random Early Detection (WRED), which is optimum for high-speed transit networks.

1.1 How WRED Works?

Weighted random early detection (WRED) is a queue management algorithm with congestion avoidance capabilities, where different queue (class) has different queue thresholds. Each queue threshold is associated to a particular IP precedence or DSCP etc... For example: A queue may have lower thresholds for lower priority packet and higher thresholds for higher priority traffic. A queue buildup will cause the lower priority packets to be dropped, hence protecting the higher priority packets in the higher priority queue.

1.2 WRED Algorithm

As shown in Figure 1, every time a new packet arrives, average queue size is calculated. Average queue size is calculated based on the previous average and current size of the queue. If the average queue size is less than the minimum queue threshold, the arriving packet is compulsory queued (passed). If the average queue size is between the minimum queue threshold and the maximum threshold, the packet is either dropped or queued (passed), depending on the packet drop probability. If the average queue size is greater than the maximum threshold, the packet is compulsory dropped.

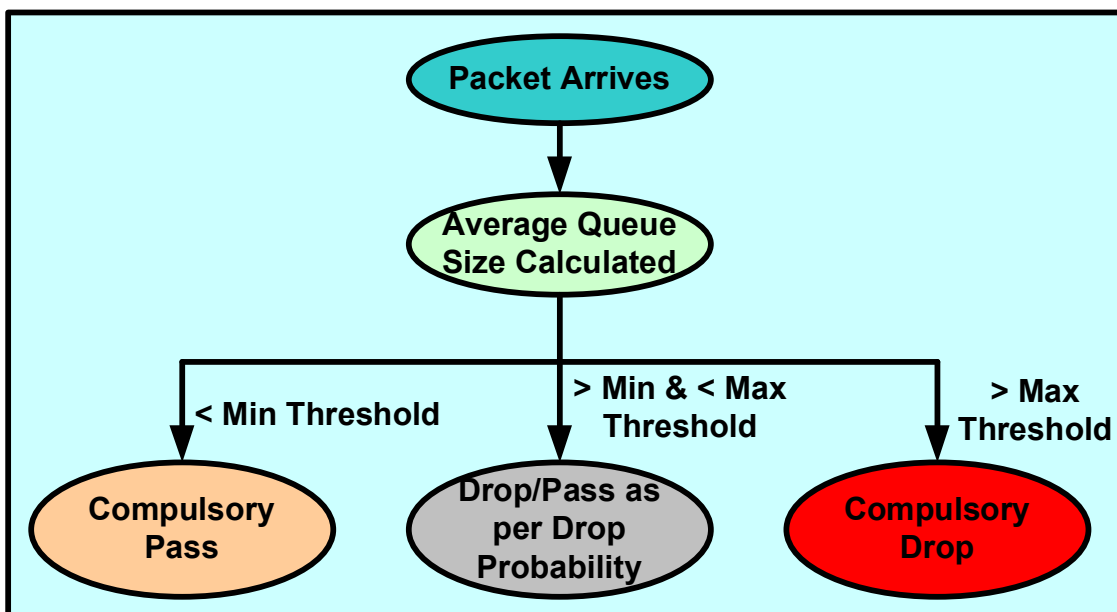


Figure 1 – WRED Algorithm Flow Diagram

The packet drop probability is based on the minimum threshold, maximum threshold, and mark probability denominator. When the average queue depth is above the minimum threshold, WRED starts dropping packets. The rate of packet drop increases linearly as the average queue size increases until the average queue size reaches the maximum threshold. When the average queue size is above the maximum threshold, all packets are dropped.

Figure 2 depicts the graphical representation of packet drop probability vs. average queue size for two types of traffic (low priority traffic and high priority traffic). The low priority traffic has lower value for minimum and maximum thresholds compared to higher priority traffic. Hence at times of congested traffic, the drop rate for lower priority traffic will be higher compared to higher priority traffic. WRED parameters including minimum threshold, maximum threshold and drop profile can be adjusted as per the requirement of different types of traffic.

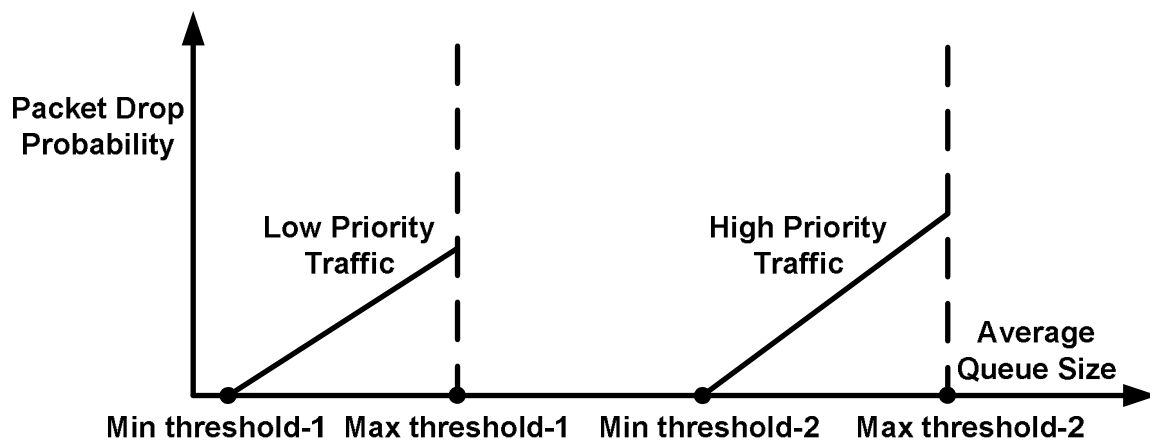


Figure 2 – WRED Packet Drop Probability

1.3 WRED Example

Table 1 shows one of the possible WRED drop profile. As the average queue size increases from 10 to 230 and reaches towards the maximum threshold, drop probability also increases linearly. When the average queue size is less than 50 (minimum threshold), drop probability is 0% and when it exceeds 50 (minimum threshold), drop probability increases linearly till it reaches to 100% on average queue size exceeding 200 (maximum threshold). Figure 3 shows the graphical representation of example shown in Table 1.

Average Queue Size	Minimum Threshold	Maximum Threshold	Drop Probability
10	50	200	0%
44	50	200	0%
48	50	200	0%
60	50	200	6%
64	50	200	6%

72	50	200	12%
76	50	200	12%
116	50	200	43%
140	50	200	56%
160	50	200	68%
188	50	200	87%
196	50	200	93%
210	50	200	100%
230	50	200	100%

Table 1 – WRED Increment Drop Profile

Note: All the figures shown in this table are taken for example and doesn't resemble to any standard formula.

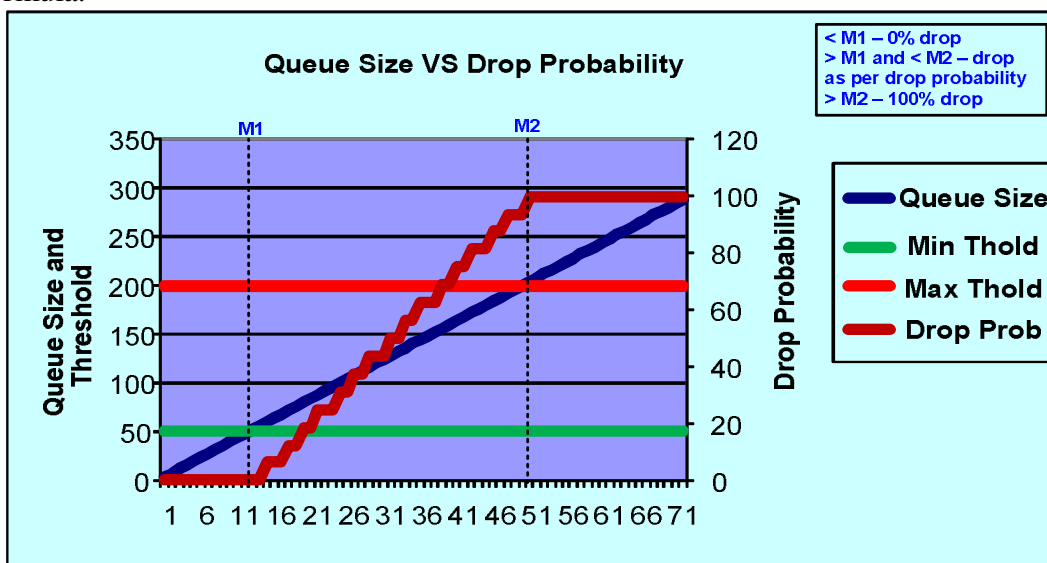


Figure 3 – WRED Increment Drop Profile

2 WRED Verification

WRED works on algorithm which takes decision to drop or pass packets at times of congested traffic depending on the average queue size and configured drop profile. The challenge in algorithm based verification is the quite obvious mind set which diverts on verification of the algorithm implementation rather than concentrating on verifying the application of product for which algorithm is used for. The same challenge applies to WRED verification. WRED verification should be focused on verifying WRED application rather than just the algorithm on which WRED works. The challenge in verifying WRED algorithm is to define verification strategy that doesn't diverts to actual design implementation of WRED but focuses on WRED application.

Listed below are few of the requirements which one may emphasize on when verifying WRED Algorithm.

1. Is the system able to send normal traffic without any drop? I.e. are all the classes (either lower priority or higher priority) getting zero drop rates when the traffic is normal?
2. Is the system able to sustain congested traffic?
3. In case of congested traffic, are all the higher priority classes getting expected lower drop rate than the lower priority traffic?
4. If the packet drops for any of the class is expected, are the packets for each class getting dropped as per the WRED algorithm (i.e. as per average queue size and configured drop profile)?
5. Is the system able to come back on line (i.e. normal traffic transmission without any drop) after the congested traffic has slowed down?

2.1 WRED Verification Approach

WRED verification approach has to be focused on verifying WRED algorithm and not WRED implementation. Unlike other verification architecture which consists of reference models duplicating the design implementation to predict the results, it will be very tricky to use such reference model for WRED verification. The risk which one might run when using reference model for predicting WRED results is getting too close to the WRED design implementation. As shown in Figure 1, WRED algorithm is completely based on configured drop probability and relation between thresholds and average queue size. Hence instead of defining WRED reference model which duplicates the exact implementation of WRED design, one should predict WRED results depending on configured drop probability and thresholds. The only issue with this approach is the difficulty in predicting the exact packet which is going to be lost. The sections below will describe how can we handle this issue and without predicting status for each packet how can we validate WRED results.

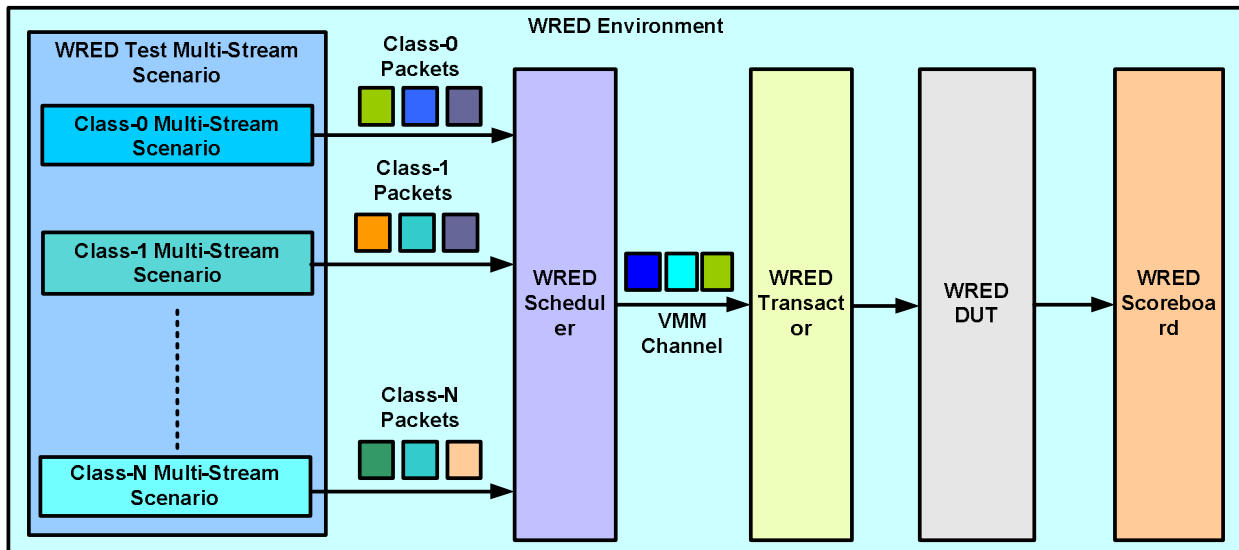


Figure 4 – WRED Verification Environment

Figure 4 shows possible WRED verification environment consisting of WRED Single-Stream and Multi-Stream Scenarios, WRED Scheduler and WRED Scoreboard clubbed inside WRED Verification Environment. WRED Scenarios (extended from VMM Scenarios) and WRED

Scheduler (extended from VMM Scheduler) will be used to generate and send stimulus and WRED Scoreboard (extended from VMM Scoreboard) will be used to predict and check WRED results.

2.1.1 Stimulus Generation Requirements

WRED stimulus generation should be focused on the verification of different types of traffic for different class targeting different thresholds and drop profiles. Stimulus generation is the most crucial part of WRED verification. Instead of fully random stimulus, here the focus is on constrained random stimulus required to generate traffic as per the required drop profiles. The selection of drop profiles, class and WRED parameters can still be random but traffic has to be constrained as per the selected configurations. The focus here is not to verify random packet/data but it is to verify different traffic rates for different classes with different drop profiles as follows:

1. Normal Traffic Generation: To generate normal traffic, packets need to be generated such that average queue size for that corresponding class never exceeds minimum threshold. Thus we need random generation for selecting class and corresponding WRED parameters (minimum threshold, maximum threshold and drop profile) whereas need constrained or directed stimulus for packet generation.
2. Congested Traffic Generation: To generate congested traffic, focus is required on generating packets/traffic such that the average queue size for corresponding class falls either inside minimum threshold and maximum threshold range or is above maximum threshold. Also, to verify different drop profiles, it is required to target different regions between minimum threshold and maximum threshold. This said, we need random generation for selecting class and corresponding WRED parameters (minimum threshold, maximum threshold and drop profile) whereas need constrained or directed stimulus for packet generation, targeting different regions between minimum and maximum threshold and corresponding drop profiles.

To summarize stimulus generator requirements:

1. Flexible: To generate traffic as per the required class, region and random WRED parameters (minimum threshold and maximum threshold).
2. Reusable: To be reused in all the tests without any modifications (except changing the region, class and WRED parameters as per the test requirement).

To make it easy for understanding, dividing the range between minimum threshold and maximum threshold in 4 different regions (Region 1 to 4) resulting into total 6 regions (one below minimum threshold, one above maximum threshold and 4 between minimum and maximum thresholds) as shown in the Figure 5.

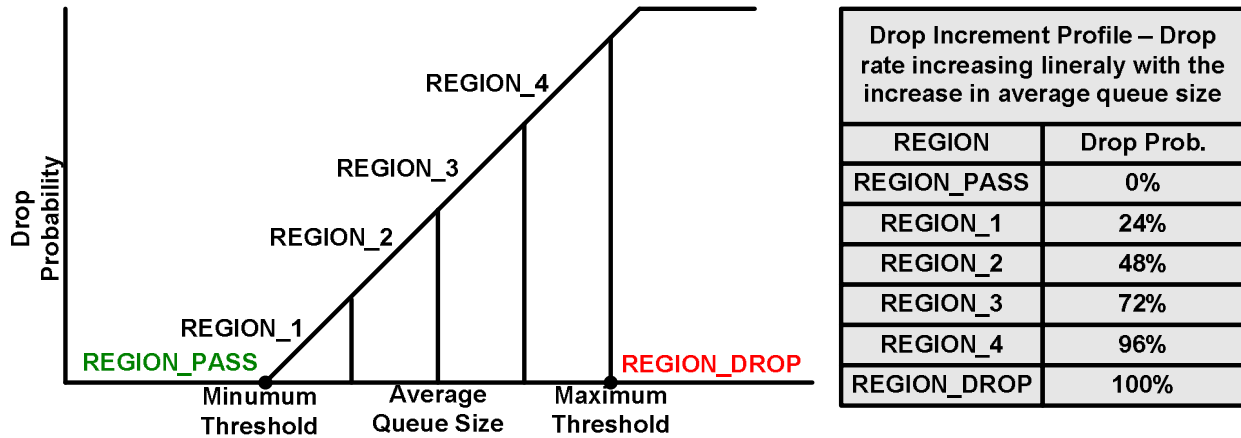


Figure 5 – WRED Pass and Drop Region

Targeted Region	Stimulus Requirements	Description
REGION_PASS	average_queue_size < minimum threshold	This region targets normal traffic.
REGION_1	minimum threshold < average_queue_size < 1/4th maximum threshold	This region targets congested traffic falling in region1. The packets falling in this region will be dropped as per the configured drop probability (For example, 24% drop probability)
REGION_2	minimum threshold < average_queue_size < 2/4th maximum threshold	This region targets congested traffic falling in region2. The packets falling in this region will be dropped as per the configured drop probability (For example, 48% drop probability)
.....
REGION_DROP	average_queue_size > maximum threshold	This region targets congested traffic falling in region_drop which implies all the packets for this class will be dropped compulsory and not as per configured drop probability

Table 2 – WRED Pass and Drop Region

2.1.2 Map WRED Stimulus Requirements with VMM Scenarios

WRED Stimulus generator implementation requirements can be addressed using VMM Scenarios and VMM Scheduler as shown in Figure 6.

VMM Scenarios: VMM scenarios are short sequences of transactions that are directed or mutually constrained, or a combination of both. User can generate sequences of random or directed stimulus using VMM scenarios as per the requirement. VMM Single-stream Scenario can be used to generate sequence of directed or constrained transactions required to be output on single output channel. By default the single-stream scenario generator has the ability to generate random scenarios. Directed or constrained stimulus can be created by extending the basic single scenario class. VMM Multi-stream Scenario provides the same ability as single-stream scenarios but in addition to that it adds the capability to drive, control and access more than one channel. Multi Stream Scenarios can be used to generate sequence of directed or constrained transactions required to be output on multiple output channel. Multi-stream scenarios can also call other multi-stream scenarios.

VMM Scheduler: VMM Scheduler funnels transactions from multiple sources into one output channel. Unlike a simple channel that indiscriminately interleaves transaction when fed from multiple sources, VMM Scheduler can identify source streams and use a user-configurable scheduling algorithm to place the transactions in the output channel.

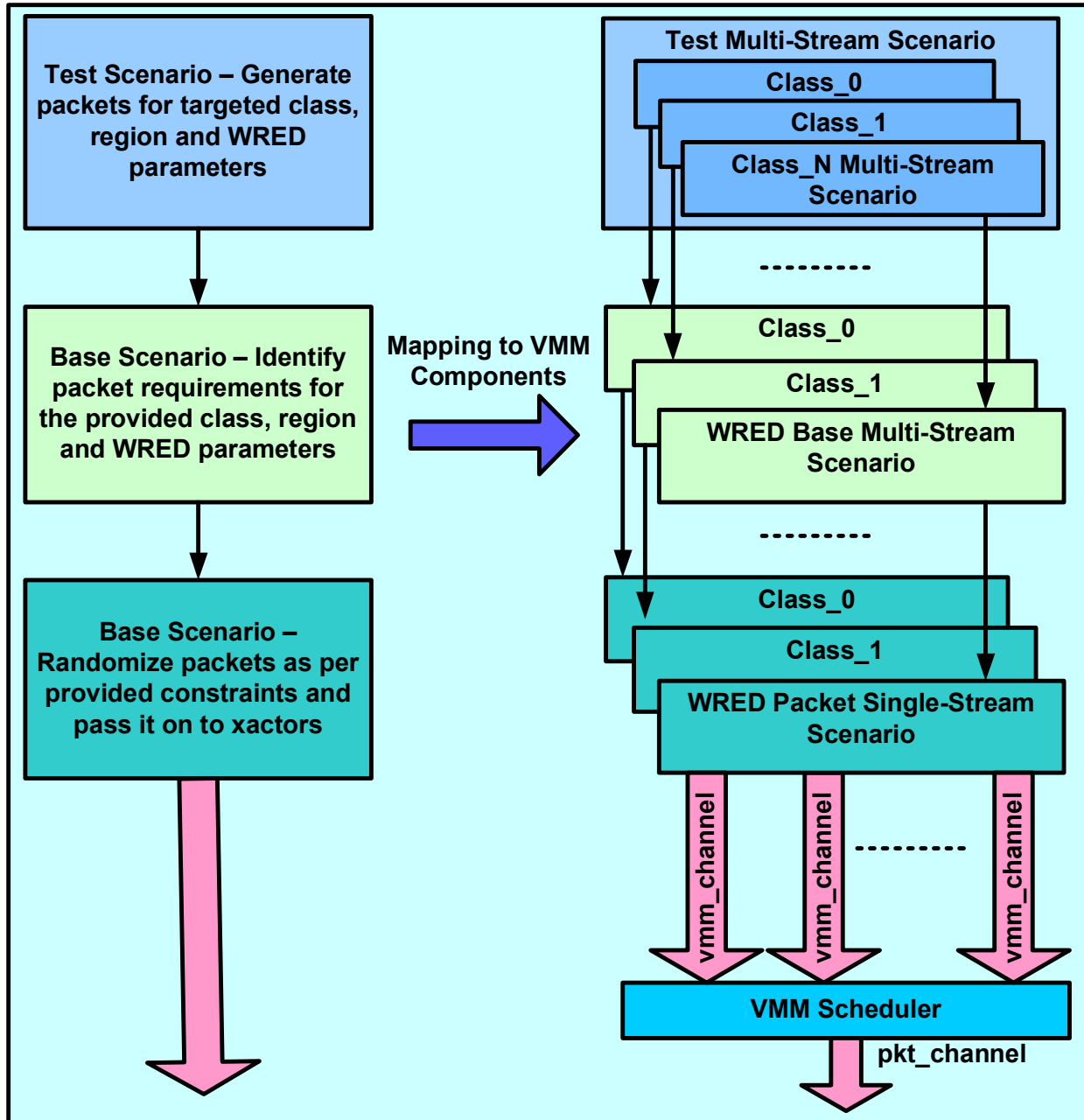


Figure 6 – Mapping WRED Stimulus requirements with VMM Components

Figure 6 describes how WRED stimulus requirements can be directly mapped to VMM scenarios and scheduler. The three step stimulus generation is as follows:

1. **Requirement:** The top level test requirement is to generate packets as per the required class, region information and WRED parameters. In addition to this, there will always be requirement to drive on other interfaces too, for example initialization of WRED design.
Solution: As the requirement is to access, control and drive more than one interface, VMM multi-stream scenario best fits here. In addition to that, multi-stream scenario execute () method provides great flexibility in terms of using various other multi-stream scenarios as well as single stream scenarios as per the requirement.

2. **Requirement:** As per the selected information, it is required to generate the packet constraints i.e. if region_1 is selected; queue size and packet length needs to be generated accordingly.
Solution: Again here the requirement is to control multiple output channels, for example one to set required queue size, other to put generated packets etc. Hence, multi-stream scenario is the best fit here too.
3. **Requirement:** Once the packet constraint information is generated, the requirement is to finally generate packets which will then be put to channel to be preceded to further transactors to be driven to WRED design.
Solution: Here the requirement is to generate the packets using the required information, either directed or constrained. Hence, single-stream scenario best fits here.

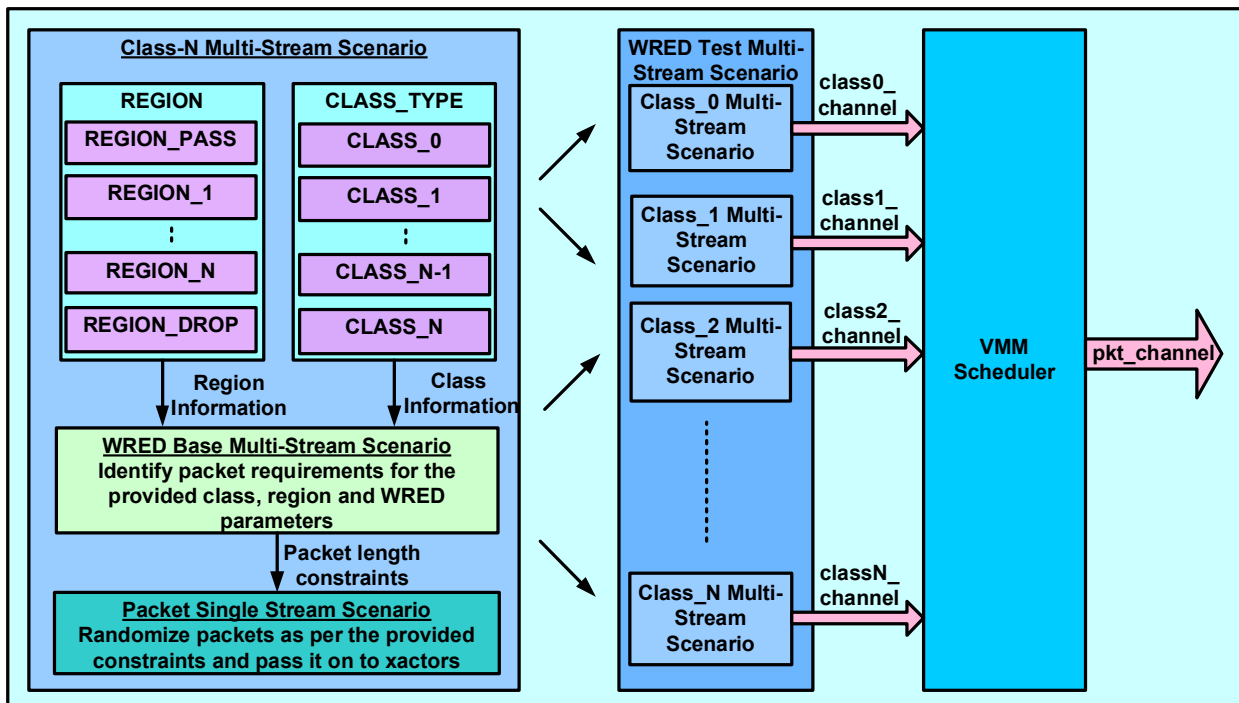


Figure 7 – WRED Stimulus Scenarios

Figure 7 summarizes WRED stimulus scenarios. The below mentioned sections describe WRED scenarios and WRED scheduler in brief.

WRED Packet Single Stream Scenario

As described in the above section, single stream scenario is required to generate single or multiple WRED packets using the provided constraints. As shown in Figure 8, min_packet_scenario extends base single stream scenario and constraints all the elements of the packet as per the requirement.

```

class min_packet_scenario extends packet_base_scenario ;
  int SEQ = define_scenario("min_packet_scenario",1) ;
  constraint ipg_cn
  {
    items[0].ipg == 0;
  }
  constraint frame_length_cn
  {
    items[0].frame_length == MIN_FLEN;
  }
  function new();
    super.new();
  endfunction
  function void pre_randomize() ;
    super.pre_randomize();
    items[0].frame_length_cn.constraint_mode(0);
    items[0].ipg_cn.constraint_mode(0);
  endfunction
endclass : min_packet_scenario

```

Figure 8 – WRED Packet Single Stream Scenario

WRED Base Multi-Stream Scenario

As described in the above section, WRED Base Multi-Stream Scenario is required to identify packet constraints as per test requirement. This is the most complex scenario as it consists of core functionality required for the stimulus generation. Depending on the provided information on region and WRED parameters from the test multi-stream scenario, this scenario identifies constraints for packet length, average queue size etc. which will then be provided to single-stream scenario for final generation of packet. As shown in Figure 9, depending on the region, packet constraints are calculated. In addition to this, this scenario will consist of all the functionality required to generated WRED packets as well as all the other supporting packets which needs to be transmitted on other interfaces, if required.

<pre> class wred_base_ms_scenario extends vmm_ms_scenario ; rand target_region_e target_region ; rand bit [CLASS_WIDTH - 1:0] class ; rand bit [QUEUE_SIZE_WIDTH - 1:0] queue_size ; min_packet_scenario pkt_trans_inst ; constraint class_c { class_r inside valid_class; } constraint queue_size_c { if (target_region == REGION_PASS) { queue_size < min_thld; } else if (target_region == REGION_DROP) { queue_size > max_thld; } else if (target_region == REGION_1) { queue_size > min_thld; queue_size < slot_1_max_size; } } </pre>	<pre> } task execute(ref int n); super.execute(n); `vmm_note(log,"Starting"); for (int i=0;i<num_pkt_cnt;i++) begin pkt_trans_inst.randomize with { length == 1; items[0].class_r == class_r; }; pkt_trans_inst.apply(classn_packet_channel, nn); n += nn; end `vmm_note(log,"Completed"); endtask endclass : wred_base_ms_scenario </pre>
--	---

Figure 9 – WRED Base Multi-Stream Scenario

WRED Test Multi-Stream Scenario

As mentioned in the above sections, WRED test scenario will use multiple instances of WRED base multi-stream scenario, one for each separate class. As shown in the Figure 10, if the requirement is to send interleaved traffic for two classes and want to target PASS and DROP region for class0 and Region 1 and Region 2 for class1 than the only code which is required in test scenario is take two instance of wred base scenario, one for class0 and other for class1 and provide the region and class information. That's it, isn't it simple? Thus test scenario completes with very few lines of code.

<pre> class wred_test_scenario extends vmm_ms_scenario; wred_base_ms_scenario wred_class0_scenario_inst ; wred_base_ms_scenario wred_class1_scenario_inst ; task execute(ref int n); super.execute(n); `vmm_note(log,"Started"); wred_class0_scenario_inst.randomize with {class_r == CLASS_0 wred_trgt_slot_r dist {REGION_DROP := 50, REGION_PASS := 50};}; wred_class1_scenario_inst.randomize with {class_r == CLASS_1 wred_trgt_slot_r dist {REGION_1 := 50, REGION_2 := 50};}; </pre>	<pre> fork wred_class0_scenario_inst.execute(n); wred_class0_scenario_inst.execute(n); join `vmm_note(log,"Completed"); endtask endclass : wred_test_scenario </pre>
---	--

Figure 10 – WRED Test Multi-Stream Scenario

VMM Scheduler

As shown in Figure 7, if the test requirement is to drive stimulus for multiple class, multiple instance of wred base multi-streams scenario will be taken, one for each class. As each class scenario will put the packet into different output channels, there will be multiple sources of WRED packets while destination is single. Also, there is no specific sequence in which the packets from multiple sources are supposed to be transmitted. As the test requirement is to interleave the packets of different classes, VMM scheduler best fits here. VMM Scheduler will randomly pick packets from multiple sources and will provide the same to output channel (wred_packet_channel) from which transactor will get the packet and then will drive it to WRED design.

2.2 Predict and Verify WRED Results

Stimulus generation is the critical and crucial part of verification and so is the checking of design results corresponding to generated stimulus. As described in Section 1.1, WRED design is expected to drop packets in times of congested traffic as per the configured drop probability. Hence predicting the exact packet that will be lost becomes almost difficult without duplicating the exact WRED design implementation. An alternative is to accept that “some” packets will be lost without trying to predict exactly which ones. And these accepted drop packets can then be used to figure out the actual drop and pass rate for validating the WRED results.

2.2.1 Prediction of WRED Results

VMM scoreboard “vmm_sb_ds::expect_with_losses()” method can be used to address WRED scoreboard requirement to accept the lost packets. It looks for an expected packet matching in the supplied observed packet and, when found, assumes that all expected packets in front of the matching packet were lost. In addition to accepting the lost packets, vmm_sb_ds::expect_with_losses() method will also provide the information on lost packets as

well as lost packets count which can further be used to get more information on actual drop rates as described in below sections.

To make WRED result prediction easy, one of the solutions is to predict results as per different regions and as per the corresponding configured drop profile. From the generated stimulus, we have information about how many packets being transmitted for which regions. Hence, prediction can be done based on the total number of packets transmitted per region and configured drop profile:

Considering the same example mentioned in Figure 5,

1. if we have transmitted 100 packets targeting REGION_DROP, all the 100 packets are predicted to drop
2. if we have transmitted 100 packets targeting REGION_PASS, all the 100 packets are predicted to pass
3. if we have transmitted 100 packets targeting REGION_1, approximate prediction for drop will be 20-30% of 100 packets i.e. 20 to 30 packets are expected to drop and rest are expected to pass
4. if we have transmitted 100 packets targeting REGION_2, approximate prediction for drop will be 40-50% of 100 packets i.e. 40 to 50 packets expected to drop and rest are expected to pass

As shown in Figure 11, generated packets are distributed as per region they are falling into and then depending on the configured drop probability, approximate figures for predicted drop count can be estimated for each region.

Region	Number of packets	Configured Drop Probability	Predicted Drop Count	Actual Drop Count
PASS	100	0%	0	x
1	100	24%	~ 20 to 30	x
2	100	48%	~ 45 to 55	x
3	100	72%	~ 70 to 80	x
4	100	96%	~90 to 100	x
DROP	100	100%	100	x

Figure 11 – WRED Predicted Result

Note: Drop count numbers shown in Figure 11 are taken for example and in actual it totally depends on the configured drop profile.

2.2.2 Verify WRED Results

To verify WRED result, the predicted drop rate for each class is required to compare with actual drop rate. As the predicted drop rate is available in terms of each region, it will require having information on actual drop rate in the same form as shown in Figure 12.

Region	Number of packets	Configured Drop Probability	Predicted Drop Count	Actual Drop Count
PASS	100	0%	0	10
1	100	24%	~ 20 to 30	50
2	100	48%	~ 45 to 55	54
3	100	72%	~ 70 to 80	75
4	100	96%	~90 to 100	90
DROP	100	100%	100	80

Figure 12 – WRED Predicted and Actual Result

vmm_sb_ds::expect_with_losses() method returns the information on lost packet count and lost packets which can be used to get information on actual drop count. Figure 13 describes WRED scoreboard which consists of all the functionality required to pick up packets from design as well verification model and then the prediction and comparison of actual and predicted results. As shown in Figure-6 below, WRED scoreboard gets all the packets from WRED VE while get only the passed packets from WRED design. Out of total 9 packets, WRED DUT drops three packets. Hence VMM scoreboard expected queue will consist of 9 packets while actual queue will consist of only 6 packets. As soon as actual queue receives 1st packet, it looks into expected queue for 1st packet and if it exists, it compares both the packets. In this example, as actual 1st packet matches with expected 1st packet, it moves forward for the comparison of actual 2nd packet. Now as 2nd packet is dropped by WRED design, when it compares the actual packet with expected packet, it mismatches. vmm_sb_ds::expect_with_losses() scoreboard will continue to compare the actual packet with the packets available in the expected queue until it finds the match. Hence it will continue to compare the actual 2nd packet with the expected 3rd packet and which when matches, it returns lost count as 1 and also provide the expected 2nd packet as the lost packet. This way, vmm_sb_ds::expect_with_losses() scoreboard will compare the actual and expected packets and as and when any packet is lost it will provide information on the same. As shown in the Figure-6, at the end lost bucket will consists of 3 packets (coloured as red) and pass bucket will consist of rest 6 packets. This lost bucket will then be used to get the information on actual drop count for each region and each class.

Once the predicted and actual drop counts are available, both the counts will be compared to know WRED results (actual drop and pass rate) for each class. As shown in Figure-7, as the actual drop count for pass region, region 1 and drop region doesn't matches with the predicted count, scoreboard will shout for the mismatch.

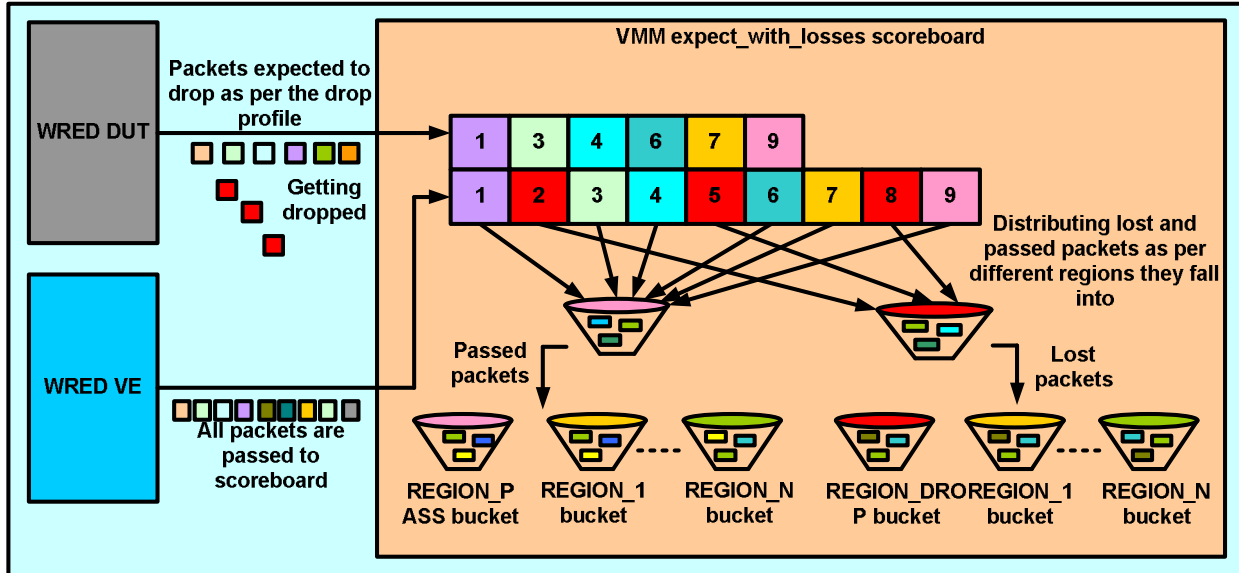


Figure 13 – WRED Scoreboard

2.3 Does Results Make Sense?

It is very difficult to predict the exact packet which is going to drop or pass by WRED design. It totally depends on the WRED design implementation. But irrespective of any WRED design implementation, it has to provide the expected results in terms of expected drop rates. As described in above sections, the prediction and validation of WRED results is dependent on the mathematical calculations derived from the WRED algorithm and configured WRED parameters. Hence it becomes quite tricky at times to gain confidence on these WRED results.

WRED results can be better viewed in form of plots as shown in Figure 3. With the help of Microsoft XLS and using the actual data (drop rates) collected from WRED design, one can plot these graphs. From the graphical representation one can get good confidence on WRED results and should be able to easily identify if something is wrong.

As shown in Figure 14, packet count, average queue size, minimum threshold, maximum threshold and actual drop probability information obtained from WRED results is entered in the XLS and this data is then used to plot queue size vs. drop probability graph. It is clearly visible from this graph that when queue size is less than minimum threshold, drop probability is 0% and as queue size increases linearly, drop probability also increases linearly till queue size exceeds maximum threshold.

These graphs can be plotted for WRED results targeting different traffic rates for each class and for different drop profiles to gain confidence on WRED results.

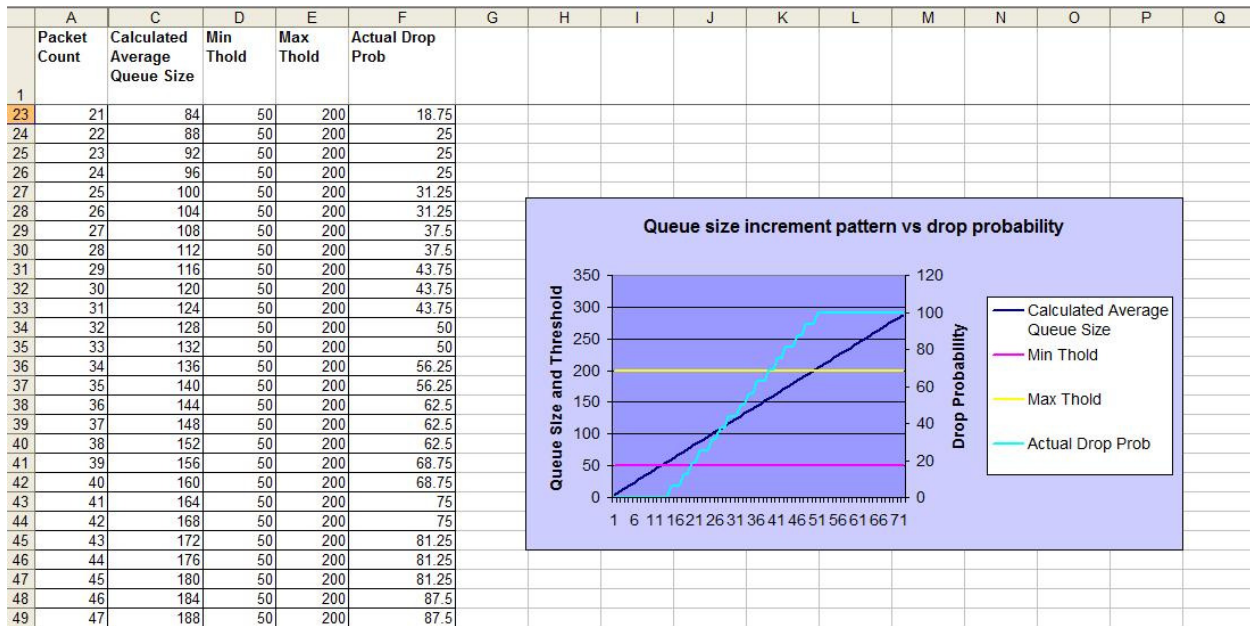


Figure 14 – WRED Result Graph – Drop Rate increasing linearly with queue size

3 Conclusions

Functional Verification is the important phase within the hardware’s design cycle. It has become one of the major tasks in committing chips to fabrication. As the technology grows, the design complexity also grows which in turn increases the complexity of verification. With so many verification techniques available today to reduce the complexity of verification, there still comes challenge to choose right technique for the right design. These challenges can not be addressed straight forwardly as each design comes with its own new challenge and the answer is often confusing and costly. It is very important to realize and understand the verification requirements before making decisions on which technique to use.

The challenge when verifying WRED algorithm is to decide right verification strategy which focuses on verification of WRED application and not WRED Algorithm implementation. Unlike other verification strategies where focus is on random based approach, here focus should be on directed or constrained approach. The main points which should be focussed on while defining WRED verification strategy includes:

1. Focus on WRED algorithm rather than WRED implementation
2. Stimulus is the most critical and crucial part of WRED verification. Stimulus generator should be architected to target real network traffic.
3. Focus should be on directed or constrained based verification environment rather than fully random based verification environment
4. WRED result prediction mechanism should be not be too close to the WRED design implementation
5. Using graphical representation to view WRED results

Stimulus generation is the most complex and crucial part of WRED Verification. The usage of VMM Multi-Stream Scenario and VMM Single-Stream Scenario can make the complex WRED

stimulus generator very simple and reusable. WRED Stimulus generator requirement is to coordinate with multiple interfaces and generate traffic as per the selected traffic rate, WRED parameters and drop profile. It will be very complex to create the required WRED stimulus generator using the basic SystemVerilog code. VMM Single Stream Scenarios can be used to generate different types of packets as per the provided constraints and can be reused in VMM Multi-Stream Scenarios to create more complex and interesting sequence of transactions. VMM Multi-Stream Scenarios targets generation and coordination of stimulus across multiple interfaces and allows hierarchical layering of scenarios which makes it reusable across the verification environments. They can be composed of individual transactions, procedural code, existing single-stream scenarios as well as other multi-stream scenarios. In a nutshell, simply providing the required logic for generating different types of traffic in the `vmm_ms_scenario::execute()` task will suffice the purpose. These Multi-Stream scenarios can then be used as a base Multi-Stream scenario in the test Multi-Stream Scenario and additional test requirements can be provided to base Multi-Stream scenarios in its `execute()` method. With these simple steps, interesting Multi-Stream scenarios can be implemented as per different test requirements.

To do stress testing of WRED design, it is required to interleave traffic for different classes which results in multiple sources of stimulus while single destination. VMM Scheduler can be used to address this requirement. We can also configure the scheduling algorithm as per the WRED requirements.

WRED Algorithm drops packet as per the configured drop probability and hence it is almost impossible to predict the exact packet which is going to drop without duplicating the exact WRED design implementation. The alternative is to accept the lost packets without trying to predict which one. These requirement can be addressed using VMM Scoreboard `vmm_sb_ds::expect_with_losses()` method.

VMM components best fits to most of WRED verification requirements. VMM Single-Stream Scenarios, VMM Multi-Stream Scenarios, VMM Scheduler and VMM Scoreboard usage can make the complex WRED verification development very easy and targeted to the requirements.

4 References

- [1] http://www.vmmcentral.org/pdfs/vmm_standard_library_guide.pdf
- [2] http://www.vmmcentral.org/pdfs/vmm_scoreboard_guide.pdf
- [3] http://en.wikipedia.org/wiki/Weighted_random_early_detection
- [4] http://www.cisco.com/en/US/docs/ios/11_2/feature/guide/wred_gs.html
- [5] http://www.cisco.com/en/US/docs/ios/12_1/qos/configuration/guide/qcdconav.html
- [6] SNUG 2009 – Using the New Features in VMM 1.1 for Multi-Stream Scenarios