DEVELOPMENT FRAMEWORK FOR CONGESTION AVOIDANCE MECHANISMS

Vasile DADARLAT, Raluca JELER, Adrian PECULEA, Bogdan IANCU,
Emil CEBUC, Cosmin ARDELEAN
Technical University of Cluj-Napoca, Faculty of Automation and Computer Science, Computer Science Department
26-28 G. Baritiu street, Cluj-Napoca, Romania, Tel: +40264202369,
{Vasile.Dadarlat, Adrian.Peculea, Bogdan.Iancu, Emil.Cebuc, Cosmin.Ardelean}@cs.utcluj.ro

Abstract: The paper is presenting a novel approach for the design and implementation of a development framework for congestion avoidance mechanisms. A method for dynamically adjustment of the queues length function of their average queue size is also proposed. The developed algorithm, called improved WRED, uses this method which allows for a better use of the bandwidth of the link. Both the proposed algorithm and the traditional WRED were tested using the framework for the development of congestion avoidance mechanisms. The experiments showed that the improved WRED has better performance than the traditional.

Keywords: *framework, WRED, QoS, congestion avoidance, algorithm.*

I. INTRODUCTION

QoS (Quality of Service) reserves resources and provides different priority to different applications, users, or data flows in order to ensure a certain level of performance to a data flow. One mechanism used in QoS implementation is congestion avoidance. WRED (Weighted Random Early Detection) is an active queue management mechanism that provides congestion avoidance [1].

A framework can be defined as an abstraction which delivers generic functionality that can be selectively overridden or specialized by user code providing specific functionality. The purpose of a framework is to offer the user capability to extend the main functionality. A software framework is a set of code or libraries which provide functionality common to a whole class of applications. While one library will usually provide one specific piece of functionality, frameworks will offer a broader range, which are all often used by one type of application. Rather than rewriting commonly used logic, a programmer can leverage a framework which provides often used functionality, limiting the time required to build an application and reducing the possibility of introducing new bugs [2].

The current paper is focused on defining a framework for the development of congestion avoidance mechanisms. The framework allows for designing and testing different algorithms for congestion avoidance in a physical test network. Also, an improved form of WRED algorithm which allows for a better use of the bandwidth of the link is proposed. Using the designed framework, the improved WRED algorithm is compared with the traditional WRED algorithm. The test proved that the proposed algorithm has better performances than the traditional one.

The paper is organized as follows: Section II provides background information related to congestion avoidance algorithms and mechanisms. Section III presents the proposed framework's architecture and an improved WRED algorithm. Section IV presents the experimental results by means of comparing the proposed improved WRED algorithm with the traditional WRED algorithm. Section V concludes the paper.

II. THEORETICAL CONSIDERATIONS

Random early detection (RED), also known as random early discard or random early drop is a proactive queue management technique for congestion avoidance in which the router discards packets before the buffer's overflow [3][4].

An active queue management is an algorithm that consists in a dropping strategy before the router's buffer is full. These algorithms contain a level of intelligence that deals with queues when the congestion is detected.

There are three possibilities for packet dropping:

1. Tail drop, which discards the last arrived packet;
2. Front drop, which removes the first packet in the queue;
3. Random drop, which eliminates a randomly selected packet within the queue.

In a traditional tail drop algorithm, a router stores as many packets as it can and removes those that cannot be kept. If buffers are constantly full, the network is congested. As a result of buffer overflow, TCP obtains congestion feedback and grow its window to fill up the router buffer, causing a loss. This tail drop loss is used as the congestion indication. The method has major drawbacks. First, tail drop distributes buffer space unfairly among traffic flows. Second, loss synchronization - if several connections share the same link, when the buffer fills up, many connections incur loss at the same time. All these connections will back off their window at the same time, resulting in an underutilization of the link. Third, when one or several connections monopolize the whole buffer and because

Manuscript received September 4, 2010; revised November 3, 2010

dropping algorithm is combined with the mechanism of "slow start" of TCP, the tail drop will not allow other connections to gain access to resources. Finally, fourth, queues are occupied for longer periods of time and this leads to network delays.

RED was designed with four objectives in mind:

1. minimize packet loss and delay:
2. avoid global synchronization of TCP sources:
3. maintain high link utilization;
4. remove bias against bursty sources.

In addition, RED addresses the traditional tail drop's issues by using the last dropping strategy for eliminating a randomly selected packet within the queue. It detects the initial stage of congestion by computing the average queue size. If the buffer is almost empty, all incoming packets are accepted. As the queue grows, the probability for dropping an incoming packet grows too. When the buffer is full, the average queue size exceeds a threshold, all incoming packets are dropped.

The following three parameters influence when a newly arriving packet is discarded: minimum threshold, maximum threshold and Mark Probability Denominator (MaxP). The minimum threshold specifies the number of packets in a queue before the queue considers discarding packets. The discard probability increases until the queue depth reaches the maximum threshold. After a queue depth exceeds the maximum threshold, all other packets that attempt to enter the queue are discarded.

RED computes the average queue size (avg). When the average queue size is above the minimum threshold, RED 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. The mark probability denominator is the fraction of packets dropped when the average queue size is at the maximum threshold. For example, if the denominator is 512, one out of every 512 packets is dropped when the average queue is at the maximum threshold. When the average queue size is above the maximum threshold, all packets are dropped. Fig. 1 summarizes the packet drop probability.

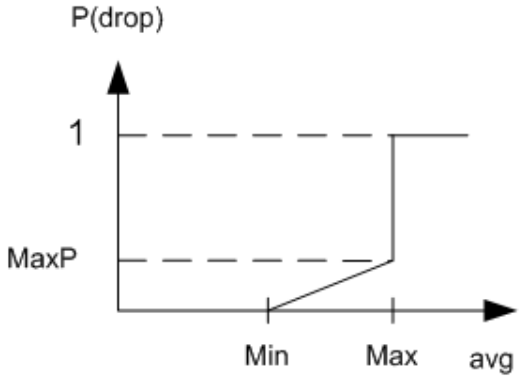

Figure 1. The packet dropping probability in RED

RED algorithm has two distinct algorithms:

1. The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the gateway queue.
2. The algorithm for calculating the packet-marking probability (when average queue size is between Min and Max) determines how frequently the gateway marks packets, given the current level of congestion.

The algorithm for calculating the average queue size take into account the period in which the queue is empty (the idle period) by estimating the number m of small packets that could have been transmitted by the gateway during the idle period. After the idle period the gateway computes the average queue size as if m packets had arrived to an empty queue during that period.

The average queue size is determined based on the following (1):

$$avg = (1 - w_q) \cdot avg + w_q \cdot q \tag{1}$$

where w_q is queue weight and q is the queue size.

If w_q is too large, the previous average becomes more important. The RED process will be slow to start dropping packets and it may continue dropping packets for a time after the actual queue size has fallen below the minimum threshold. Thus RED will not react to congestion and packets will be transmitted or dropped as if RED were not in effect. In this case the averaging procedure will not filter out transient congestion at the gateway.

If w_q is set too low, then avg responds too slowly to changes in the actual queue size. In this case, the gateway is unable to detect the initial stages of congestion [5].

As avg varies from Min to Max, the packet-marking probability p_b varies linearly from 0 to Max_p (2):

$$p_b = \frac{Max_p(avg - Min)}{(Max - Min)} \tag{2}$$

The minimum threshold value should be set high enough to maximize the link utilization. If the minimum threshold is too low, packets may be dropped unnecessarily, and the transmission link will not be fully used.

The difference between the maximum threshold and the minimum threshold should be large enough to avoid global synchronization. If the difference is too small, many packets may be dropped at once, resulting in global synchronization.

Optimal values for Min and Max depend on the desired average queue size. If the traffic is fairly bursty, then Min must be correspondingly large to allow the link utilization to be maintained at an acceptably high level. On the other side, the optimal value for Max depends in part on the maximum average delay then can be allowed by the gateway. The RED gateway functions most effectively when (Max - Min) is larger than the typical increase in the calculated average queue size in one roundtrip time. A useful rule-of-thumb is set Max to at least twice Min.

The final packet-marking probability p_a increases slowly as the count increases since the last marked packet (3):

$$p_a = \frac{p_b}{(1 - count \cdot p_b)} \tag{3}$$

This ensures that the router does not wait too long before marking a packet.

WRED – Weighted Random Early Detection is an extension of RED where the probability that packets will be

dropped is adjusted according to IP precedence levels. Typically, packets are sorted into queues based on fields such as IP precedence, either DIFFSERV code-points or ToS values. Allowing queues to have different parameters is a simple way to implement QoS policy based on classes of traffic. Visually, we can picture WRED as supporting multiple thresholds based on weights, as shown in Figure 2 below.

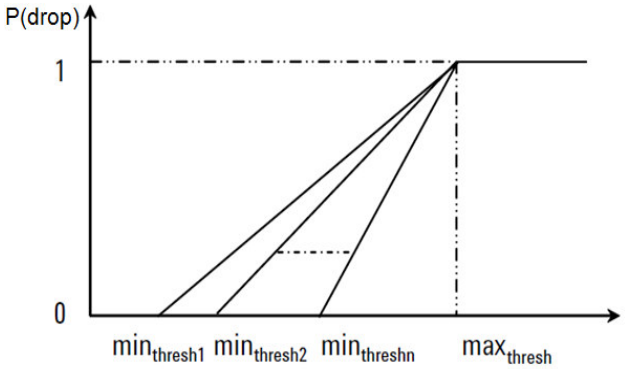

Figure 2. The packet dropping probability in WRED

WRED avoids the globalization problems that occur when tail drop is used as the congestion avoidance mechanism on the router.

WRED drops packets randomly prior to congestion and tells the packet source to decrease the transmission rate. If the packet source is TCP, it will decrease the transmission rate until all packets reach their destination and the congestion is cleared. As a result, WRED is useful only for TCP. On other protocols, the packet source may not respond or may transmit at the same transmission rate. Thus, the packet dropping does not avoid the congestion.

If WRED uses IP precedence as criterion for packet dropping, the packets with a higher IP Precedence are less likely to be dropped than packets with a lower precedence. Thus, the higher the priority of a packet, the higher will be the probability that the packet is delivered.

If WRED is based on type of traffic, the packets from some classes of traffic are more likely to be dropped than packets from other classes.

Even if RED is the most common active queue management algorithm, there are more variation on this topic. Here is a brief description for some other queue management algorithms that have RED as a starting point.

Dynamic Random Early Detection (DRED) introduces a new parameter: warning line. The average queue size is estimated and is dynamically adjusted. DRED scheme responds early enough to the increased number of packets at the gateway. Also, the maximum drop probability of packets show improved performance over the original RED. This scheme demonstrated superiority by avoiding global synchronization and there is great reduction in the fluctuations of the actual queue size. Also, its early response avoids buffer overflow at the gateways when the queue is near full [6].

FRED – FRED or Fair Random Early Detection imposes the same loss rate on all flows, regardless of their bandwidths. FRED also uses per-flow active accounting, and tracks the state of active flows.

SRED – Stabilized RED attempts to estimate the number of connections, and also identify potential misbehaving flows [7].

Several variations of the Random Early Detection QoS tool implemented in Cisco equipments can be used for congestion avoidance configuration [8]. Cisco IOS Software supports only WRED, which is enabled by the random-detect CLI command. The minimum threshold, maximum threshold and Mark Probability Denominator are tunable parameters so that the system engineer can choose the appropriate values in order to improve the network behavior. DSCP-based WRED uses the AF drop-precedence values of a packet's DSCP markings to influence its discarding probability. DSCP-based WRED configuration is enabled by the dscp-based keyword of the random-detect command. RFC 3168 defines a method for the network to inform the sender about the congestion. Explicit congestion notification (ECN) uses the final two bits of the ToS field in the IP header to communicate the congestion. The two bits are ECT and CE. ECT (ECN-Capable Transport) indicates weather the device supports ECN. CE (Congestion Experienced) indicates weather congestion was experienced. ECN marks the packets instead of dropping them, to communicate the existence of congestion. WRED ECN configuration is enabled by the ecn keyword of the random-detect command.

III. DEVELOPMENT FRAMEWORK FOR CONGESTION AVOIDANCE MECHANISMS

The current work is concentrated in designing and implementing a framework for the development of congestion avoidance mechanisms, in a physical test network. The use of the physical test network includes the adopted approach in the experimental methodologies category, methodologies that have proven accuracy close to that of real cases [9]. Also, the current work is focused in researching new techniques that will overcome the main drawbacks of WRED algorithm. The paper proposes a method for dynamically adjustment of the queues length function of their average queue size. The algorithm developed uses this method which allows for a better use of the bandwidth of the link.

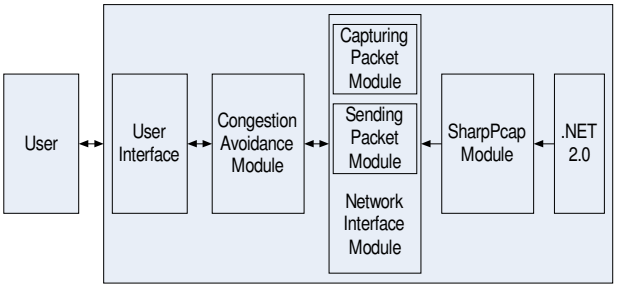

Figure 3. Framework architecture

The framework for the development of congestion avoidance mechanisms (Fig. 3) uses a component based architecture, consisting of the following main modules:

1. User;
2. User interface;
3. SharpPcap module;
4. .NET 2.0 module;
5. Module for capturing packets on the interfaces previous detected;
6. Congestion avoidance module;

7. Sending packets module.

The user interface module interacts with the user through a console window in which all the information about the network devices and specific information for the queue management algorithms are displayed.

The .NET and the SharpPcap module modules are based on the famous WinPcap component. The purpose of these modules is to provide an API for capturing, injecting, analyzing and building packets using any .NET language such as C#.

The modules for capturing and sending the packets are part of the network interface module. The capturing module also has additional functions for the detection and handling of the network devices, and also for packets' control - using a WinpCap wrapper [10] for C# .NET [11]. The sending module uses the FIFO algorithm to send the packets stored in the management queues.

The routing and congestion avoidance module is the core of the application and has the following tasks:

1. Creation and maintaining of the routing tables,
2. Marking traffic packets according to the predefined rules,
3. Placing the packets into their corresponding queues,
4. Determining the output interface,
5. Applying congestion avoidance algorithms,
6. Traffic forwarding.

The framework functionality is focused on the modules described above. Here are the main functions of the system:

1. Detection of the available network devices,
2. Creation and management of the routing tables,
3. Packet capturing and classification,
4. Congestion avoidance using WRED-type algorithms
5. Packet transmission.

At the framework start, all available network interfaces are detected and a list with all available network devices is built. This list contains complete information about an attached adapter: the name and a human readable description of the corresponding device. For each device in the list we retrieve similar information with the 'ipconfig' command available in Windows NT: IP information (IP address, subnet mask and default gateway), the MAC address (physical address) of the adapter, DHCP and WINS information.

Then, after the adapters' list is obtained, the packets that flow through the network are captured, analyzed and, based on the collected information (protocol and port) the packets are classified into their corresponding class of traffic. In our packet handler we first do a check to verify that the packet received from the network device is of a specific type (TCPPacket, UDPPacket, ICMPPacket etc.) and past a specific port, and only then try add it to a specific class (queue).

The congestion avoidance algorithms, based on WRED, are applied on these classes and the packets are either added to the queues or marked to be dropped.

The packets that are stored in the queues are send to the destination device based on the FIFO algorithm.

The framework, through its modular software solution, allows for adapting the developed system to run various congestion avoidance algorithms, just by changing or modification of some components. Also, the system allows for saving and analyzing tests information, such as the number of packets discarded by the congestion avoidance mechanism. Thus, the framework allows in-depth analysis of the congestion avoidance algorithms' behavior.

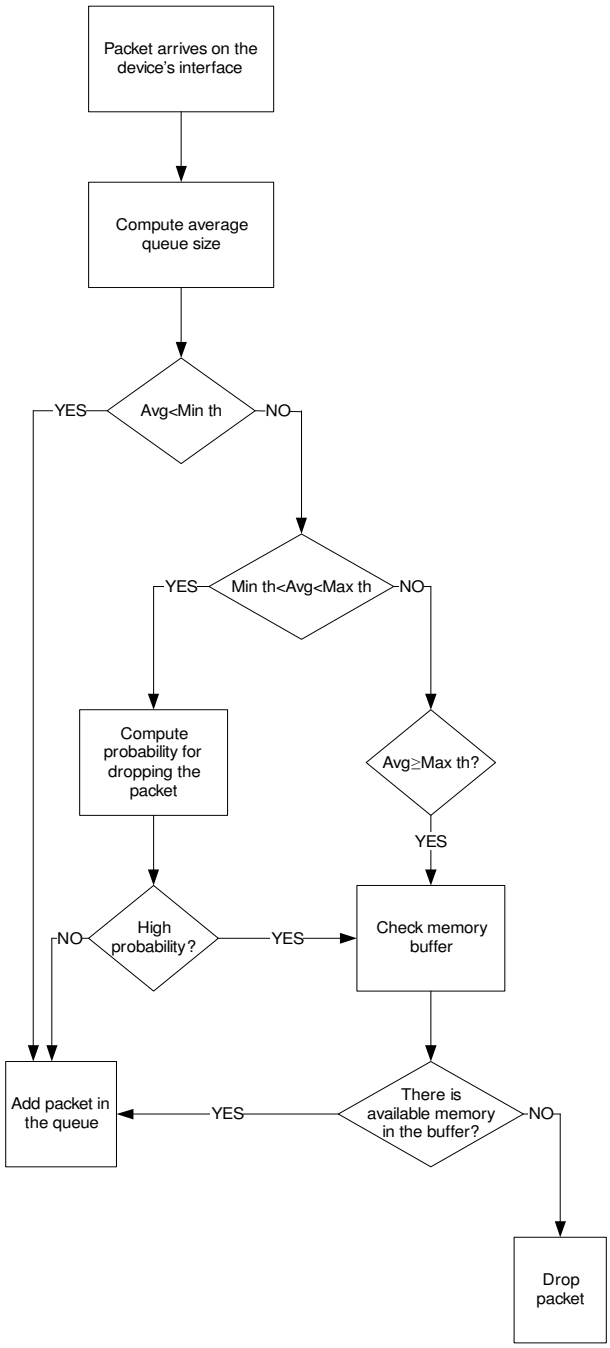

Figure 4. Improved WRED algorithm

As the traditional WRED, the improved algorithm has two major components. First component, the algorithm for computing the average queue size, determines the degree of burstiness that will be allowed in the gateway queue. This part is similar in both algorithms. The second component, the algorithm for calculating the packet-marking probability (when average queue size is between Min and Max), determines how frequently the gateway marks packets, given the current level of congestion.

For each queue, a minimum amount of memory is

guaranteed. The remaining memory, memory buffer, is dynamically allocated to queues function of their average queue size. Also, the allocated memory is dynamically released when the average queue size decreases. Thus, the memory buffer is dynamically used by any queue function of the current traffic profile and level. This ensures a better use of memory and therefore a better use of the bandwidth of the link. The algorithm for calculating the packet-marking probability uses this memory organization and dynamically allocates and releases memory to and from queues function of their current average queue size. Thus, classes without intense traffic will not occupy unjustified amounts of memory and classes with more intense traffic will benefit from additional amounts of memory, allocated from the memory buffer.

Here is how the improved WRED algorithm works on each queue (Figure 4):

1. For each packet arrival, calculate the average queue size based on (1) described above.
2. If the average queue size is between the two thresholds (min and max), calculate probability pa, for dropping the packet. If the packets need to be dropped, check the memory buffer. If the memory buffer is empty, drop the packet, else add the packet to the current queue and decrement the length for the current memory buffer. The values for the min and max thresholds will be incremented.
3. If the average queue size is grater than max threshold, verify the memory buffer. If memory buffer is empty, drop the packet, else add the packet to the current queue. If the packet is added in the current queue, decrement the length for the current memory buffer. The values for the min and max thresholds will be incremented.

At packet sending, check values for the min and max thresholds. When the limit of the min and max thresholds are grater than theirs initial values, the memory buffer will be incremented and the values for the min and max thresholds will be decreased.

IV. EXPERIMENTAL RESULTS

The algorithm was tested on the network presented in Fig. 5. The framework for the development of congestion avoidance mechanisms was installed on the router. The improved WRED algorithm was compared with the traditional one by generating similar traffic patterns for each algorithm and comparing the data transfer performances. The following two parameters were followed: throughput and number of packets dropped. Five different traffic patterns were generated for each algorithm. The results were more than satisfactory and proved that the improved WRED algorithm has better performance.

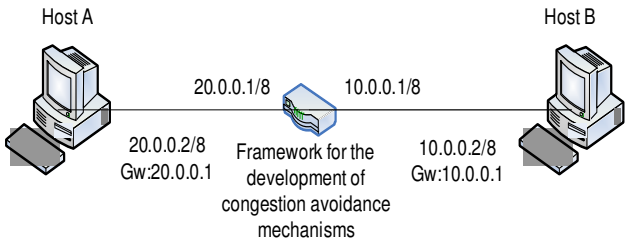

Figure 5. Testing Network

Experiment 1

In the first experiment, the traffic pattern consisted of two traffic classes (Fig. 6). The first class generated 64Kbps ICMP traffic both from Host A to Host B and from Host B to Host A. The second class was represented by a 10 MB FTP transfer, from Host A to Host B.

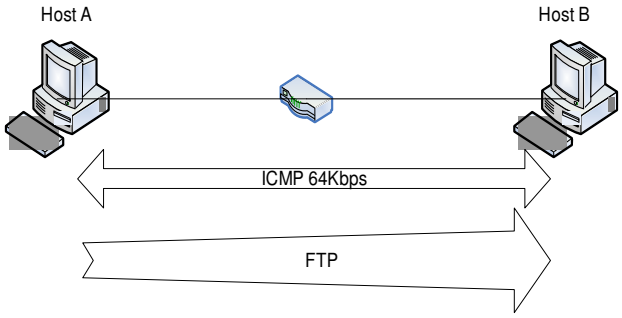

Figure 6. First experiment - traffic pattern with two traffic classes

Using the improved WRED algorithm the 10 MB file was transferred through the FTP protocol in 11.16 seconds with 939.92 Kbytes/sec and there were dropped 4 packets (0 packets were dropped randomly and 4 were dropped because the average queue size exceeded the max threshold). Using the traditional algorithm, the file was transferred in 18.92 seconds with 554.16 Kbytes/sec and there were dropped 13 packets (10 packets were dropped randomly and 3 were dropped because the average queue size exceeded the max threshold). The results are synthesized in Table 1 and Fig. 7 illustrates the FTP throughput difference between the two WRED implementations.

	Throughput (KBps)	Number of packets dropped
Improved WRED	939.92	4
WRED	554.16	13

Table 1. Results for the first experiment

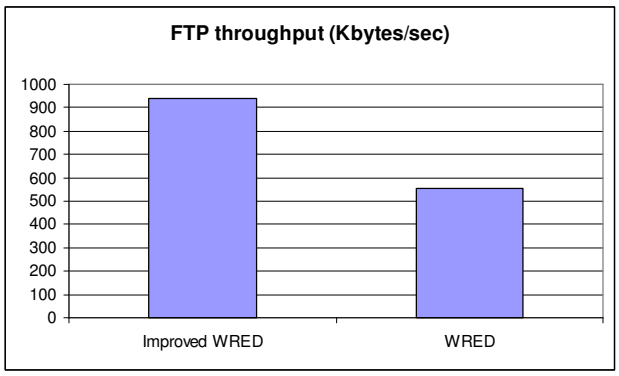

Figure 7. FTP throughput for Experiment 1

Experiment 2

In the second experiment, we have added a third traffic class consisting of 640 Kbps UDP traffic, from Host A to Host B, using a custom benchmarking system for generating traffic test [12] (Fig. 8). The benchmarking allows the possibility to define and store complex traffic patterns that can be

recharged for making further measurements, to test various QoS techniques based on the same traffic characteristics.

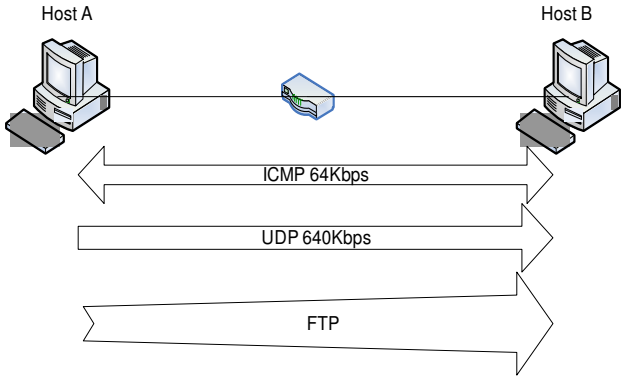

Figure 8. Second experiment - traffic pattern with three traffic classes

Using the improved WRED algorithm the 10 MB file was transferred through the FTP protocol in 11.38 seconds with 921.83 Kbytes/sec and there were dropped 5 packets (3 packets were dropped randomly and 2 were dropped because the average queue size exceeded the max threshold). Using the traditional algorithm, the file was transferred in 18.59 seconds with 563.96 Kbytes/sec and there were dropped 9 packets (5 packets were dropped randomly and 4 were dropped because the average queue size exceeded the max threshold). The results are synthesized in Table 2 and Fig. 9 illustrates the FTP throughput difference between the two WRED implementations.

	Throughput (KBps)	Number of packets dropped
Improved WRED	921.83	5
WRED	563.96	9

Table 2. Results for the second experiment

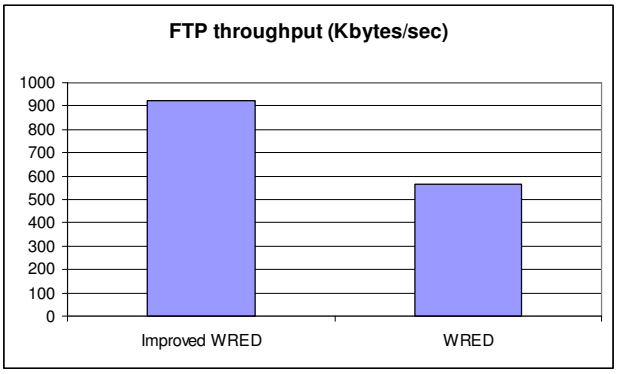

Figure 9. FTP throughput for Experiment 2

Experiment 3
In the third experiment, we have added the fourth traffic class consisting of 640 Kbps UDP traffic, from Host A to Host B (Fig. 10).

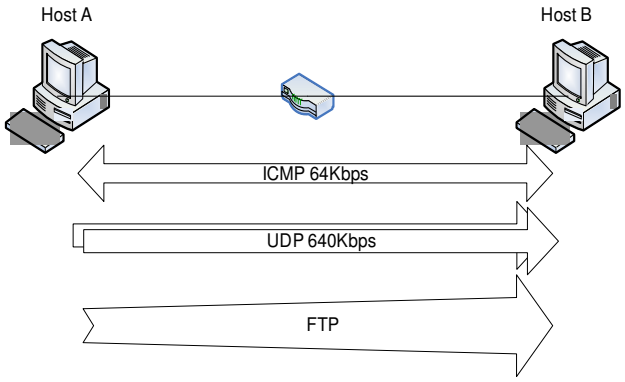

Figure 10. Third experiment - traffic pattern with four traffic classes

Using the improved WRED algorithm the 10 MB file was transferred through the FTP protocol in 12.25 seconds with 855.98 Kbytes/sec and there were dropped 2 packets (0 packets were dropped randomly and 2 were dropped because the average queue size exceeded the max threshold). Using the traditional algorithm, the file was transferred in 21.86 seconds with 479.70 Kbytes/sec and there were dropped 14 packets (12 packets were dropped randomly and 2 were dropped because the average queue size exceeded the max threshold). The results are synthesized in Table 3 and Fig. 11 illustrates the FTP throughput difference between the two WRED implementations.

	Throughput (KBps)	Number of packets dropped
Improved WRED	855.98	2
WRED	479.70	14

Table 3. Results for the third experiment

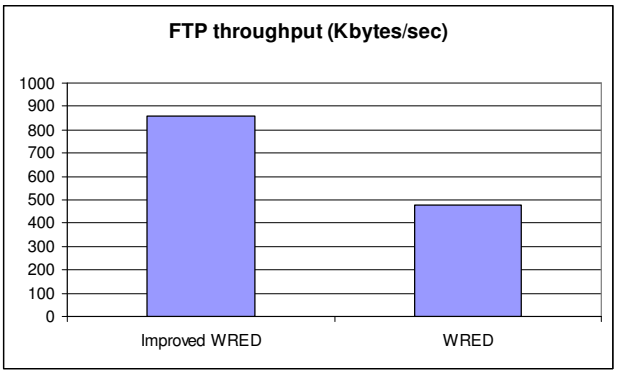

Figure 11. FTP throughput for Experiment 3

Experiment 4
In the fourth experiment, we have added the fifth traffic class consisting of 640 Kbps UDP traffic, from Host A to Host B (Fig. 12).

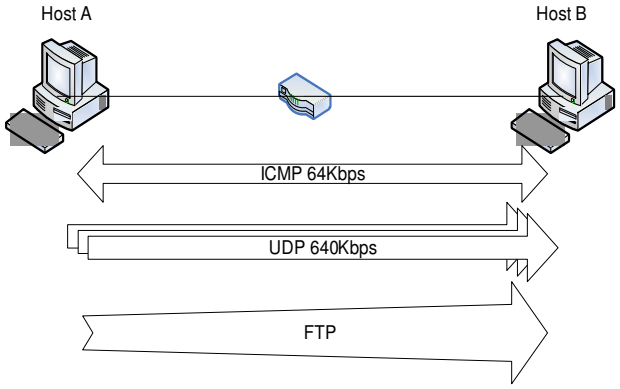

Figure 12. Fourth experiment - traffic pattern with five traffic classes

Using the improved WRED algorithm the 10 MB file was transferred through the FTP protocol in 15.75 seconds with 665.76 Kbytes/sec and there were no dropped packets. Using the traditional algorithm, the file was transferred in 17.39 seconds with 662.98 Kbytes/sec and there were dropped 19 packets (15 packets were dropped randomly and 4 were dropped because the average queue size exceeded the max threshold). The results are synthesized in Table 4 and Fig. 13 illustrates the FTP throughput difference between the two WRED implementations.

	Throughput (KBps)	Number of packets dropped
Improved WRED	665.76	0
WRED	662.98	19

Table 4. Results for the fourth experiment

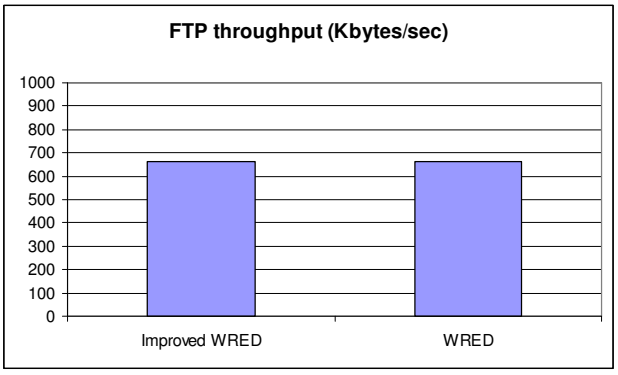

Figure 13. FTP throughput for Experiment 4

Experiment 5

Finally, in the fifth experiment, we have added the sixth traffic class consisting of 640 Kbps UDP traffic, from Host A to Host B (Fig. 14).

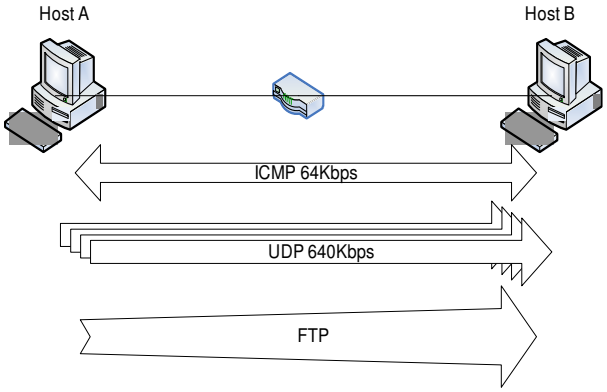

Figure 14. Fifth experiment - traffic pattern with six traffic classes

Using the improved WRED algorithm the 10 MB file was transferred through the FTP protocol in 19.70 seconds with 532.19 Kbytes/sec and there were no dropped packets. Using the traditional algorithm, the file was transferred in 17.55 seconds with 597.58 Kbytes/sec and there were dropped 16 packets (14 packets were dropped randomly and 2 were dropped because the average queue size exceeded the max threshold). The results are synthesized in Table 5 and Fig. 15 illustrates the FTP throughput difference between the two WRED implementations.

	Throughput (KBps)	Number of packets dropped
Improved WRED	532.19	0
WRED	597.58	16

Table 5. Results for the fifth experiment

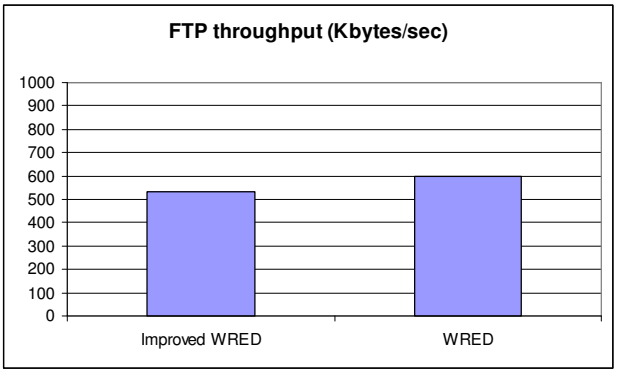

Figure 15. FTP throughput for Experiment 5

In Fig. 16 it can be observed the FTP throughput difference between the two WRED implementations for all the experiments.

Figure 16. FTP throughput for the two WRED implementations

As it was expected, when the network is loaded with a small number of traffic classes the improved WRED has better performance than the traditional one. The improved WRED dynamically allocates and releases memory to and from queues function of their current average queue size. Classes without intense or any traffic will not occupy unjustified amounts of memory and classes with more intense traffic will benefit from additional amounts of memory, allocated from the memory buffer.

When the network is loaded with a large number of traffic classes the two WRED implementations present similar performance.

Using WRED the average FTP throughput obtained is 571.67 Kbytes/sec while using the improved WRED the average FTP throughput obtained is 783.13, which represents an increase of 36.98 percents.

In Fig. 17 it can be observed the number of packets dropped during the experiments. The improved WRED dropped fewer packets than WRED, the number of packets dropped by the improved WRED representing 15.49 percents from the number of packets dropped by WRED.

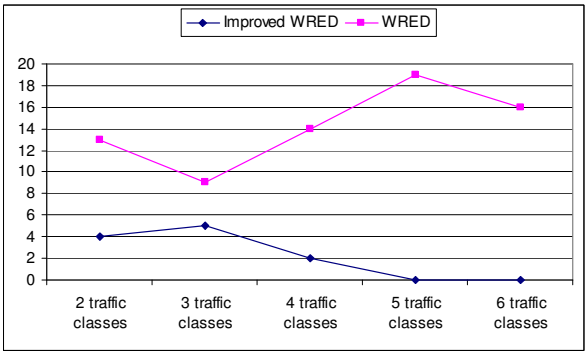

Figure 17. Packets dropped during the experiments

The parameters considered for the algorithms comparison prove that the improved WRED algorithm has better performance than the traditional one.

V. CONCLUSIONS

The framework for the development of congestion avoidance mechanisms allows for the assessment of the performance and functionalities of different congestion avoidance algorithms using an experimental methodology. The main advantages of this system are the accuracy close to that of real cases and the possibility to run various congestion avoidance algorithms and perform in-depth algorithms' behavior analysis.

The memory organization proposed guarantees for each queue a minimum amount of memory and the remaining memory, memory buffer, is dynamically used by any queue function of the current traffic profile and level. The improved algorithm for calculating the packet-marking probability uses this memory organization and dynamically allocates and releases memory to and from queues function of their current average queue size. This ensures a better use of memory and therefore a better use of the bandwidth of the link.

The framework for the development of congestion avoidance mechanisms was used in order to compare the behavior and performance of the improved WRED algorithm with the traditional one. The experiments revealed that the improved WRED algorithm has superior performance in comparison with the traditional WRED algorithm.

ACKNOWLEDGMENTS

This work was supported by the PNII-IDEI 328/2007 QAF - Quality of Service Aware Frameworks for Networks and Middleware research project within the framework National Research, Development and Innovation Programme initiated by The National University Research Council Romania (CNCSIS - UEFISCSU)

REFERENCES

[1] Z. Wang, *Internet QoS: architectures and mechanisms for Quality of Service*, Morgan Kaufmann, San Francisco, 2001.
[2] "Framework", *DocForge*. http://docforge.com/wiki/Framework, Retrieved 17 February 2010.
[3] "RED", http://www.icir.org/floyd/red.html, Retrieved 23 November 2009.
[4] L. Peterson, B. Davie, *Computer Networks, A systems approach 4th Edition*, Morgan Kaufmann Publishers, 2007.
[5] S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397 – 413, 1993.
[6] A. A. Akintola, G. A. Aderounmu, L. A. Akanbi, M. O. Adigun, "Modeling and Performance Analysis of Dynamic Random Early Detection DRED Gateway for Congestion Avoidance", Proceeding of I^{n}SITE (Informing Science + IT Education) Conference, pp. 623 – 636, 2009.
[7] "TCP and Queue Management, White Paper", Agilent Technologies http://cp.literature.agilent.com/litweb/pdf/5989-7873EN.pdf, Retrieved 23 November 2009.
[8] T. Szigeti, C. Hattigh, *End-to-End QoS Network Design*, Cisco Press, 2005.
[9] A. Hughes, W. Emmerich, "Using programmable network management techniques to establish experimental networking testbeds", BT Technology Journal, vol. 21, 2003.
[10] "SharpPcap, Packet capture framework for the .NET environment", http://www.tamirgal.com/blog/page/SharpPcap.aspx, Retrieved 28 November 2009.
[11] J. Sharp, *Microsoft® Visual C#® 2005 Step by Step*, Microsoft, 2005.
[12] B. Iancu, A. Peculea, V. Dadarlat, I. Ignat, E. Cebuc, Z. Baruch, „QoS parameters' benchmarking system with complex traffic pattern definition", Proceeding of RoEduNet 6th International Conference, pp. 44 – 49, 2007.