Chapter Two

Review of Congestion Control Methods

2.1 Introduction

When a network has several sources that require and request network resources to send their data, it is said that the network sources are competing for the resources regardless the numbers of both requests and resources [108]. When the sources demands exceed the available resources, buffers at the network routers begin to fill up, which often causes the network to become congested. This problem results in the degradation of the network performance by 1) increasing both packet loss rate and queueing delay 2) decreasing the throughput performance and 3) unfairly sharing network resources among the network sources.

In order to enhance the network performance by increasing the throughput, maintaining queue size as small as possible, and decreasing both packet loss rate and queueing delay, many researchers have developed congestion control approaches [39, 41, 43, 46, 50, 68, 69, 96]. The ultimate goal of these approaches is to manage and control the congested network

as early as possible [12, 51], which eventually provides a satisfactory QoS for traffic loads. Furthermore, establishing a fair share among network sources is another primary goal for the above approaches [39, 42].

Transport Control Protocol (TCP) is mainly utilised in the internet and it is a reliable method for transferring data between two end points (the source and the destination) [8, 63, 64, 81, 95, 100, 105]. Before the advent of TCP, networks used to rely on a connectionless protocols to transfer data between sources and destinations [108, 119], where the protocol was unable to identify whether data packets had been successfully received by the destinations. In other words, the protocol was unable determine whether a data packet had been corrupted, received, out of order, duplicated or lost because of the congestion in the network.

TCP has several properties [95, 104, 108, 119]; first, it establishes a connection between two ends, in which one of the two ends transmits a connection request to the other, then the other end replies by sending an acceptance or rejection message. If the reply is a rejection, the connection will be cancelled, otherwise, the connection is established, and both ends can exchange data messages. Second, the TCP is a reliable method for identifying the delivery status of data packets sent from the source to the destination because of the exchanged acknowledgements between the source and the destination. For instance, the TCP sources can determine which packets are missed. Third, in TCP each source uses a timer to figure out when data packets are sent or are retransmitted in cases of congestion in the network. Also, each source tracks which data packet should be sent next using the acknowledgements it receives from the destination. Each source sending data packets monitors its congestion window (cwnd) which is related to transmission rate,

hence, when the cwnd reaches a certain level (called either data packet loss index or congestion index), the source decreases it cwnd in order to alleviate the congestion.

The aim of this chapter is to survey TCP and Active Queue Management (AQM) approaches in computer networks that have been utilised to control the congestion problem.

This chapter is organised as follows: The TCP congestion control algorithms are introduced in Section 2.2. The congestion control is discussed in Section 2.3. An AQM literature review is presented in Section 2.4. Finally, a chapter summary is made in Section 2.5.

2.2 TCP Congestion Control Algorithms

The ultimate aim of any TCP congestion control algorithm is controlling the congestion in the network by decreasing the sources transmitting rates. [66, 67] proposed a number of TCP congestion control methods in the internet, including the slow start, congestion avoidance and fast retransmission [67], and the fast recovery [66]. In the next four subsections, four TCP congestion control algorithms are described.

2.2.1 Slow Start Algorithm

The slow start algorithm (SS) was proposed to control congestion in the internet, and this method starts working when one of the following events occurs:

a) When a new TCP connection is established.

b) When a TCP connection regains active after along idle period of time.

c) When a TCP connection regains active after the expiration of a timeout.

When the SS method is active due to one of the above events, the source initialises its cwnd to the size of a single TCP segment. The cwnd corresponds to the amount of data that could be sent from the source to its destination across the network prior the presence of congestion. The reason for setting the cwnd to a single TCP segment is to prevent the router buffers from building up quickly [8, 81, 100]. The SS method begins sending the TCP segment to its destination, then waits until it receives an acknowledgement from the destination. After the source receives the acknowledgment, it increases its cwnd size to two segments, which enables it to send two TCP data segments. Again, and after the source receives an acknowledgement that both TCP segments are successfully received by the destination, it increases its cwnd to four segments, and so forth. It should be noted that the source increases its cwnd size exponentially and this is shown in Figure 2.1. Figure 2.2 indicates how the SS method is implemented.

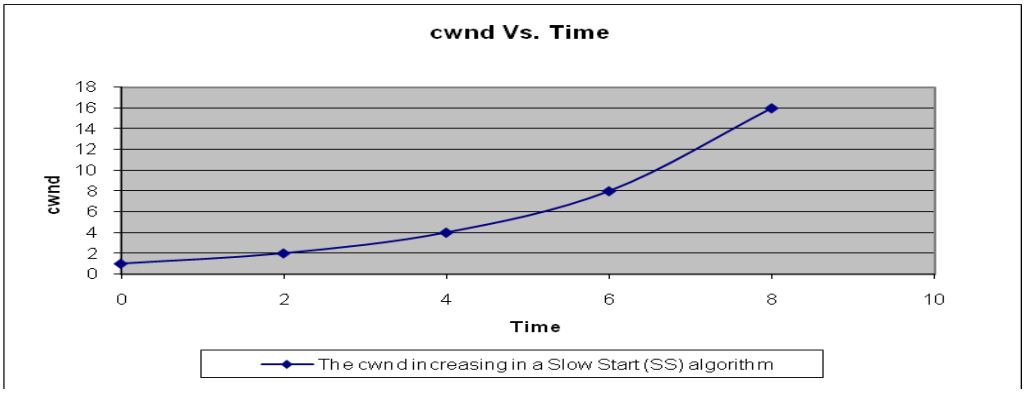

Figure 2.1: The exponential increasing of cwnd in SS method [67, 100].

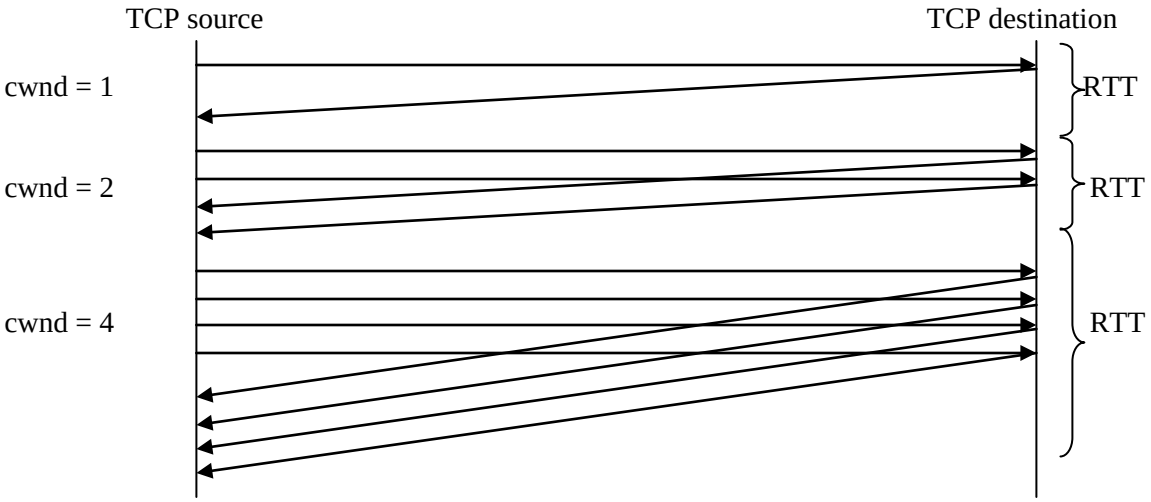

Figure 2.2: The SS method implementation [117].

It is noted in Figure 2.2 that the source increases its cwnd size with the same number of acknowledgements it receives from its destination during a single round trip time (RTT) if each TCP segment was acknowledged separately. The source remains in the SS implementation until either the cwnd size becomes above the destination announced window or congestion occurs in the network due to receiving duplicate acknowledgements by the source. In this case, the source stops implementing the SS method and enters a second stage called congestion avoidance. The next subsection gives the full details concerning this method.

2.2.2 Congestion Avoidance Algorithm

Congestion avoidance (CA) algorithm [67] was proposed in 1988 by Van Jacobson to alleviate congestion in the internet. The CA method gets initialised when either congestion

is present or when the source cwnd size exceeds the destination announced window. The CA method relies on two parameters, the cwnd and the slow start threshold (ssthreshold). When the network becomes congested, the source decreases its cwnd size by setting the ssthreshold parameter to half of the cwnd size value. After the source cwnd size is reduced, the source enters the CA mode, in which each time the source receives an acknowledgement successfully, it increases its cwnd size by one TCP segment linearly. Furthermore, the source increases its cwnd size by one segment solely each RTT regardless the number of acknowledgments it receives during each RTT.

There are two ways to discover congestion by the TCP source, 1) When the source receives duplicate acknowledgements, and 2) When there is a retransmission timeout (ending of retransmission timeout period). When congestion is present due to receiving duplicate acknowledgments by the source, an example is presented in Figure 2.3.

It is observed in Figure 2.3 that the source sends several TCP segments to its destination, where each TCP segment has a size of 600 bytes. The source transmits the first TCP segment of size 600 bytes, and waits for an acknowledgment from the destination whether it has successfully received that segment. This is done using acknowledgment number 601, and all the TCP data segments with data range below this acknowledgment (less than 601) are already successfully received by the destination. It should be noted that acknowledgement with number 601 means that the next expected TCP segment at the destination starts with 601 and all the TCP segments up to 600 are received successfully by the TCP destination.

It is illustrated in Figure 2.3 that the source sends the TCP segment that starts with 1201 rather than 601, hence, when the destination receives this segment, it considers this segment

out of order, and generates another acknowledgement with content of 601 (duplicate acknowledgement) to its source. Then, the source either decides that the TCP segment with number 601 is out of order or lost due to occurrence of congestion in the network. Thus, the source waits longer and continues sending TCP segments to its destination. Now if the TCP source receives a third duplicate acknowledgment (acknowledgement with number 601), it declares that the TCP segment is lost, and resends the lost TCP segment after reducing its cwnd threshold value to half in order to control the congestion. Otherwise, the source declares that the TCP segment is out of order, and resends the correct TCP segment without reducing its cwnd. This will maintain the throughput performance in the network. Lastly, Figure 2.3 indicates that the TCP segment with number 601 is out of order, and the TCP segment with number 1801 is a lost segment. The TCP segments with number 1801 was indicated as a lost segment by the TCP source due to the TCP source received three duplicate acknowledgements of TCP segment number 1801 from the destination. These acknowledgements mean that the TCP segment with number 1801 does not delivered to the destination. After a third duplicate acknowledgement is received by the TCP source, the TCP source indicates the TCP segment with number 1801 is lost due to the congestion. By the way, the purpose from duplicate acknowledgements is to identify by the TCP source whether a particular TCP segment is out of order or lost [8, 81, 100].

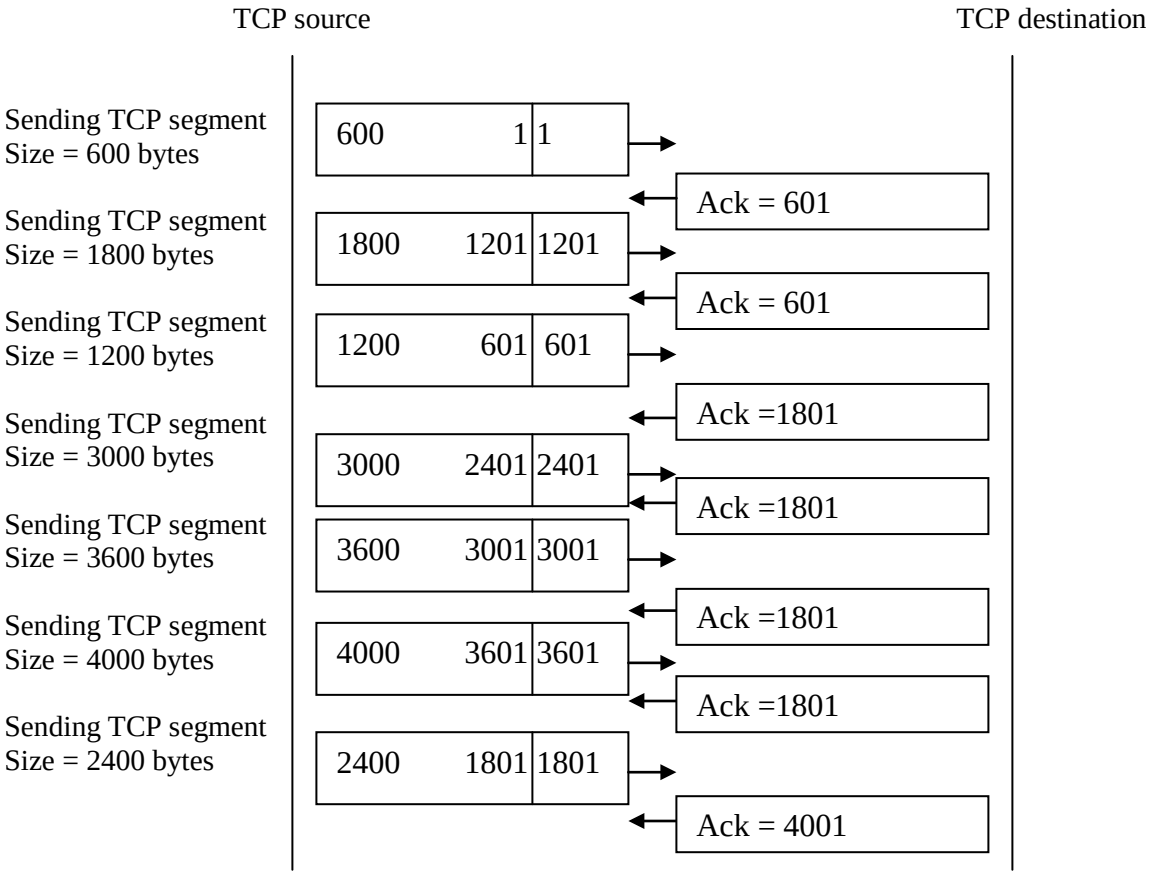

Figure 2.3: An example illustrating the reception of duplicate acknowledgements by a TCP source [104].

The second way for detecting packets lost by the source is the ending of a retransmission timeout, where each time the source sends a TCP segment, it initialises a timer, and waits until it receives an acknowledgement that the sent TCP segment is delivered to the destination. It is pointed out that the destination continues producing acknowledgments as long as TCP segments are coming to it from the source, even if these segments are out of order. For instance, if the last TCP segment is lost before reaching the destination, the destination would not be able to generate a duplicate acknowledgement since no more TCP segments can be sent by the source, and this leads to the end of a

retransmission timeout at the source. As a result of this, the source enters the SS mode after resetting the cwnd to the size of one TCP segment.

Switching from the CA mode to the SS mode takes place when one of the following two events occurs, the first event is the ending of a retransmission timeout, and the second event is the occurrence of congestion, which causes the reduction of the cwnd size multiple times. Figure 2.4 illustrates that if the cwnd size is equal to or less than the ssthreshold, the source would be in the SS mode, otherwise, it would be in the CA mode.

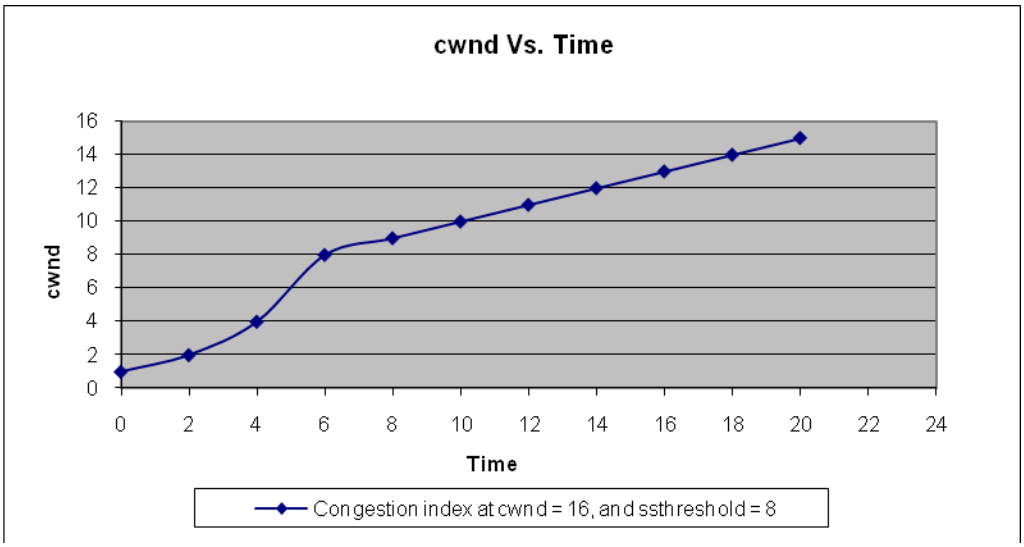

Figure 2.4: A TCP source implementation of the SS and CA mechanisms [117].

2.2.3 Fast Retransmission Algorithm

The fast retransmission (FR) algorithm [67] was proposed in order to manage congestion in the internet. As discussed in the previous section, the reason for receiving duplicate acknowledgements by the source is because either the TCP segment is out of order or it is

lost. It should be clear that the solution utilised by the source to distinguish between lost or out of order TCP segments is to wait longer to examine whether there are more duplicate acknowledgments, at least up to three duplicate acknowledgements. If the third duplicate acknowledgement is exchanged from destination to its source, then the TCP segment is lost, otherwise the TCP segment is out of order. In cases where the segment is lost, the FR algorithm starts working by sending the lost segment to the destination after shrinking the cwnd size to one segment through entering the SS algorithm. Setting the cwnd to one segment will deteriorate the TCP throughput since the TCP source only can send a single TCP segment, whereas its remaining TCP segments will be dropped.

To solve this problem, [66] has proposed a TCP congestion control algorithm to maintain the throughput performance when segments are lost. This algorithm is called the Fast Recovery algorithm [66], which will be explained in Subsection 2.2.4. On the other hand, if the segment is out of order, then no decreasing in the cwnd is applied, and therefore the throughput performance will be maintained. The FR method has several advantages such as, 1) Maintaining the throughput performance when the source is maintained its cwnd size, i.e. receiving copies from the same TCP segments by the destination will not reduce the cwnd of TCP source, therefore the throughput performance will maintain, 2) When TCP segments are lost, the source enters the CA mode rather than the SS mode and does not wait until the occurrence of a retransmission timeout, instead it retransmits the lost TCP segments as soon as it receives the third duplicate acknowledgement from the TCP destination.

2.2.4 Fast Recovery Algorithm

The fast recovery (FRec) algorithm was proposed by Van Jacobson in 1990 [66] in order to enhance the throughput performance in the internet as well as to manage congestion. As previously mentioned, the reception of duplicate acknowledgments by the source does not guarantee that the TCP segment is lost since this segment might be out of order. This explains the source waiting time to examine whether it will receive a third duplicate acknowledgement in order to declare that the TCP segment is lost. In addition, each time the destination receives an out of order segment, it generates a duplicate acknowledgement to the source; when the source receives the third duplicate acknowledgement, it declares the existence of congestion, and performs the FRec method as follows:

1. Setting the ssthreshold to half of the cwnd size, but not smaller than two segments through entering in the CA algorithm.
2. Retransmitting the lost TCP segment by implementing the FR method.
3. Setting the cwnd to the ssthreshold plus three TCP segments that caused the three duplicate acknowledgements.

In cases when the source receives less than three duplicate acknowledgements, the source increments its cwnd by TCP segments that were acknowledged by the duplicate acknowledgements. Finally, if the source receives the expected acknowledgement after receiving the third duplicate acknowledgment from the destination, it performs the following:

1. Sets its cwnd value to ssthreshold value.

2. This expected acknowledgment acknowledges all the segments that have been received by the TCP destination.
3. Remain in the CA implementation.

The FRec algorithm maintains the throughput performance when congestion occurs by initialising the CA algorithm rather than the SS algorithm and this avoid cutting its cwnd size into one segment. Therefore, many segments queued at the routers are not discarded, which leads to improving the throughput performance. After the TCP source has entered the CA algorithm, it retransmits the lost segments to its destinations utilising the FR algorithm. Basically, the implementation of the FRec algorithm includes the implementation of the CA algorithm followed by the FR algorithm. The FRec method improves the TCP performance when a single TCP segment is lost from a particular source [66]. Figure 2.5 represents an example of the implementation of all TCP congestion control algorithms (SS, CA, FR, FRec).

Figure 2.5 demonstrates that the source starts performing the SS method after establishing the TCP connection with the destination, hence, it initialises its cwnd to a single TCP segment, and increases its cwnd size by adding one TCP segment each time it receives an acknowledgement from the destination. It is noted in Figure 2.5 that the source in the SS method is increasing exponentially, until congestion occurs in the internet since the lost TCP segments existed, and this takes place when the cwnd size becomes 16. At this moment, the source stops implementing the SS method and enters the CA method, then retransmits the lost TCP segments to their destinations using the FR method.

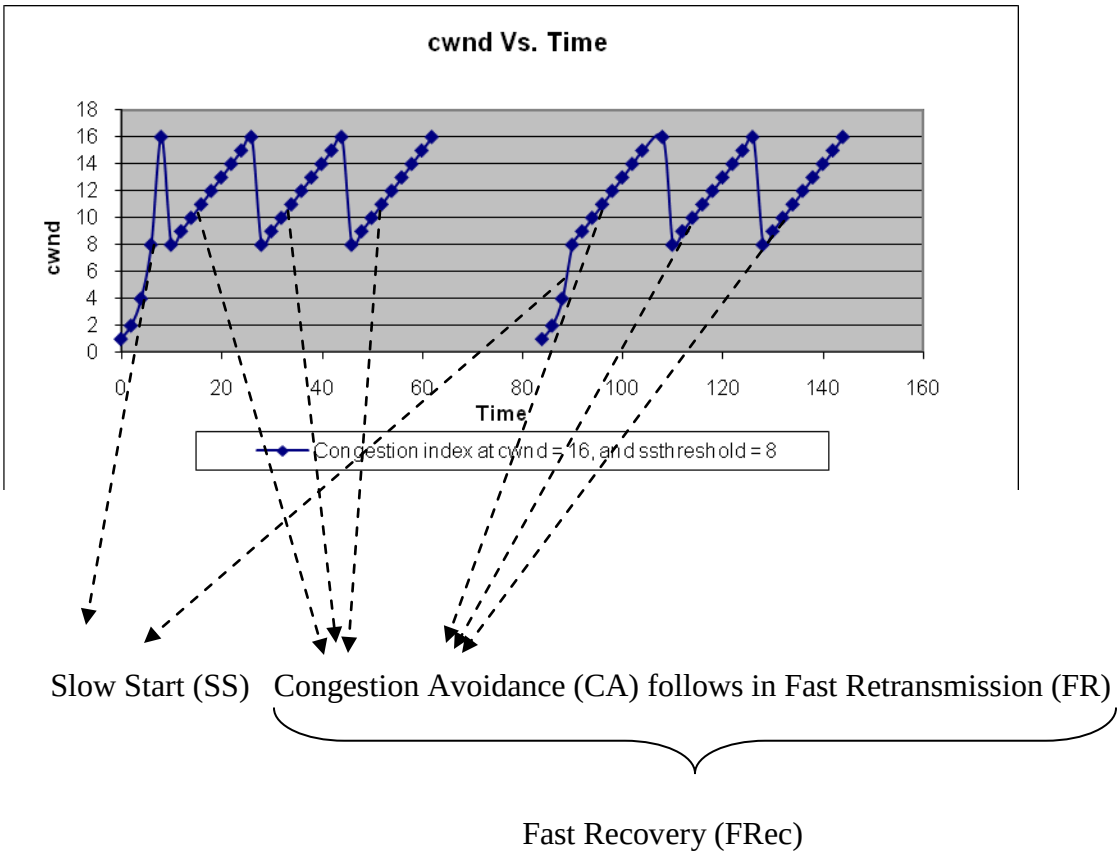

Figure 2.5: Implementation of TCP congestion control algorithms [104, 117].

The sequential implementation of CA and the FR methods is the implementation of the FRec method. This process gets repeated until a retransmission timeout appears, which may cause the shrinking of the source cwnd to a single TCP segment. As a result of that the source returns to the SS method. An example on switching from the CA to SS in Figure 2.5 is the existence of retransmission timeout in the time period [64 – 82]. Lastly, after the retransmission timeout is ended, the source enters the SS, and the source remains in the SS until its cwnd size reaches the ssthreshold value. On other hand, if the cwnd size for the source becomes larger than the ssthreshold value, the source switches from the SS to the CA. Therefore, if the cwnd size of the source is equal to or smaller than the ssthreshold

value, then the source implements the SS method, otherwise it implements the CA method. In Figure 2.5, the linearly increasing of the cwnd when the cwnd is larger than the ssthreshold represents the implementation of CA algorithm.

2.3 Congestion Control

One of the first methods which was used successfully in managing the congestion was Drop Tail (DT) [20, 22, 107], which was used in the internet for several years. The DT method relies on large sized buffers at the routers in order to achieve a good network performance. To overcome some of the DT drawbacks (discussed in Subsection 2.3.1), two types of congestion control approaches have been discussed in [110], i.e. end-to-end system and router buffer. An example of an end-to-end system is the TCP [8, 81, 92, 95, 100, 104, 108, 119] in which the TCP source detects the congestion through observing the packet dropping rate at the router buffers. The sources react to this type of congestion by reducing their transmitting rates.

In the second type of congestion control, the routers buffers play a significant role in detecting the congestion through evaluating the network behaviour [24, 31, 36, 39, 40, 43, 46, 50, 73, 84]. Particularly, the routers buffers detect the congestion by comparing the number of arrived packets with the available buffer space in which if the number of packets is greater than a certain level, then the router buffer sends a congestion notification to the

sources to reduce their transmitting rates. An example of this congestion type is the one that AQM methods deal with [16, 30, 102, 103].

2.3.1 The Drop-tail Method

The DT method [20, 22, 107] has been used for many years in the internet for controlling the congestion incidents. This method relies on setting the router buffer length in the internet to a maximum size in order to achieve high throughput. However, this may also lead to high packet queueing delay, which is not desirable especially for real time traffic [111]. On the other hand, if the router buffer is set to a small size, low queueing delay will be achieved but the throughput performance will deteriorate as well the packet loss rate increases. The network sources become aware of the congestion using the DT method solely after the router buffers overflow.

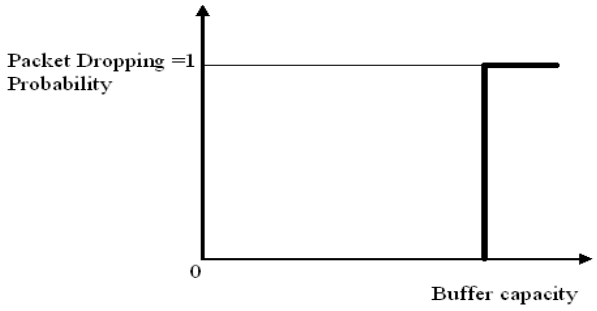

Figure 2.6: DT packet dropping probability function.

Figure 2.6 shows the D_p for the DT method, where using this method in the TCP networks may degrade the TCP performance since the sources solely adjust their transmitting rates after the DT router buffers overflow. There are other drawbacks associated with the DT method, which also degrade the TCP performance. One of these is the lockout phenomenon [20] that arises when one or more sources are monopolising the

router buffer spaces for a long time and the rest of sources are waiting. Another drawback is the full queues [20], which happens when the router buffers are almost full and they remain so for a while, thus, both packet queueing delay and dropping rate are increased. A third drawback is global synchronisation [46], which affects the throughput when the network is congested, in which the router buffers notify all the sources of the congestion in order to reduce their transmitting rates. However, since several sources have not played any role in causing the congestion, reducing their transmitting rates is unnecessary and may reduce the throughput performance.

Another important DT limitation is the bias versus bursty traffic flows. The TCP rate control favours short round trip time (RTT) flows on those with large RTT to achieve high throughput performance [42, 99, 107]. The short RTT traffic flows increase their transmission windows quicker than those with large RTT and as a result, traffic flows with short RTT increase their network resources quicker than those with large RTT. This may cause unfair share among the network flows. Moreover, traffic flows with short RTT contribute in building up router buffers quicker than those with large RTT [42, 46].

2.4 Active Queue Management Methods

The Active Queue Management (AQM) [16, 30, 102, 103] are a set of methods that have developed mainly to control congestion in networks early and to achieve high QoS for traffic loads. Examples of AQM methods are, RED [46], ARED [49], REM [9, 73], BLUE

[39, 43], Stochastic Fair BLUE (SFB) [38], GRED [50], DRED [11], SRED [90], Generalised Random Early Evasion Network (GREEN) [42, 121], Adjustment RED [110], Fuzzy BLUE [122], Fuzzy Exponential Marking (FEM) [27, 76], Early Random Drop [58] and Decbit [98] and others.

Moreover, AQM methods have been proposed to overcome the DT limitations discussed earlier. For instance, unlike the DT method, which starts dropping packets simply after the router buffers overflow, the AQM methods usually start dropping packets in early stages, hence this enables the sources to decrease their transmitting rates early before the router buffers become completely occupied. In AQM methods, the network sources identify the congestion either by dropping packets at the router buffers (implicit feedback) or by setting an Explicit Congestion Notification (ECN) bit [13, 53, 94, 96, 97, 103] in the packet header (explicit feedback). According to the ECN bit, when congestion occurs, the router buffer sets an ECN bit in the arriving packet header, and then sends the packet to its destination. When the destination receives the marked packet, it informs the source about the congestion via an acknowledgement [8, 81, 100].

[9] has suggested that one can design and implement an AQM method if the following three questions can be answered:

1.) What is the congestion metric that will help in controlling the congestion?

2.) How will the congested router buffer inform the connections to modify their transmission rates?

3.) What is the packet dropping probability function and what is the policy used for dropping the packets? i.e. linearly [46, 49, 50], exponentially [9, 73], etc.

This section is organised as follows, Subsection 2.4.1 introduces the Early Random Drop algorithm and in Subsection 2.4.2, the Decbit algorithm is presented. The RED algorithm and its advantages and disadvantages are discussed in Subsection 2.4.3. The ARED and GRED algorithms are covered in Subsections 2.4.4 and 2.4.5, respectively. Subsections 2.4.6 and 2.4.7 are devoted to the SRED and DRED algorithms respectively. Subsection 2.4.8 illustrates the Adjustment RED algorithm and Subsection 2.4.9 explains the FEM algorithm. BLUE, SFB and Fuzzy BLUE algorithms are demonstrated in Subsection 2.4.10 and Subsections 2.4.11 and 2.4.12, explain the GREEN and REM algorithms, respectively.

2.4.1 Early Random Drop Algorithm

The Early Random Drop method was proposed in 1989 as a congestion control method that uses the queue length (ql) and a fixed threshold position at the router buffer to detect the congestion [58]. Briefly, when the ql becomes larger than the fixed threshold position at the router buffer, the router buffer starts dropping the arriving packets on a constant rate basis. Furthermore, since Early Random Drop is one of the AQM methods, it starts dropping packets in the early stages before the router buffer becomes full, which eventually reduces the global synchronisation (discussed later in Subsection 2.4.3.2), and maintains the throughput in the network [16, 30, 102, 103]. An advantage of the Early Random Drop method, which is still debatable, is its ability to detect misbehaving flows (discussed later in Subsection 2.4.10.1) [58, 79, 106, 126, 127].

2.4.2 Decbit Algorithm

The Decbit [98] uses a binary indication bit and the average queue length (aql) as a congestion metric in the network. To clarify, for every packet that arrives at the router buffer, Decbit calculates the size of the current aql, if it is less than one, no packets are dropped, whereas if the aql size exceeds one, Decbit marks the arriving packet and transfers it to its destination. Finally, the marked packet is sent back via an acknowledgment from the destination to the source, and the source can identify the congestion from the acknowledgment's content. It should be noted that the sources modify their transmission windows once each two RTTs, and if half or more of the packets in the last window size are marked by the congestion indication bit, then the transmission window size decreases exponentially [9, 73], otherwise, it increases linearly [46, 49, 50].

2.4.3 Random Early Detection (RED) Algorithm

The Random Early Detection (RED) was proposed in 1993 by S. Floyd and V. Jacobson [46] gives a congestion control method, which has been successfully adopted by the Internet Engineering Task Force (IETF) in RFC 2309 [20]. RED drops packets randomly before the router buffer overflows and utilises the computed aql and two thresholds ($\min threshold$ and $\max threshold$) to manage the congestion [46]. The above two thresholds correspond to the minthreshold and the maxthreshold positions at the router

buffer, respectively. Generally, RED can detect the congestion as follows: After the router computes the aql, it compares it with the $\min threshold$ and the $\max threshold$. If the aql is less than the $\min threshold$, no congestion will occur, and therefore no packets will be dropped. Whereas if the aql is between the two thresholds, then the arriving packets will be dropped probabilistically and RED calculates the D_p in order to alleviate the congestion. Finally when the aql is above the $\max threshold$, then every arriving packet will be dropped and the D_p value becomes one. Figure 2.7 shows the RED general source code and Figure 2.8 contains the full details. It is noted that in Figure 2.8 RED uses several parameters for calculating both the aql and the D_p. These parameters can be defined as follows:

Definition 1: $current\_time$: represents the current time.

Definition 2: $idle\_time$: represents the starting idle time at the RED router buffer.

For every arriving packet at a RED router buffer, the router buffer does the following:

1. Calculate the aql.
2. Check the aql position with two thresholds ($\min threshold$ and $\max threshold$) positions at a RED router buffer, then the router performs it's right action as follows:

$if(aql < \min threshold)$
No congestion is occurred.

$if(aql \geq \max threshold)$
A heavy congestion is existed. As a result, drops/marks every arriving packet with $D_p = 1$;

$if(\min threshold \leq aql \;\&\&\; aql < \max threshold)$
Congestion is presented; therefore marks/drops every arriving packet randomly with calculating its D_p value.

Figure 2.7: The general source code for the RED method [46].

Definition 3: n: represents the number of packets sent to the RED router buffer during an idle interval time.

Definition 4: C: A counter that represents the number of packets arrived at the RED router buffer and were not dropped since the last packet was dropped.

Definition 5: D_p : represents the instantaneous packet dropping probability.

Definition 6: D_{init} : represents the initial packet dropping probability.

Definition 7: q_instantaneous : represents the instantaneous queue length.

Definition 8: qw : represents the queue weight.

Definition 9: $D_{\max}$: represents the maximum value of D_p .

Definition 10: $q(time)$: represents the linear function for the time.

Figure 2.9 displays how RED drops packets based on its congestion metric, i.e. D_{init} vs. aql . It is noticeable from Figure 2.9 that no packets are dropped when the computed aql is less than $\min threshold$. While the packets start to be dropped probabilistically when the aql is between the $\min threshold$ and the $\max threshold$, with the D_{init} value increasing linearly until the aql reaches the $\max threshold$. However, if the aql is larger than

1. Initialisation stage
   $C = -1$;
   $aql = 0.0$;
2. For every arriving packet at a RED router buffer:
   2.1 Calculate the aql for this packet at the RED router buffer.
   2.2 Examine the queue status at a router buffer, is it empty or not?
   if (The queue at a RED router buffer = = empty)
   {
   Compute n, where $n = q(current\_time - idle\_time)$;
   $aql = aql \times (1 - qw)^n$;
   }
   else
   $aql = aql \times (1 - qw) + qw \times q\_instantaneous$;
3. Determine a congestion status at the RED router buffer:
   if $(aql < \min threshold)$
   {
   $D_p = 0$; // No congestion is occurred
   Set $C = -1$;
   }
   else if $(\min threshold \le aql \,\&\&\, aql < \max threshold)$
   {
   $C = C + 1$;
   Calculate D_p value for the arriving packet as follows:
   $$D_{init} = \frac{D_{\max} \times (aql - \min threshold)}{(\max threshold - \min threshold)};$$
   $$D_p = \frac{D_{init}}{(1 - C \times D_{init})};$$
   Marks/Drops arriving packet randomly with calculating its D_p value (probabilistically) due to a RED router buffer is congested.
   Set $C = 0$;
   }
   else // *if* $(aql \ge \max threshold)$
   {
   Marks/Drops every arriving packet with $D_p = 1$ since a heavy congestion is existed at a RED router buffer;
   Set $C = 0$;
   }
4. When the RED router buffer becomes empty
   Set $idle\_time = current\_time$;

Figure 2.8: RED method in full details [46].

or equal to the $\max threshold$, RED drops every arriving packet, and the D_{init} value becomes one.

2.4.3.1 Setting qw, $\min threshold$, $\max threshold$ and $D_{\max}$ Parameters in RED

According to [46], there are guidelines for setting some of RED's parameters, i.e. qw, $\min threshold$, $\max threshold$ and $D_{\max}$. These guidelines can be explained as follows:

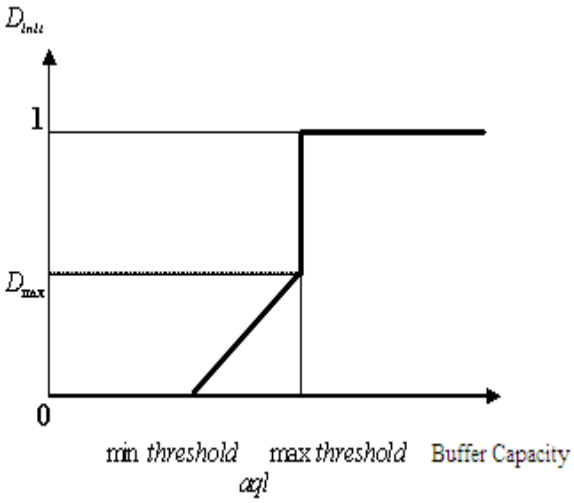

Figure 2.9: D_{init} vs. aql.

1. Setting qw: This parameter is used in calculating the aql, and has a range limit, i.e. lower limit (0.001) and upper limit (shown below). If the qw is set to a low value, the aql increases slowly, and therefore RED might not be able to detect congestion in the preliminary stages. On the other hand, the qw upper value can be explained using the following example: Assume that the RED router buffer is idle, i.e. the aql equals zero. Then consider that after a period of time, N packets have arrived at the router buffer, the aql will be calculated in this situation as follows:

$$aql(N) = \sum_{i=1}^{N} i(1-qw)^{N-i} \times qw \quad \text{..... (2.1)}$$

$$aql(N) = qw \times (1-qw)^{N} \times \sum_{i=1}^{N} i(1-qw)^{-i} \quad \text{..... (2.2)}$$

$$aql(N) = qw \times (1-qw)^{N} \times \sum_{i=1}^{N} i \times \left(\frac{1}{(1-qw)} \right)^{i} \quad \text{..... (2.3)}$$

In this case, qw upper value must be set according to equation (2.3) since $aql(N) < \min threshold$ [46].

2. Setting the $D_{\max}$ parameter: $D_{\max}$ can be known when the *aql* reaches the $\max threshold$ position at the router buffer. Generally, RED keeps the *aql* between the $\min threshold$ and the $\max threshold$ to avoid heavy congestion. [46] has set the $D_{\max}$ to a value that does not exceed 0.1 in order to control the congestion by keeping the *aql* value between the $\min threshold$ and the $\max threshold$. If the $D_{\max}$ parameter is set to a value larger than 0.1, then the RED router will not be able to keep the *aql* between the above two thresholds, and thus, every packet that arrives at the router buffer will be dropped, i.e. $D_p = 1$.

3. Setting the $\min threshold$ and the $\max threshold$ parameters: These thresholds are set to values that guarantee a desirable *aql*. The $\min threshold$ normally depends on the traffic load status, i.e. if the traffic load is fairly bursty, the $\min threshold$ is set to a large value to maintain the throughput performance. Nevertheless, if the $\min threshold$ is set to a low value, then RED will drop

packets in the early stages and will achieve a low throughput. On the other hand, setting the $\max_{threshold}$ depends on the largest queueing delay value achieved when the aql reaches the $\max_{threshold}$. Moreover, the $\max_{threshold}$ parameter is often set to at least double of the $\min_{threshold}$ value in order to maintain the throughput performance [46].

2.4.3.2 RED Advantages and Disadvantages in Controlling the Congested Routers

RED has many advantages, some of which are summarised as follows:

- The RED router avoids the bias versus bursty traffic flows problem, which arises when accepting packets from flows with short RTT and dropping packets from flows with long RTT [42, 121]. RED removes this problem using the randomisation packet dropping method [46]. It should be noted that the RED router allows bursty packets, which come from bursty flows, and accommodate them by leaving some available buffer rooms, where the bursty traffic occurs when the packet sending rates vary (low or high) or changes suddenly.
- RED avoids the global synchronisation phenomenon [127], which occurs when all the sources in the network are notified of congestion. This makes all the sources react to the congestion by reducing their transmission windows sizes to alleviate it.

However, this solution is not the best due to the fact that many of these sources may have not contributed in causing this congestion. Consequently, it is not necessary for these sources to reduce their transmission windows sizes and could result in decreasing the throughput. The RED router avoids the global synchronisation problem using a randomisation method in dropping packets, which the RED router buffer notifies only the sources that caused the congestion in the network, and these sources reply to this notification by decreasing their transmitting rate.

- The RED router can determine if a flow is misbehaving, i.e. has more than its fair share of network resources, by monitoring its arriving packets. When the router starts dropping packets because of the congestion, the D_p of a source represents the amount of bandwidth shared at the router [46]. Thus, the RED router can determine which sources have the largest D_p at the router and therefore flows with the largest D_p represents those that have the largest share in the bandwidth, and thus the RED router can detect the misbehaving flows. Examples on misbehaving flows are the TCP sources which share large amount of bandwidth at the router, and have either a large windows size or a short RTT.

Despite the advantages of RED which have been discussed above, it suffers from the following disadvantages according to [83, 84, 90]:

- The aql varies according to the level of the congestion and could be sensitive to RED's initial parameter values. For instance, when a heavy congestion occurs, this indicates that the aql is close to the $\max threshold$ position at the router buffer.

Whereas, if the congestion happens in preliminary stages, then the aql will be close to the $\min threshold$. In other words, the aql cannot be predicted since the average queueing delay depends on the RED parameters and the traffic flow status (heavy or light).

- Similarly, the throughput fraction relies on the RED parameter settings and the traffic flow status. When the aql is above the $\max threshold$, the router will drop every arriving packet, and therefore the throughput fraction in the network will be decreased to alleviate the congestion.
- The RED algorithm cannot stabilise the aql between the $\min threshold$ and the $\max threshold$ position at the router buffers when the traffic loads change suddenly, for instance for bursty traffic. Thus, the aql may exceed the $\max threshold$ position and that will lead to frequent dropping of packets at the router buffers.
- The RED congestion measure (aql) depends on the amount of traffic load; when the traffic load increases this causes an increase in the aql and, consequently, the aql could became bigger than the $\max threshold$. Therefore, every arriving packet will be dropped.

2.4.4 Adaptive Random Early Detection (ARED) Algorithm

The Adaptive Random Early Detection (ARED) [40, 41] was proposed in order to address some of RED's disadvantages, which have been discussed in Subsection 2.4.3.2. The main contribution in ARED is stabilising the *aql* between the $\min threshold$ and the $\max threshold$ through a dynamic modification of the $D_{\max}$ parameter [49]. As a result, the ARED algorithm avoids discarding arriving packets by preventing the *aql* value from reaching the $\max threshold$. A modified version of ARED [49] was developed in 2001 and has the following properties, and it differs with the original ARED in the fourth property.

1. The $\min threshold$ is set in a straightforward manner similar to [46].
2. The $\max threshold$ is set automatically based on $\min threshold$, and the queue weight (qw) is set similar to [46].
3. The $D_{\max}$ is adopted from [49] in order to stabilise the *aql* at the target level, i.e. (T_{aql}), where $T_{aql} = \frac{(\max threshold + \min threshold)}{2}$. $D_{\max}$ is always kept between [0.01, 0.5].
4. The additive increase multiplicative decrease (AIMD) [49] method is used rather than the multiplicative increase multiplicative decrease (MIMD) method [40, 41] in setting the $D_{\max}$ value.

The source code of the modified ARED algorithm is shown in Figure 2.10, where the period of time parameter is set to 0.5 seconds [49], and the T_{aql} is restricted to the

following interval: $\left\{\begin{matrix}\min threshold + 0.4.(\max threshold - \min threshold), \min threshold \\ + 0.6.(\max threshold - \min threshold)\end{matrix}\right\}$

[49]. Finally $d1$ and $d2$ correspond to the amount removed from the $D_{\max}$ when the aql is below the T_{aql} and the amount added to the $D_{\max}$ when the aql is above the T_{aql}, respectively. $d1 = 0.9$ and $d2 = \min imum\left(0.01, \frac{D_{\max}}{4}\right)$.

2.4.4.1 Tuning and Setting the Modified ARED Parameters

1. $D_{\max}$ parameter: The $D_{\max}$ value is restricted to the period [0.01, 0.5] and it is used each period of time as shown in Figure 2.10. The reason for selecting "0.5" as an upper bound for $D_{\max}$ is due to the fact that "0.5" is the maximum D_p value and it is not recommended to set $D_{\max}$ to a value larger than "0.5" since this may eliminate more than half of the arriving packets [49].

Every period of time in seconds
$if(aql < T_{aql} \;\&\&\; D_{\max} \geq 0.01)$
$D_{\max} = D_{\max} \times d1;$ // Decrease $D_{\max}$ value
$esleif(aql > T_{aql} \;\&\&\; D_{\max} \leq 0.50)$
$D_{\max} = D_{\max} + d2;$ // Increase $D_{\max}$ value

Figure 2.10: ARED algorithm source code [49].

2. $d1$ and $d2$ parameters: The ARED authors indicate that the $D_{\max}$ value must equal at least log 0.02 / log $d1$ period of time in seconds in order to decrease it from 0.5 to 0.01, i.e. in 20.1 seconds. But, the $D_{\max}$ needs at least $0.49/d2$ period of time in

seconds to increase it from 0.01 to 0.5, which is equivalent to $\left\lceil \frac{0.49}{\min imum\left(0.01, \frac{D_{\max}}{4}\right)} \right\rceil = 24.5 \sec onds.$ The default values of $d1$ and $d2$ are 0.9 and $\min imum\left(0.01, \frac{D_{\max}}{4}\right)$, respectively. To tune $d1$ and $d2$ parameters, D is considered as the packet dropping and it is assumed that $D_{\max} > D$. Then when the aql is larger than T_{aql}, $D_{\max}$ is increased by $d2$ and the aql is decreased from $\min threshold + \frac{D}{D_{\max}}.(\max threshold - \min threshold)$ to $\min threshold + \frac{D}{(D_{\max} + d2)}.(\max threshold - \min threshold)$. This decreasing in the aql value is equivalent to a value $\frac{D.d2}{D_{\max}.(D_{\max} + d2)}.(\max threshold - \min threshold)$. As long as the decreasing in the aql is less than $0.2.(\max threshold - \min threshold)$, the recommended value for $d2$ is either $d2 < 0.25.D_{\max}$ or $\frac{d2}{(d2 + D_{\max})} < 0.2$. Further, the recommended value for $d1$ is either $d1 > 0.83$ or $\frac{(1 - d1)}{d1} < 0.2$.

3. The $\min threshold$, $\max threshold$, T_{aql} and qw parameters: The $\min threshold$ is set in a straightforward manner similar to [49], and the $\max threshold$ and the T_{aql} parameters are set automatically. For instance, [52] sets the $\max threshold$ and T_{aql}

to $3.\min threshold$ and $\left(\begin{array}{l} 2.\min threshold \\ OR \\ \dfrac{(\max threshold + \min threshold)}{2} \end{array} \right)$, respectively. Finally the qw parameter is set according to [46], i.e. to 0.002.

Finally, the ARED method has some limitations, such as 1) more parameters to deal with, 2) it cannot stabilise the aql between $\min threshold$ and $\max threshold$ when heavy congestion is present, 3) parameterisation, the ARED must set its parameters to specific values in order to obtain a satisfactory performance and 4) using the aql by the ARED, if the aql is less than the $\min threshold$ and heavy congestion occurs the aql will take time to build up, whereas the router buffer is likely to be overflowing, Thus, no packets can be dropped, although the RED router buffer is already overflowing.

2.4.5 Gentle Random Early Detection (GRED) Algorithm

To overcome some of the RED's drawbacks discussed earlier, the GRED algorithm [50] was proposed, where its main aim is to stabilise the aql at a certain level. GRED employs the RED approach in calculating the D_P. The pseudocode for the GRED is shown in Figure 2.11 that contains three main thresholds ($\min threshold$, $\max threshold$, double $\max threshold$) and the target level for the average queue length (T_{aql}).

According to Figure 2.11, when the *aql* at the router is below the $\min threshold$, no packets are dropped. Whereas, if the *aql* is between the $\min threshold$ and the $\max threshold$, the router will drop the arriving packets randomly similar to RED. Moreover, if the *aql* is between the $\max threshold$ and $double \max threshold$, the router will drop the arriving packets randomly. It should be noted that in this case the *aql* is stabilised at T_{aql} (the value of the T_{aql} is computed by equation (2.4) [52]). Finally, if the *aql* is equal to or greater than the $double \max threshold$, the GRED router marks the arriving packets with $D_p = 1$.

Equations 2.5 and 2.6 are examples on the setting values for $\max threshold$ and the $double \max threshold$, respectively, and the $\min threshold$ and qw are given values similar to those of RED [46].

$$T_{aql} = 2 \min threshold = \frac{(\min threshold + \max threshold)}{2} \quad \text{..... (2.4)}$$

$$\max threshold = 3 \min threshold \quad \text{..... (2.5)}$$

$$double \max threshold = 2 \max threshold \quad \text{..... (2.6)}$$

Lastly, GRED has some limitations; these limitations are 1) more thresholds to deal with, 2) parameterisation and 3) using the *aql* by the GRED. The last two limitations have been explained in the end of ARED subsections.

2.4.6 Stabilised Random Early Drop (SRED) Algorithm

The Stabilised Random Early Drop (SRED) algorithm [90] was proposed to overcome some of RED's limitations especially the computed *aql* dependency on the number of TCP connections [46]. This problem happens when the number of TCP connections is large, which makes the *aql* to become large as well, i.e. $aql \geq \max threshold$, and consequently allows the RED router to drop every arriving packet [46]. The SRED algorithm treats this problem by estimating the number of active TCP sources, i.e., sources with at least a single packet at the router buffer. Another of RED's limitation that SRED overcomes is the ability to distinguish the misbehaving flows in the networks (discussed earlier in Subsection 2.4.3.2). The SRED identifies the misbehaving flows and limits their

1. Initialisation stage
   $C = -1$;
   $aql = 0.0$;
2. For every arriving packet at a GRED router buffer:
   2.1 Calculate the aql for this packet at a GRED router buffer.
   2.2 Examine the queue status at the router buffer is it empty or not?
   if (The queue at the GRED router buffer = = empty)
   {
   Compute n, where $n = g(current\_time - idle\_time)$;
   $aql = aql \times (1 - qw)^n$;
   }
   else
   $aql = aql \times (1 - qw) + qw \times q\_instantaneous$;
3. Determine a congestion status at the GRED router buffer:
   $if(aql < \min threshold)$
   {
   $D_p = 0$; // No congestion is occurred
   Set $C = -1$;
   }
   $elseif(\min threshold \leq aql \;\&\&\; aql < \max threshold)$
   {
   $C = C + 1$;
   Calculate D_p value for the arriving packet as follows:
   $$D_{tmp} = \frac{D_{max} \times (aql - \min threshold)}{(\max threshold - \min threshold)};$$
   $$D_p = \frac{D_{tmp}}{(1 - C \times D_{tmp})};$$
   Marks/Drops arriving packet randomly using its calculated D_p value (probabilistically);
   Set $C = 0$;
   }
   $elseif(\max threshold \leq aql \;\&\&\; aql < double\max threshold)$
   {
   $C = C + 1$;
   Calculate D_p value for the arriving packet as follows:
   $$D_{tmp} = D_{max} + \frac{(1 - D_{max}) \times (aql - \max threshold)}{(double\max threshold - \max threshold)};$$
   $$D_p = \frac{D_{tmp}}{(1 - C \times D_{tmp})};$$
   Marks/Drops arriving packet randomly using its calculated D_p value (probabilistically);
   Set $C = 0$;
   }
   $else$ // $if(aql \geq double\max threshold)$
   {
   Marks/Drops every arriving packet with $D_p = 1$ since a heavy congestion is exi
   Set $C = 0$;
   }
4. When a GRED router buffer becomes empty
   Set $idle\_time = current\_time$;

Figure 2.11: GRED mechanism within details [50].

transmitting window sizes as explained below.

The congestion metrics used by SRED are the estimated number of active flows, and the current ql. One of SRED's aims is stabilising the current ql at a certain level regardless of the number of active TCP connections in the network [11, 90]. To do so, a list of flows at the router buffer called the zombie list is used, where every packet that arrives at the router buffer is added as a record to the zombie list. Moreover, not only the packet is added, also its information, i.e. the flow identifier, count, timestamp, etc. When the zombie list becomes full, every arriving packet is then compared randomly with a selected packet from the zombie list. If both packets match (belong to the same flow), a hit is declared, the count increases by 1, and the timestamp is set to the packet arrival time. Otherwise, a miss is declared, the count and timestamp are set to 0 and the packet arrival time to the router buffer, respectively. The hit values are used to identify the misbehaving flows since they cause hits more than the behaving flows [90]. In other words, when packets arrive at the SRED router from a flow and a packet matches several zombie records, many hits are produced and the count is incremented many times for that flow. Now, from the definition of misbehaving flows (flows which have gained more than their fair share in bandwidth), flows with high count values and several hits are those that are likely to be misbehaving.

Moreover, the SRED authors estimate the frequency of hits $P(m)$ for the $\frac{N}{D}$ packets, where D denotes the probability that two packets are not a match, and N denotes the size of the zombie list. For a packet m, the hit function is given in equation (2.7).

$$Hit(m) = \begin{cases} 1, \text{ if hit exists} \\ \\ 0, \text{ if hit does not exist} \end{cases} \quad \ldots\ldots (2.7)$$

Then $P(m) = P(m-1)(1-\delta) + Hit(m)\delta$ …………………………………………….. (2.8)

where $0 < \delta < 1$, and $\delta \approx \frac{D}{N}$ …………………………………………………………. (2.9)

δ parameter was set as in [90], which shown in equation (2.10) below.

$$\delta \approx \frac{1}{N} \quad \ldots\ldots (2.10)$$

After calculating $P(m)$, the number of the active flows is computed by taking the inverse of $P(m)$ value, i.e. $\left(P(m)^{-1}\right)$, where $P(m)^{-1}$ denotes the number of estimated active flows before packet m has arrived [90]. After $P(m)^{-1}$ is computed, the $D_{sred}(ql)$ value can be calculated using equation (2.11).

$$D_{sred}(ql) = \begin{cases} 0, \text{ If } 0 \le ql < \frac{k}{6} \\ \\ \frac{D_{\max}}{4}, \text{ If } \frac{K}{6} \le ql < \frac{K}{3} \\ \\ D_{\max}, \text{ If } \frac{K}{3} \le ql \le K \end{cases} \quad \ldots\ldots (2.11)$$

where K = the capacity of the buffer, $D_{\max}$ = the maximum probability value for packets dropping. Furthermore, the $D_{\max}$ parameter is often set to 0.15 and sometimes to 0.09 or 0.12 [90]. If the $D_{\max}$ is set to a value larger than 0.15, a high D_p will result. D_p can be

calculated according to $P(m)^{-1}$ and the ql and whenever a packet, i.e. m, arrives at the router, $P(m)$ is updated and the number of estimated active flows is computed. Now, based on $P(m)^{-1}$ and the ql, the SRED router decides whether to drop m. The D_p function is shown in equation (2.12) when $0 < P(m) < \frac{1}{256}$.

$$D_p = D_{sred}(ql).\min imum\left(\frac{1}{(256.P(m))^2},1\right) \quad \text{..... (2.12)}$$

From equation (2.11), $D_{sred}(ql)$ relies on three values $\left(D_{\max}, \frac{D_{\max}}{4}, 0\right)$. When the ql is below the $\frac{k}{6}$ location at the SRED router, no packets are dropped, but if the ql is between $\frac{k}{6}$ and $\frac{K}{3}$, then $\frac{D_{\max}}{4}$ is the selected value for the $D_{sred}(ql)$ function and is applied it in equation (2.12) to compute the D_p. Finally, if the ql reaches $\frac{K}{3}$, the $D_{\max}$ value is selected for the $D_{sred}(ql)$ function and it is applied in equation (2.12) to calculate D_p. Also, in equation (2.12), the selection of 256 has been made stochastically, and needs further investigation [90]. Finally, when $1 \geq P(m) \geq \frac{1}{256}$, D_p can be estimated from equation (2.13). Another version of SRED, called full SRED, was proposed in [90]. This version is similar to SRED but depends on the number of estimated active flows, ql and the hit function, in computing the D_p. The D_p function for the full version is shown in equation (2.14).

$$D_p = \frac{D_{sred}(ql)}{65.536}.\frac{1}{P(m)^2} \approx \frac{D_{sred}(ql)}{65.536}(\#of\_flows)^2 \quad \text{(2.13)}$$

$$D_p = D_{sred}(ql).\min imum\left(\frac{1}{(256.P(m))^2},1\right).\left(\frac{Hit(m)}{P(m)}+1\right) \quad \text{(2.14)}$$

It is obvious from equation (2.14) that the full SRED version uses the hit function immediately in order to evaluate the D_p result , whereas the simple version of SRED has not exploited the hit function directly in obtaining the D_p, rather it initially applies the hit function in equation (2.8) to find out the $P(m)$. Then, through the $P(m)^{-1}$, the D_p value is generated (equations 2.12 and 2.13).

2.4.7 Dynamic Random Early Drop (DRED) Algorithm

The Dynamic Random Early Drop (DRED) [11] was proposed in 2001 to overcome the aql dependency on the number of TCP connections [20, 46, 100]. Similar to most AQM methods, the aim of the DRED is stabilising the ql at a predetermined level regardless the number of TCP connections in the network. Moreover, DRED depends on a fixed unit of time (Ct), where for every Ct, ql and the error signal $Err(i)$ are computed, respectively. The calculated $Err(i)$ shown in equation (2.15) depends on both ql and T_{ql}.

$$Err(i) = ql(i) - T_{ql} \quad \text{(2.15)}$$

Then, based on the $Err(i)$ value, the filtered error signal $Fil(i)$ is found as follows:

$$Fil(i) = Fil(i-1).(1-qw) + Err(i).qw \quad \text{(2.16)}$$

It can be observed from equation (2.16) that DRED uses a low pass filter similar to that of RED in order to calculate $Fil(i)$, where the low pass filter is an exponential weighted moving average (EWMA) [46]. The qw parameter denotes the queue weight and it is used to estimate the $Fil(i)$ value. This $Fil(i)$ can be utilised every unit of time to update D_P. Finally, the capacity of the DRED router buffer (K) and a parameter called ε is used to identify the feedback gain of D_p [11], which is shown in equation (2.17).

$$D_p(i) = \min imum\left\{\max imum\left(D_p(i-1) + \varepsilon.\frac{Fil(i)}{K}, 0\right), 1\right\} \quad \text{..... (2.17)}$$

Further, [11] has indicated that the c value gets updated only when ql is larger than or equal to the drop threshold (th) in order to achieve high link utilisation. In other words, no packet dropping occurs when $ql < th$, which indicates that ql is a crucial parameter for detecting the congestion in DRED [11]. Finally, DRED marks packets by adding an ECN bit [13, 53, 94, 96] in their headers or drops packets at its router buffers. The flow chart shown in Figure 2.12 explains how D_p gets calculated according to [24].

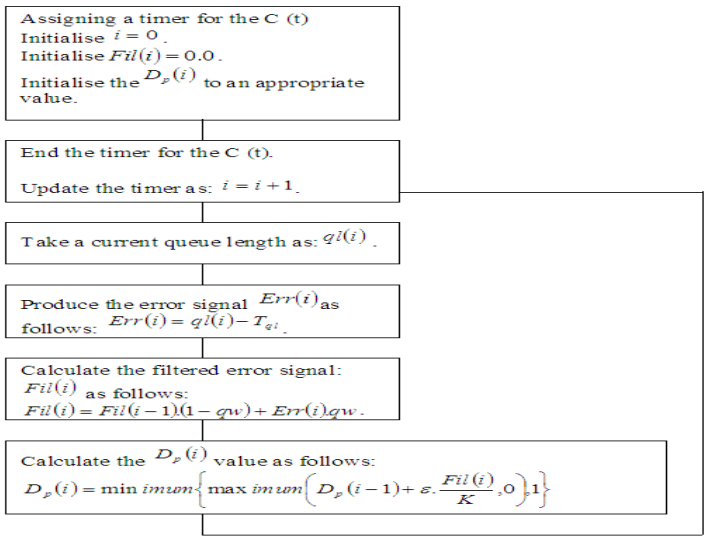

Figure 2.12: The flow chart of the D_p calculations for the DRED [24].

2.4.7.1 Tuning and Setting the DRED Parameters

Table 2.1 shows the DRED parameters with their recommended values from [24].

Table 2.1: DRED parameters from [24].

The DRED parameter	Setting the DRED parameters
Units of time C (t)	Time for sending 10 packets or another appropriate value.
Target level for the queue length (T_{ql})	$T_{ql} = \frac{1}{2} \times K$.
Indication congestion threshold	$th = 0.9 \times T_{ql}$.
The buffer capacity at the DRED router	K .
Queue weight (qw)	0.002.
System control parameter (ε)	0.00005.

The main limitations of DRED are, 1) the DRED has to set its parameters to specific values to provide a good performance (parameterisation) and 2) DRED uses the instantaneous queue length rather than the average queue length and this may result in unnecessary probabilistic dropping of packets when the congestion is of a very transient nature.

2.4.8 The Adjustment RED

Many AQM methods, including RED [46, 65], DRED [11], BLUE [39, 43] and SRED [90] use output queues (OQs) as one of the congestion metrics. However, the authors of [110] have used a combination of input queues (IQs) and OQs at the routers as congestion metrics

in an algorithm called Adjustment RED. The IQs are queues that exist at the input routers side. The adjustment RED replaces the ql parameter in the aql equation of RED with a weighted total of IQs and OQs as shown in equation (2.18).

In equation (2.18), qw denotes the queue weight, IQ corresponds to the total virtual OQs, and ω represents a weighting parameter used with IQs to help supplying a balance between the packet loss rate and the aql [110]. The source code for the adjustment RED algorithm is shown in Figure 2.13.

Initialisation stage
$C = -1;$
$aql = 0.0;$
2. For every arriving packet at a router buffer:
2.1 Calculate the aql for this packet at a router buffer.
2.2 Examine the queue status at the router is it empty or not?
if (The queue at the router buffer = = empty)
{
Compute n, where $n = q(current\_time - idle\_time)$;
$aql = aql \times (1 - qw)^n$;
}
else
$aql = aql \times (1 - qw) + qw \times (\omega \times IQ + OQ)$;
3. Determine a congestion status at the router buffer:
if $(aql < \min threshold)$
{
$D_p = 0$; // No congestion is occurred
Set $C = -1$;
}
elseif $(\min threshold \le aql \;\&\&\; aql < \max threshold)$
{
$C = C + 1$;
Calculate D_p value for the arriving packet as follows:

$$D_{init} = \frac{D_{max} \times (aql - \min threshold)}{(\max threshold - \min threshold)};$$

$$D_p = \frac{D_{init}}{(1 - C \times D_{init})};$$

Marks/Drops the arriving packet randomly using its calculated D_p value (probabilistically).
Set $C = 0$;
}
else // *if* $(aql \ge \max threshold)$
{
Marks/Drops every arriving packet with $D_p = 1$ since a heavy congestion is existed;
Set $C = 0$;
}
4. When the router buffer becomes empty
Set $idle\_time = current\_time$;

Figure 2.13: The adjustment RED mechanism source code.

$$aql = aql \,.(1 - qw) + qw \,.(\omega.IQ + OQ) \quad \text{..... (2.18)}$$

2.4.9 Fuzzy Explicit Marking Approach in Differentiated Services Networks

Before the Fuzzy Explicit Marking (FEM) algorithm [27] is presented the differentiated services (Diff-Serv) architecture and how to control the congestion within the Diff-Serv network is discussed.

2.4.9.1 Differentiated Services Architecture

The IETF developed a new internet architecture model called Diff-Serv [18] aiming to deliver a satisfactory QoS for different types of services in the IP networks. In this architecture, the traffic is split into different classes of services, and each class is treated independently, which makes its QoS different [18]. In [62], the Diff-Serv architecture has two group types, the first type is called the assured forwarding per hop group (AFG), where the AFG is used to deliver the IP packets for separate service classes, i.e. best effort traffic and assured traffic. The primary goal for the AFG is to supply a satisfactory QoS for different classes of services. When the network becomes congested, the routers start dropping packets from the traffic class that caused the congestion. Particularly, consider a network that consists of two classes of services (best effort and assured). When the congestion occurs in the network, the routers drop packets from the best effort traffic since it has a lower priority than the assured traffic class [27, 29].

The second Diff-Serv architecture group type is called the expedited forwarding per hop group (EFG), where in this group the traffic classes ask the network routers to provide enough resources in order to provide a good QoS for the different service classes. Now the question is: How the routers can discriminate between the different service classes in order to make the right decision of whether to drop the arriving packets? The authors of [86] have answered this question using the so called the type of service (ToS) field's bits. The ToS bits are set in the header of an arriving packet in order to determine the class of service to which it belongs. This enables the routers to drop the arriving packets according to 1) the congestion level in the network and 2) The class of service.

2.4.9.2 Controlling the Congestion Within the Diff-Serv. Network

The authors of [27] have used the RED algorithm to manage the congestion in the Diff-Serv. network. An example is presented on how RED and RED input output (RIO) [27, 28, 31] (discussed below) algorithms manage the congested networks. It is assumed that the network has three different service classes (best effort, assured, expedited) and a leaky bucket traffic shaper that is used to examine whether the arriving packets comply with the service level agreement (SLA). Figure 2.14 demonstrates how RED and RIO control the congestion in the Diff-Serv. network.

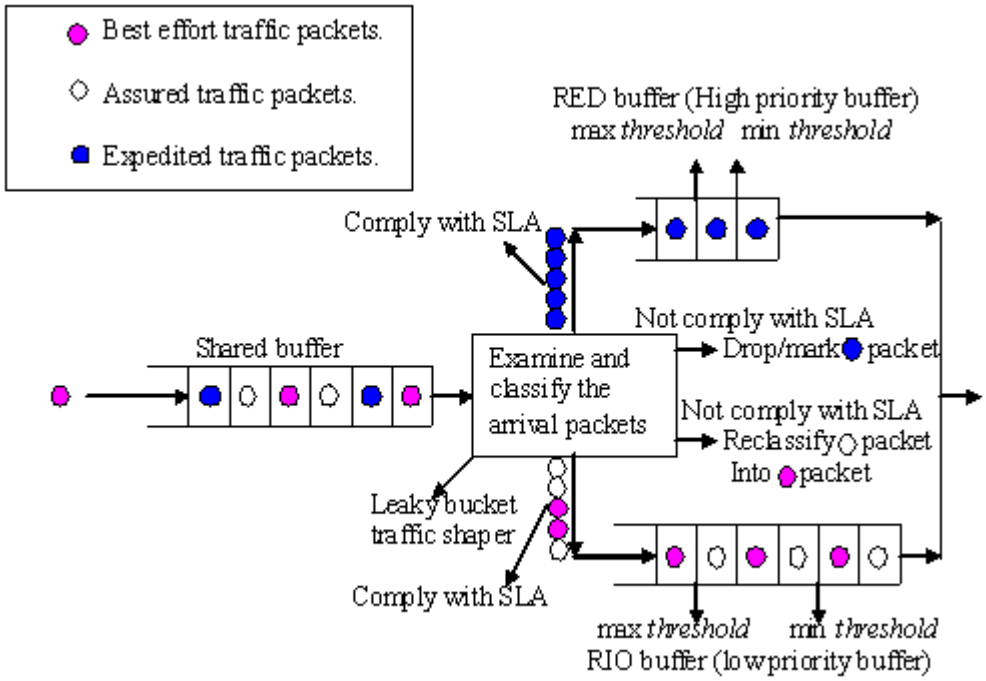

Figure 2.14: RED and RIO algorithms in controlling congestion in a Diff-Serv. Network [27].

It can be observed in Figure 2.14 that the arriving packets are queued and shared in a single buffer called “the shared buffer”, regardless of the classes of service they belong to. During the queueing, the packets wait until they get examined by the leaky bucket traffic shaper to check whether they comply with the SLA. If the packets from the expedited traffic class comply with the SLA, they get forwarded to the RED’s high priority buffer. Otherwise, they will be dropped. In cases where packets from the assured traffic class comply with the SLA, they will be forwarded to the RIO low priority buffer; otherwise they will be reclassified as best effort packets and then transmitted back to the RIO low priority buffer. Both the assured and the best effort packets share the RIO low priority buffer. RIO means RED Input/Output packets of the connection conference agreement [27].

RIO uses a similar packet dropping policy to RED with minor differences such as RIO has different positions for the $\min$ *thresholds* and the $\max$ *thresholds* at the low priority buffer. The $\min$ *threshold* and the $\max$ *threshold* positions for RIO's input buffer differs from those for its output buffer. Moreover, the initial packet dropping probabilities (D_{init}) for the input and output buffers are shown in Figures 2.15 and 2.16 respectively. RIO marks candidate packets that might be dropped either by inserting an ECN bit [13, 53] in their headers or by discarding them randomly at the low priority buffer.

Finally, each class of service has different positions for their $\min$ *threshold* and $\max$ *threshold* at RED and RIO buffers. For instance, the best effort class has the lowest position for the $\min$ *threshold* at RIO's buffer, which enables it to drop packets early.

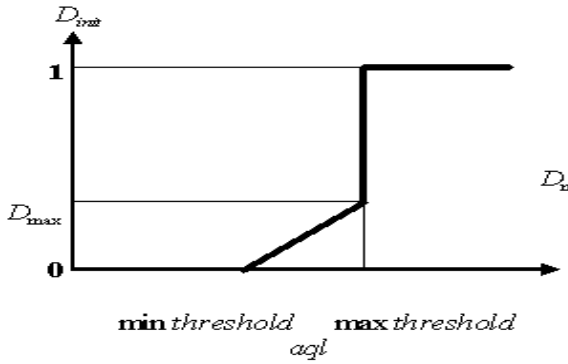

Figure 2.15: D_{init} calculation for the input buffer [27].

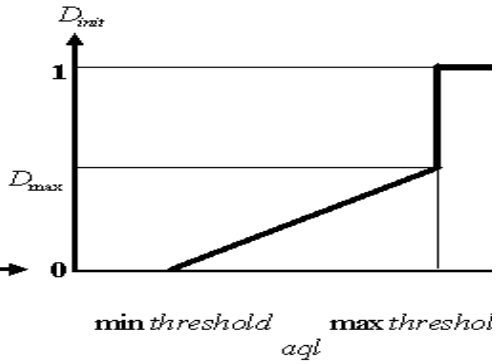

Figure 2.16: D_{init} calculation for the output buffer [27].

Moreover, the best effort class has the highest position for the $\max$ *threshold* at RIO'sbuffer, which enables it to drop packets with the highest D_P value.

2.4.9.3 Fuzzy Explicit Marking (FEM) Algorithm

The Fuzzy Explicit Marking (FEM) algorithm [27] is an AQM algorithm that uses fuzzy logic [15, 17, 61, 85, 93, 125] and RED to manage congestion incidents in the networks. FEM was primarily proposed to overcome one of RED's problems which occurs when the aql is larger than the $\max threshold$ and this will degrade the RED performance. FEM replaces the positions of the $\min threshold$ and the $\max threshold$ at the router buffers using a fuzzy inference process (FIP) [29, 78, 85] which gets implemented at each router buffer and for each class of service. Furthermore, FEM calculates D_p using two input linguistic variables (current buffer length, changing rate in the buffer), and then D_p will be represented as an output linguistic variable.

Figure 2.17 shows the fuzzy sets for the input and output linguistic variables. In order to calculate D_p, FEM uses different sets of fuzzy logic rules built by domain experts for each class of service besides the input linguistic variables. Figures 2.18 and 2.19 display the fuzzy logic rules for the best effort and the assured classes of service, respectively. The membership functions and their degrees are specified by experts in the congestion control domain. Moreover, the membership function shapes for the input and output linguistic variables are trapezoidal [71] or triangular [85] and these shapes provide simple computation of the crisp value for the output linguistic variable.

The process of computing D_p in FEM can be briefly summarised as follows: Whenever a packet arrives at the router buffer, the class of service that the arrived packet belongs to and its associated fuzzy logic rules decide whether to drop the arrived packet. Then, FIP

router calculates the D_p value dynamically when it receives the input linguistic variables, and the fuzzy logic rules sets for each class of service. It should be noted that FEM marks the arrival packet by adding an ECN bit to its header.

Current buffer length (CBL) = {empty, low, moderate, full}.
Change rate in the buffer (CRB) = {zero, decreasing, increasing}.
Packets dropping/marking probability (D_p) = {zero, low, medium, high}.

Figure 2.17: The input and output linguistic variables fuzzy sets in FEM mechanism [27].

Figures 2.19 and 2.20 show that rules belong to the best effort class are more active than those of the assured class with respect to dropping the arrival packets. In computing D_p and in cases of the best effort class, FEM does not drop packets until the CBL becomes moderate (fuzzy set) and the CRB is in a zero fuzzy set. In this case, the calculated D_p becomes in a low fuzzy set. However, when the CBL becomes in a full fuzzy set, D_p then will depend on the CRB status, if the CRB is in a decreasing fuzzy set, D_p value becomes in a medium fuzzy set, whereas, if the CRB is in either a zero or an increasing fuzzy set,

If CBL is empty then D_p value is zero
If CBL is low and CRB is decreasing then D_p value is zero
If CBL is low and CRB is zero then D_p value is zero
If CBL is low and CRB is increasing then D_p value is zero
If CBL is moderate and CRB is decreasing then D_p value is zero
If CBL is moderate and CRB is zero then D_p value is low
If CBL is moderate and CRB is increasing then D_p value is medium
If CBL is full and CRB is decreasing then D_p value is medium
If CBL is full and CRB is zero then D_p value is high
If CBL is full and CRB is increasing then D_p value is high

Figure 2.18: The set of fuzzy logic rules for the best effort class of service [27].

If CBL is empty then D_p value is zero
If CBL is low and CRB is decreasing then D_p value is zero
If CBL is low and CRB is zero then D_p value is zero
If CBL is low and CRB is increasing then D_p value is zero
If CBL is moderate and CRB is decreasing then D_p value is zero
If CBL is moderate and CRB is zero then D_p value is zero
If CBL is moderate and CRB is increasing then D_p value is low
If CBL is full and CRB is decreasing then D_p value is low
If CBL is full and CRB is zero then D_p value is low
If CBL is full and CRB is increasing then D_p value medium

Figure 2.19: The set of fuzzy logic rules for the assured class of service [27].

D_p value becomes in a high fuzzy set. On the other hand and for the assured class of service, FEM does not drop packets until the CBL becomes in a moderate fuzzy set and the CRB is in an increasing fuzzy set. In this case, D_p value becomes in a low fuzzy set. The same D_p result was achieved when the CBL is in a full fuzzy set and the CRB is in either a decreasing or a zero fuzzy set, whereas if the CBL is in a full fuzzy set and the CRB is in an increasing fuzzy set, then D_p becomes in a medium fuzzy set.

2.4.10 BLUE Algorithm

The BLUE algorithm [39, 43] was proposed to enhance the performance of the classic RED algorithm. BLUE relies on a single D_P and a certain threshold, where if the buffer length at the router is larger than the threshold, BLUE increases D_p to manage the congestion. Further, if the buffer length is empty or the link is idle, then D_p will be decreased. Unlike RED, which utilises the aql as a main congestion metric [46], BLUE relies on the packet loss, the link utilisation and the buffer length at the router [39, 43]. The packet loss and the buffer length are tuned by adjusting D_p, whereas, the link utilisation is achieved through the link status. Similar to RED, BLUE avoids the problems associated with bursty traffic flows by allowing space to accommodate the bursty packets in its router buffer. The BLUE pseudocode is shown in Figure 2.20. Figure 2.20 shows that BLUE uses several parameters to adjust the D_p value, i.e. a $freeze\_time$ is used to determine the lowest time period

between two successive D_p adjustments. The $freeze\_time$ can be set as a fixed parameter according to [39, 43], though it can also be set as a random parameter in order to avoid global synchronisation [47]. P_{inc} and P_{dec} are used to identify the amount by which D_p should be increased or decreased. Further, P_{inc} must be larger than P_{dec} in order to prevent underutilisation [43]. BLUE solves the global synchronisation problem by either choosing a $freeze\_time$ parameter randomly or by dropping packets randomly similar to RED's approach. The BLUE limitations are the same as those in DRED. Subsection 2.4.10.1 explains how BLUE and its variants treat the problem of misbehaving flows.

On losing of the packets or $(threshold < B_{length})$

```
if ((current_time – last_adjustment) > freeze_time)
{
        Dp = Dp + Pinc ;
        last_adjustment = current_time ;
}
```

When the buffer is empty (or $B_{length} = 0$)

```
if ((current_time – last_adjustment) > freeze_time)
{
        Dp = Dp – Pdec ;
        last_adjustment = current_time ;
}
```

Figure 2.20: The BLUE algorithm [39, 43].

2.4.10.1 Behaving and Misbehaving Flows in Networks

The TCP flows can be classified either as a behaving flow (responsive flow) or misbehaving flow (non responsive flow). The responsive flows react when congestion occurs by reducing their transmission window sizes to alleviate the congestion, thus, often a good performance is achieved. Non responsive flows usually do not change their transmission rates when congestion occurs, therefore, a poor network performance is achieved. Many researchers have introduced different approaches to solve the misbehaving

flows problems such as [37, 74, 90], where their ultimate goal is to identify and limit transmitting rates for the misbehaving flows since these flows normally do not react properly when congestion is present in the network [46, 90]. In this section, the BLUE-related algorithms is surveyed that deal with the misbehaving flows in a network.

2.4.10.2 Stochastic Fair BLUE (SFB) Algorithm

The Stochastic Fair BLUE (SFB) [38] is a BLUE-like algorithm that depends upon the $B.v$ bins, where B represents the number of bins arranged in each v level, and v is the number of levels, where every v level has an autonomous hash function. The purpose of the hash functions is to map a flow to a bin in a certain level (the level that the hash function is linked to). Furthermore, the hash function relies on the connection ID (source address, destination address, source port number, destination port number, protocol) in the mapping policy [38]. The SFB utilises BLUE's strategy in calculating D_p in which each bin in the SFB has a D_p that gets updated according to the bin queue size. Hence, whenever a packet arrives at the router buffer, it gets mapped to one bin in each level using the hash function. If the number of packets mapped to a particular bin is greater than a predetermined threshold (exceeds the bin occupancy threshold (B_{size})) then the D_p value for that bin will increase, while if the B_{size} becomes idle the D_p will decrease.

Further, D_p for the misbehaving flows in SFB [38] changes to "1" quickly, whereas, D_p for the behaving flows is calculated similar to [39, 43]. The SFB algorithm decides to

drop packets based on $D_{\min}$, which corresponds to the minimum D_p value for all the bins that a particular flow is mapped to. In cases where $D_{\min}$ is less than "1", the SFB identifies the flow as a behaving one and it drops the arriving packets within this $D_{\min}$. Otherwise, the SFB identifies the flow as a misbehaving one and reduces the transmission window sizes for these flows by cutting down the amount of network resources which they use. The pseudocode for the SFB algorithm and an example that distinguishes between the behaving and the misbehaving flows are demonstrated in Figures 2.21 and 2.22, respectively. The q_{size} parameter represents the number of packets mapped to a particular bin, both P_{inc} and P_{dec} parameters are similar to those used in BLUE [43].

1. Identification stage
   Arr [v levels] [B bins]: array of v . B bins;
   $Idle = 0$;
   $Busy = 1$;
2. Maps the arriving packet into a single bin in every level using the hash functions, as follows:
   Compute $hash_0, hash_1, hash_2, \ldots, hash_{v-1}$

$for(j = 0; j < v; j++)$

{

$if(Arr[j][hash_j].q_{size} == Idle)$

$Arr[j][hash_j].D_p = Arr[j][hash_j].D_p - P_{dec}$;

$if(Arr[j][hash_j].q_{size} > B_{size})$

{

$Arr[j][hash_j].D_p = Arr[j][hash_j].D_p + P_{inc}$;

Drops/Marks the arriving packet similar to the BLUE algorithm;

}

$D_{min} = \min imum(Arr[0][hash_0].D_{min}, Arr[1][hash_1].D_{min}, \ldots, Arr[v-1][hash_{v-1}].D_{min})$;

$if(D_{min} == Busy)$

Limit the transmission window size for the misbehaving flow;

$else$

Drops/Marks the arriving packet with D_{min} value;

}

Figure 2.21: The SFB algorithm [38, 43].

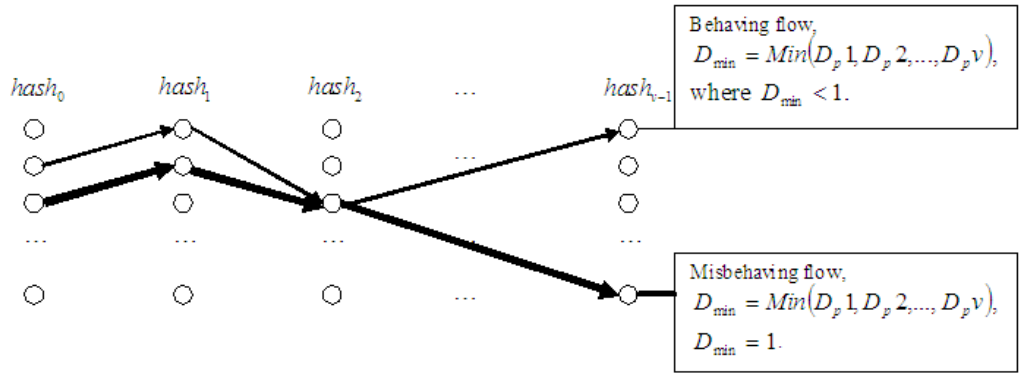

Figure 2.22: An example for distinguishing between the behaving flows and the misbehaving flows [38, 43].

2.4.10.3 Fuzzy BLUE (FB) Algorithm

The Fuzzy BLUE (FB) [122] is an AQM algorithm that extends BLUE and uses fuzzy logic [71, 85, 125]. It mainly employs a fuzzy BLUE controller (FBC) implemented at the router as a congestion detector in the network. The FBC depends on two input linguistic variables (buffer occupancy (BO), packet loss (PL) fraction) to generate one output linguistic variable, i.e. (D_p). Moreover, to simplify the computation of the membership functions in the fuzzy part of FB, the authors applied either trapezoidal [71] or triangular [85] membership functions on the two input linguistic variables and the one output linguistic variable. Figure 2.23 shows the fuzzy sets for both inputs and the output variables [122].

After the fuzzy sets are determined for the linguistic variables, the fuzzy rules of FB are then input by the domain experts. The FB fuzzy rules are shown in Figure 2.24, and Figures 2.25-2.27 represent the membership functions and their degrees for BO, PL and D_p, respectively. Moreover, FB employs two methods for designing the fuzzy rules, i.e. theory

method and the trial and error method [122]. In theory method, rules were designed to produce a satisfactory QoS for traffic loads, whereas in the trial and error method, rules were designed from experience and the extensive experiments of the domain expert. After the fuzzy rules are constructed, fuzzy inference (FI) is applied to obtain a crisp value for the output linguistic variable (D_p). This can be done using the FBC, which performs the following four steps [78, 85]:

1. Fuzzification: Maps each input variable to its corresponding membership function degree. In other words, identify the area where the input variable has valid values with reference to the membership functions. For instance, a low linguistic variable value for buffer occupancy (BO) varies from 0.0 to 80 membership function degrees (see Figure 2.25).
2. Rule evaluation: Evaluate the "if" part for each fuzzy rule using the input linguistic variable values specified in the first step. Thus, the prediction of the "then" part is made for each fuzzy rule.
3. Aggregating all "then" results (membership degrees) into a single fuzzy set: This step explains the aggregation technique that combines all the "then" results (membership degrees) of the output linguistic variable (D_p) into a single fuzzy set. Clearly, all the membership degrees which obtained from "then" parts in the linguistic rules are aggregated to give a single fuzzy set. Then this single fuzzy set will use as an input in the next step (defuzification).
4. Defuzification: Calculate a single crisp value for the output linguistic variable (D_p) using its membership function and the centre of gravity (COG) method [85]. In this

step, the defuzifier aggregates the output rules ("then" parts) into a single output rule and uses it as an input to the defuzification step, then it applies the COG method to obtain a single crisp value for the D_p.

Buffer occupancy (BO) = {low, middle, high}.
Packet loss (PL) = {small, medium, big}.
Packets dropping probability (D_p) = {zero, low, moderate, high}.

Figure 2.23: The input and output linguistic variables fuzzy sets in FB mechanism [123].

If BO is low and PL is small then D_p value is zero
If BO is low and PL is medium then D_p value is zero
If BO is low and PL is big then D_p value is zero
If BO is middle and PL is small then D_p value is zero
If BO is middle and PL is medium then D_p value is zero
If BO is middle and PL is big then D_p value is low
If BO is high and PL is small then D_p value is zero
If BO is high and PL is medium then D_p value is moderate
If BO is high and PL is big then D_p value is high

Figure 2.24: The FB rules [123].

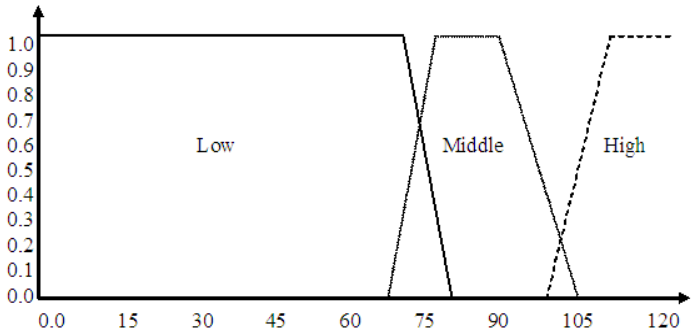

Figure 2.25: The membership function for the buffer occupancy (BO) linguistic variable [123].

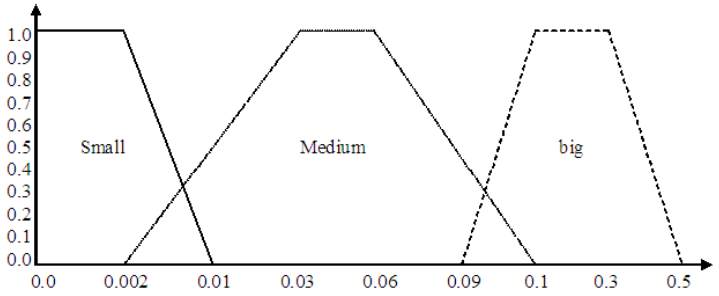

Figure 2.26: The membership function for the packet loss fraction (PL) linguistic variable [123].

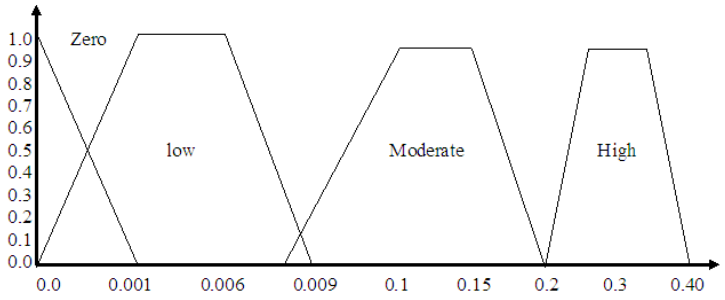

Figure 2.27: The membership function for the packet dropping probability (D_p) linguistic variable [123].

2.4.11 Generalised Random Early Evasion Network (GREEN) Algorithm

The Generalised Random Early Evasion Network (GREEN) [42, 121] is one of the AQM methods that provides a fair share of network resources among the TCP sources in the network. GREEN relies on the status of the sources to decide whether to drop the arriving packets and adopts the bandwidth estimated equation from [82] (shown in equation (2.19))

to compute the bandwidth for each TCP source in the network. The computed bandwidth for a source shown in equation (2.19) depends on the sources D_p, and it's RTT. In addition, $Seg_{\max}$ and f parameters shown in the equation below represent the maximum segment size and the fix parameter for a TCP source, respectively. f relies on both the acknowledgment policy (delay acknowledgement) and the dropping policy (randomly, periodically).

$$Band = \frac{Seg_{\max} \times f}{(RTT) \times \sqrt{D_p}} \quad \text{..... (2.19)}$$

To clarify how the GREEN algorithm works [42], consider a network, where X is assumed to be the number of active TCP flows in the network and K is the link buffer capacity shared by X. [42] has estimated the bandwidth shared by the TCP flows in the network using $\left(\frac{K}{X}\right)$, where each flow in the network gets the same amount of $\left(\frac{K}{X}\right)$ bandwidth. Now, by replacing $Band$ in equation (2.19) with $\left(\frac{K}{X}\right)$, the following is obtained:

$$\frac{K}{X} = \frac{Seg_{\max} \times f}{(RTT) \times \sqrt{D_p}} \quad \text{..... (2.20)}$$

Then from equation (2.20), the D_p is obtained as below,

$$D_p = \left(\frac{X \times Seg_{\max} \times f}{(RTT) \times K} \right)^2 \quad \text{..... (2.21)}$$

It can be observed from equation (2.21) that the D_p for each TCP flow relies on both the RTT and the X values. Hence, when X is large and there is short RTT for the TCP flows,

then the network becomes heavily congested since every flow increases its transmission window size quickly.

Furthermore, GREEN solves the bias versus bursty traffic flows problem using the RTT inverse (see equation (2.21)), where it removes the discrimination among the TCP flows (flows with long or short RTT). Moreover, to compute D_p, GREEN estimates X, $Seg_{\max}$ and RTT. Firstly, X is estimated by counting every TCP flow that has at least a single packet at the router buffer. Secondly, $Seg_{\max}$ is estimated by monitoring the size for every arrival packet at the router buffer. Thirdly, two methods for estimating the RTT value have been used in [42], where in the first method GREEN uses the IP fields to hold the RTT which has been added by the TCP source, and then this IP gets sent via the router to the destination. The second method uses the IDMapsService [54], in which the GREEN router employs an IDMaps to compute the RTT value. The GREEN router is similar to the IDMaps tracer, in which it computes the RTT estimates from a database. This database is used to retrieve the requested RTT with respect to the IP addresses for source and destination.

In an experimental study [42], GREEN was compared with the SFB [38] and the DT [20, 22] methods with reference to different performance metrics such the fair share in the available network resources among the TCP sources in the network, link utilisation, ql, and packet loss rate. GREEN has used the Jain's fairness index [25], shown in equation (2.22), to obtain the fairness degree for the TCP sources in the network, where th_i represents the throughput for all the TCP sources, and x is the number of active sources in the network. The fairness degree calculated in equation (2.22) falls between 0 and 1 in

which “1” indicates that all the TCP sources take the same share from the available network resources.

$$Fairness(th_1, th_2, ..., th_x) = \frac{\left(\sum_{i=1}^{x} th_i\right)^2}{x\sum_{i=1}^{x} th_i^{\,2}} \quad \text{..... (2.22)}$$

In the experimental study [42], GREEN achieved the highest fairness degree among the algorithms considered and the DT method outperformed the SFB according to fairness metric results. Further, the DT method has the highest link utilisation value since it does not drop packets as long as the router buffer does not overflow and the SFB algorithm outperformed GREEN in terms of link utilisation metric results. Also, the ql for the DT is highest since the aql of the DT increases when the number of TCP sources increases [42], and consequently the DT method has a poor performance regarding to ql metric. Both the GREEN and SFB algorithms have the similar ql results. Finally, the GREEN, SFB and DT methods have almost the same packet loss rate.

2.4.12 Random Exponential Marking (REM) Algorithm

The Random Exponential Marking (REM) was proposed in [9, 73] to obtain high utilisation and negligible packet loss and queueing delay. REM has two main properties, the comparison of transmission rate and queue length property and the total price property. The aim for the first property is to stabilise both the transmission rates (R) of the network

sources at the link capacity (L) and the queue length (ql) at a certain level (target queue length (T_{ql})), whereas the price property is used to determine the D_P, and thus REM uses it as a congestion metric. This price parameter depends on the rate mismatch and the queue mismatch, where the rate mismatch corresponds to the difference between the sources transmission rates and the link capacity, and the queue mismatch equals the difference between the queue length and the target queue length [9, 73]. The price value increases when the total two mismatches (W_{Total}) is positive, otherwise, it decreases as shown in equation (2.23).

$$W_{Total} = \phi\left[\partial\left(ql - T_{ql}\right) + R - L\right] \quad \text{(2.23)}$$

where ϕ and ∂ are fixed parameters greater than 0.

According to equation (2.23), W_{Total} is positive when at least one of the two mismatches is positive, otherwise, it is negative. When the number of network sources increases, this increases R and ql values, and they could become above L and T_{ql}, respectively. Thus, their weighted total (W_{Total}) would become positive. Since the price value is used to determine D_P, if the price value increases, congestion occurs in the network and the D_P increases exponentially, otherwise it decreases. Thus, when the price values increases, this makes the network sources decrease their transmission rates in order to alleviate the congestion incident at the network. On the other hand, the price value becomes stable if the queue length is in equilibrium (ql has the same T_{ql} value) and R is equal to L. In this case D_P is stabilised and W_{Total} becomes zero.

Equation (2.24) shows how to adjust the price value at each router buffer (B) in particular time interval (t).

$$\Pr ice_B(t+1) = \left[\Pr ice_B(t) + \phi\left[\partial_B\left(ql_B(t) - T_{qlB}\right) + R_B(t) - L_B(t)\right]\right]^+ \text{..... (2.24)}$$

where $\Pr ice_B(t)$ represents the price value at the B router buffer at a t time interval, $ql_B(t)$ represents the queue length at the B router buffer at the t time interval, T_{qlB} represents the target queue length for the B router, $R_B(t)$ represents the input transmission rate at the B router buffer at the t time interval, and $L_B(t)$ represents the amount of bandwidth allocated at the B router buffer at the t time interval. Moreover, ϕ and ∂_B are fixed parameters greater than 0, where ∂_B can be set by each router buffer independently and used to make a tradeoff between utilisation and queueing delay [73]. ϕ is used to control the REM responses to different network situations.

Furthermore, $[x]^+ = \max imum\{x,0\}$, $\left(ql_B(t) - T_{qlB}\right)$represents the queue mismatch, $R_B(t) - L_B(t)$ represents the rate mismatch, $\partial_B\left(ql_B(t) - T_{qlB}\right) + R_B(t) - L_B(t)$represents weighted total(W_{Total}). Hence, if $\partial_B\left(ql_B(t) - T_{qlB}\right) + R_B(t) - L_B(t)$ is positive, then, $\Pr ice_B(t+1)$ increases because of the congestion, and D_P increases exponentially. Whereas if $\partial_B\left(ql_B(t) - T_{qlB}\right) + R_B(t) - L_B(t)$ is negative, then, $\Pr ice_B(t+1)$ and D_P are decreased, and the network sources increase their transmission rates. Finally, if $\partial_B\left(ql_B(t) - T_{qlB}\right) + R_B(t) - L_B(t)$ is equal to zero, the queue length becomes stable, and therefore the D_P and $\Pr ice_B(t+1)$ also become stabilised, i.e. $\Pr ice_B(t+1) = \Pr ice_B(t)$.

The total price for all routers' buffers is used by REM as a congestion metric. This happens when the source sends a particular packet through n routers' buffers along the path to its destination, then each router buffer calculates its price value, and the total prices for the routers' buffers can help in calculating the D_P along the path from source to its destination. When some routers' buffers at the path between source and destination are congested, their price values increases, and this explains the exponential growth in the D_P. In order to alleviate the congestion, the routers inform the sources about the congestion, and the sources adjust their transmission rates. Assume that B is the number of router buffers that the packet has passed through, i.e. $B = 1,2,3,...,n$. Assume also that these routers have $\Pr ice_B(t)$ prices at t time interval, then the packet dropping probability (D_B) for each router buffer is shown in equation (2.25) below,

$$D_B(t) = 1 - \delta^{-\Pr ice_B(t)} \quad \text{..... (2.25)}$$

where δ is a fixed parameter greater than 1. The computation of $D_p(t)$ is displayed in equation (2.26).

$$D_p(t) = 1 - \delta^{-\sum B \Pr ice_B(t)} = 1 - \prod_{B=1}^{n} (1 - D_B(t)) \quad \text{..... (2.26)}$$

The $D_p(t)$ value relies on total prices, i.e. $\left(\sum B.price_B(t)\right)$. If $\left(\sum B.price_B(t)\right)$ is large, then the $D_p(t)$ value is large as well. Whereas, if $\left(\sum B.price_B(t)\right)$ is small, then the $D_p(t)$ value is small as well. The $D_p(t)$ value is defined in equation (2.27) below,

$$D_p(t) \cong (\log_e \delta) \sum_B \Pr ice_B(t) \quad \text{..... (2.27)}$$

2.5 Chapter Summary

In this chapter, different AQM congestion control techniques (shown in Table 2.2), including DT, Early Random Drop, Decbit, RED, REM, GRED, ARED, DRED, SRED, BLUE, SFB, fuzzy BLUE, FEM, GREEN and adjustment RED, are surveyed. Particularly, their congestion metrics, methodologies used to detect and manage the congestion, strengths and weaknesses are discussed.

This survey was indicated that all the AQM techniques are mainly aimed to control the congested networks in an early stage, which means before the router buffers overflow. It is inferred beside the main aim, there are other aims for every AQM technique, for instance, ARED and DRED were presented to stabilise their congestion measures (average queue length (aql for ARED) and queue length (ql for DRED)) at small values in order to prevent building up their router buffers. SRED and SFB have a common goal, which is identifying the misbehaving flows and limit the transmission rates for these flows. Furthermore, GREEN is aimed to distribute a fair share of network resources among sources. REM was introduced to achieve a high utilisation and negligible delay and packet loss rate.

Many AQM techniques were proposed to solve some of RED's problems, such as ARED and DRED stabilise their congestion measures at values smaller than that of RED, therefore they avoid overflowing their router buffers better than RED. In addition, ARED was designed to decrease the RED reliance on it's parameter values with aim of obtaining a

satisfactory performance. So the ARED was used an adaptive $D_{\max}$ rather than the fixed one.

Until now the RED's parameters have no optimal values to set. Thus, this thesis proposes some AQM techniques to reduce a large dependency of RED on its parameters. These techniques are Dynamic RED (REDD) and fuzzy logic controller algorithms (REDD1, REDD2, REDD3).

It can be inferred also no AQM technique was proposed to cover all the congestion control issues in computer networks, i.e. internet.

This survey has not focused on designing AQM techniques based on analytical models since relatively few research works were concentrated on analysing the performance of AQM techniques using queueing analysis approaches. This thesis proposes some discrete-time queue analytical models based around the mechanisms used in RED, GRED, DRED and BLUE to analyse and compare their performance with the established AQM techniques in detecting congestion in wired networks. Using the aql by some AQM techniques, like RED leads to the following problem: At a short interval, the arrival rate momentarily increases; this may cause the RED router buffer to fill up and overflow. The aql may be less than the $\min threshold$ and takes a time to build up. Thus, no packets can be dropped although the RED router buffer is overflowing. To overcome this problem, the instantaneous ql is used instead of aql in the proposed analytical models, so that when the ql passes the $\min threshold$ this causes probabilistic dropping to occur before the buffer has chance to overflow.

The TCP congestion control methods are effectively windows based mechanisms in that they rely on adjusting the congestion window size to limit the number of arriving packets in a specific length of time. The AQM methods are mostly rate based in the sense that probabilistically dropping packets before the buffer is full reduces the effective arrival rate.

In networks such as the internet, AQM methods like RED are intended to operate with TCP. However, in this thesis we consider the AQM methods independently in order to evaluate their effectiveness as buffer management methods without superimposing the additional traffic profile of a transport protocol such as TCP on them.

Table 2.2: The congestion metric and packet dropping policies for the AQM methods considered.

AQM Method	Congestion Metric	Packet Dropping (or marking) Policy
RED	Average Queue Length (aql).	Randomisation, when aql is between $\min th$ & $\max th$.
GRED	Average Queue Length (aql).	Randomisation, when aql is between $\min th$ & $2\max th$.
DRED	Queue Length (ql).	Randomisation, when current ql reaches th.
ARED	Average Queue Length (aql).	Randomisation, when aql is between $\min th$ & $\max th$.
SRED	The estimated number of active flows ($P(m)^{-1}$), and the current queue length (ql).	SRED drops the arrival packets relies on the $P(m)^{-1}$ and current ql .
REM	price.	If calculated weighted total (W_{Total}) is positive, the D_p increases exponentially. Otherwise, the D_p will decrease exponentially.
FEM	Fuzzy Inference Process (FIP).	FEM calculates D_p using FIP, which FIP depends on two input linguistic variables (current buffer length (CBL) and changing rate in the buffer (CRB)).
BLUE	Queue Length (ql), Packet Loss & Link Idle.	Randomisation; the D_P increases when the current ql exceeds th. Whereas, the D_P decreases when the current ql becomes idle.
SFB	Queue Length (ql), Packet Loss & Link Idle.	Randomisation; the D_P increases when the current ql exceeds a particular th i.e. the bin occupancy (B_{size}). Whereas, the D_P decreases when the current ql becomes zero.
FB	Fuzzy BLUE Controller (FBC).	FB evaluates D_P utilising FBC, and the FBC in his role exploits two input linguistic variables that denoted by buffer occupancy (BO) and packet loss (PL) to carry out the D_P result.
GREEN	The number of active connections (X) and the round trip time (RTT) for packets.	GREEN uses X, RTT and the parameters of f (fixed parameter), $Seg_{\max}$ (maximum segment size) and K (link buffer capacity) to compute D_p.
Adjustment RED	Average Queue Length (aql) with combination of input and output queues (IQs and OQs), respectively.	Randomisation; when aql is between $\min th$ & $\max th$.
Decbit	Binary indication bit and Average Queue Length (aql).	If the aql is larger than 1, Decbit marks every arriving packet. Otherwise, no packets are marked.
Early Random Drop	Queue Length (ql).	Dropping the arrival packets with a constant probability.
DT	Queue Length (ql).	Periodically.