International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)

http://ijcjournal.org/

Burst Loss Reduction Using Fuzzy-Based Adaptive Burst Length Assembly Technique for Optical Burst Switched Networks

Abubakar Muhammad Umaru*

Yusuf Maitama Sule University, Kano. Nigeria
Email: amumaru@nwu.edu.ng
Email: amumaru@yahoo.com

Abstract

The optical burst switching (OBS) paradigm is perceived as an intermediate switching technology prior to the realization of an all-optical network. Burst assembly is the first process that takes place at the edge of an OBS network. It is crucial to the performance of an OBS network because it greatly influences loss and delay on such networks. Burst assembly is an important process while burst loss ratio (BLR) and delay are important issues in OBS. In this paper, an intelligent burst assembly algorithm called a Fuzzy-based Adaptive Length Burst Assembly (FALBA) algorithm that is based on fuzzy logic and tuning of fuzzy logic parameters is proposed for OBS network. FALBA was evaluated against itself and the fuzzy adaptive threshold (FAT) burst assembly algorithm using 12 configurations via simulation. The 12 configurations were derived from three rule sets (denoted 0,1,2), two defuzzification techniques (Centroid [C]and Largest of Maximum[L]) and two aggregation methods (Max[M] and Sum[S]) of fuzzy logic. Simulation results have shown that $FALBA_{0LM}$ has the best BLR performance when compared to its other configurations and the FAT. However, with respect to delay, FAT only outperforms all configurations of FALBA at low loads (0.0-0.4) but the performance of FAT also decreases as the load (0.4-1.0) increases. Therefore, at high loads (0.4-1.0) $FALBA_{2CS}$ has the best delay performance. Our results deduce that $FALBA_{0LM}$ can be used for loss-sensitive applications while $FALBA_{2CS}$ can be used for delay sensitive applications.

Keywords: Burst Assembly; Burstification; Edge Node; Fuzzy Logic Control; Optical Burst Switching; Delay.

---

* Corresponding author.

1. Introduction

The increasing demand for high bandwidth by bandwidth-greedy applications requires a switching technology that is capable of meeting such demand. Optical Circuit Switching (OCS), Optical Burst Switching (OBS) and Optical Packet Switching (OPS) are three main switching paradigms considered to address these demands. Of the three paradigms, OBS [1] provides huge bandwidth [2] and better link utilization through statistical multiplexing than OCS. Furthermore unlike OPS which requires optical memory which is still technologically immature [3] OBS works without such limitation. These merits of OBS over OCS and OPS makes it an active area of research and a candidate for the next generation optical transport backbone. An OBS network is comprised of edge (ingress and egress) and core nodes which are interconnected by wavelength division multiplexing (WDM) optical links. The edge nodes provides functions for burst assembly/disassembly, offset-time computation, signaling, and routing and wavelength assignment while contention resolution and scheduling functions are performed at the core nodes [4]. Authors in [5-8] have conducted a comprehensive review of OBS network in which they identified the major issues as follows: burst assembling, contention resolution, quality of service (QoS) provisioning, routing and wavelength assignment and core node scheduling. Among these functions, burst assembly is a crucial one and this is due to its effect on the congestion and contention levels it has on the network which consequently affects network performance. Works in [9,8] have classified burst assembly in their studies which covers a small set of burst assembly algorithms. Core node scheduling is vital to the performance of OBS network and authors in [10] have reviewed and classified them into proactive and reactive scheduling algorithms. Furthermore, Reference [11] compared a selected set core node scheduling algorithms with focus on QoS. Mechanisms providing QoS differentiation with respect to their complexities and efficiencies have been reviewed and classified in [12]. Authors in [13] have reviewed various routing techniques used to proactively prevent contention in OBS. OBS network architecture comprises of edge and core nodes interconnected by multiple-channel WDM links. Fig. 1 shows model architecture of an OBS network. In OBS, end users' data are aggregated and transported as jumbo packets known as bursts. Therefore, burst assembly is the process of aggregating end users data into a burst based on predefined requirements and procedure. Using Just-Enough-Time (JET) [14] signaling technique, a newly generated burst is delayed at the edge node until after a period of time known as the offset-time expires before it is transmitted into the network. Following the burst generation, a corresponding control packet (CP) is also generated and transmitted into the network ahead of its burst. This CP is responsible for reserving resources for its burst at the various core nodes. The basic OBS architecture supports a one-to-one mapping between the CP and its burst. The CP contains routing, burst arrival time, burst duration and QoS information [3]. The core node uses the CP information to make reservation or resolve contention. Upon a successful reservation or resolved contention, the core node updates the CP information and forwards it to downstream node/s otherwise the CP and its corresponding burst will be dropped. This process is repeated at every core node until the CP gets to its destination. Burst disassembly takes place upon arrival of the burst at the egress node which forwards the IP packets to the end users. Fig. 1 shows a typical OBS network architecture. An important issue in the burst assembly process is how to create bursts of optimal sizes such that burst contention can be minimized and therefore, burst loss is reduced. This issue has to deal with the burst size creation parameter. Additionally, burst assembly algorithm if not properly designed cause high delay at low traffic conditions and such characteristic makes them unsuitable for

delay-sensitive applications. Therefore, in order to alleviate the aforementioned problems, an intelligent burst assembly algorithm is proposed. The proposed algorithm uses fuzzy logic to make the burst length threshold value adaptive subject to the current value of the burst length threshold, the incoming traffic load of the assembler and the bandwidth of the outgoing channel. The main goal of the proposed algorithm is to improve burst loss and delay performances in an OBS network. The paper is organized as follows: Section I provides a general introduction to OBS, its issues, its architecture and mode of operation. Section II describes related works and the working process of the threshold based burst assembly algorithm. Section III describes the design and implementation procedure of the proposed burst assembly algorithm using fuzzy logic controller. It also describes the testing and tuning configurations and simulation parameters used. Section IV presents the simulation results and their discussions. Finally, Section V concludes the paper.

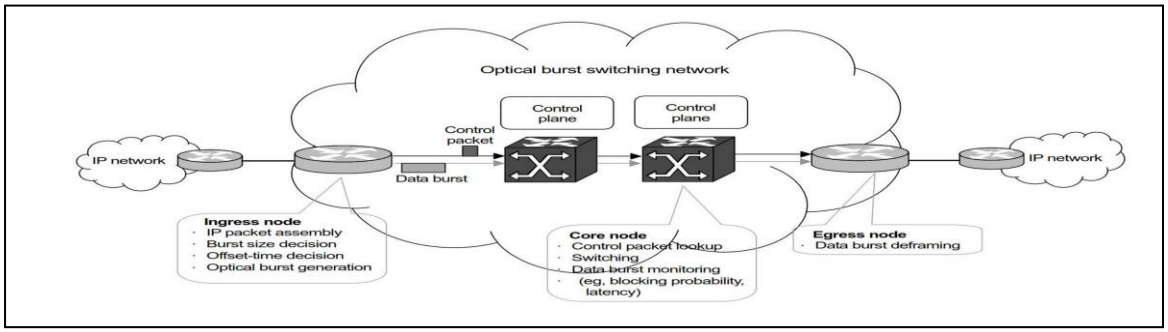

Figure 1: An OBS Network Architecture [3].

2. Review of Related Works

Burst assembly algorithms have been categorized into four groups: time-based, threshold-based, hybrid-based and adaptive based [15, 16]. The timer-based burst assembly algorithm [17] is the pioneer burst assembly algorithm in OBS. As the name implies, it uses a timer to generate a burst whenever the timer threshold expires. Thus, the timer assembly algorithm produces variable-sized bursts at periodic intervals [18]. The simplicity of the timer-based assembly technique is its strength. However, at high loads, the timer algorithm generates large bursts that increase the burst blocking probability at the core nodes. Also, burst of small sizes are generated at low loads by the algorithm. Furthermore, the timer parameter of the algorithm does not adapt with the dynamic nature of incoming traffic [19], thus even when the burst size is large enough for transmission it cannot generate a burst until its timer expires. Hence, resulting into poor transmission efficiency [20]. The second group of burst assembly algorithms is the threshold-based burst assembly algorithm which are also known as length, size or volume burst assembly algorithm. The threshold-based burst assembly algorithm [21] is similar to the timer-based assembly algorithm except that it uses the burst size as a parameter for burst generation instead of a timer. Thus, threshold-based assembly algorithm generates fixed-size bursts at random intervals of time. Even though, the threshold-based assembly algorithm is simple and it has a low implementation complexity, it still has limitations. At low loads, packets experience high delay while at high loads, fixed-size bursts are rapidly generated and transmitted into the network. The high number of bursts produced at high loads usually lead burst loss resulting from burst contentions. Similar to the timer algorithm, the threshold-based assembly parameter does not adapt with the incoming traffic thereby making it unable to fully utilise the available resources. The

length or threshold-based burst assembly process in OBS starts upon the arrival of the first packet to an edge node. Upon the arrival of the first packet to the assembler, a burst length counter denoted as *b_length* that is associated with the assembler is initialized to zero as shown in Equation 1. After the initialization, the burst length counter is incremented by the size of the packet which is denoted by *packetSize*. Subsequent packets arriving at the assembler will continue to increment the burst length counter as shown in Equation 2. However, before the burst length counter is updated, a check is made against a pre-defined burst length threshold which is denoted by *pb_length* to determine whether the maximum burst size is reached as shown in Equation **Error! Reference source not found.**. If the burst length counter is equal to or greater than the preset burst length threshold, the burst length counter is reset to zero as in Equation 1 and the packets in the assembler queue are assembled into a burst and forwarded to the burst scheduler for onward transmission to its destination. Otherwise, the burst length counter is updated according to Equation 2 and the process is repeated for as long as traffic keeps arriving at the assembler.

$$b_{length}=0 \quad (1)$$

$$b_{length} = b_{length} + packetSize \quad (2)$$

$$b_{length} \geq pb\_length \quad (3)$$

As it can be observed from the above description of the length burst assembly technique, the burst length threshold (i.e *pb_length*) remains constant throughout the burst assembly process. This constant burst length threshold parameters makes the length burst assembly algorithm to lack the flexibility to adapt to dynamic traffic conditions. Additionally, this algorithm causes high delay at low traffic conditions and such characteristic makes it unsuitable for delay-sensitive applications. In order to provide service differentiation through burst assembly, Vokkarane and his colleagues [22] proposed the threshold-based composite burst assembly algorithm. This algorithm aggregates different classes of packets that are destined for the same egress node into the same burst. Furthermore, the packets are placed inside the burst in decreasing order of priority starting from the head of the burst. Thus, whenever contention occurs, burst segmentation is used to drop the tail of the scheduled burst. Even though this technique provides some degree of service differentiation, it only improves the loss performance of high priority packets. Furthermore, the assembly threshold parameter is also not adaptive. Moreover, burst segmentation must be implemented at the core nodes for the technique to be functional. Likewise, the threshold-based mixed-assembly technique proposed by Zhang and his colleagues [23] is similar to the work in Vokkarane and his colleagues [22]. The threshold-based mixed-assembly technique lacks the mechanism to adjust the assembly threshold parameter. The limitations of the Timer and Threshold-based algorithms paved way for the hybrid burst assembly algorithms. The Max-Time-Min-Max-Length [24] and the Min-burst Length-Max-Assembly-Period (MBMAP) [25] burst assembly algorithms were proposed to address the limitations of the timer and threshold-based assembly algorithms. In both algorithms, a burst is generated when either the maximum time is reached or when the minimum burst length is attained. Hence, they are considered as hybrid algorithms. These hybrid algorithms have addressed the problems of long packet delay during low loads and large bursts under high loads. Furthermore, the implementation of these algorithms are relatively simple. Even though they solve the problems of the basic algorithms they do not consider the dynamic nature of the incoming traffic in order to adjust the timer or size thresholds of the

algorithms [26]. Also, under such condition, the hybrid burst assembly algorithms will behave exactly like the timer and size-threshold assembly algorithms.The last group assembly algorithms belong to the adaptive class of burst assembly algorithms. Algorithms in this class have their burst creation parameter (timer, burst-size or both) continuously adjusted subject to real time traffic [27] or network state information (such as loss, delay congestion level etc). Adaptive burst assembly algorithms are intended to improve the performance of the OBS network. Researchers have used different approaches such as prediction, soft computing and feedback mechanisms to adjust the burst creation parameters of assemblers.In order to minimize the burst transmission delay caused during burst assembly and transmission, Authors such as [28,29,30-33,26] have proposed several prediction-based burst assembly algorithms. Such algorithms predict either the incoming traffic, burst-size or timer thresholds. The predicted parameter is used to generate a burst. The purpose of predicting the burst length is to include the predicted burst size into the burst control packet (BCP) for early transmission so that resource reservation at downstream nodes can start early even before the burst is generated. Thus, by sending the BCP ahead of burst completion the delay can be reduced. Reference [28] proposed a set of burst assembly schemes together with a fast reservation protocol (FRP). The burst assembly schemes work towards reducing the delay experienced by packets while the FR protocol works towards reducing the burst end-to-end delay. They used linear prediction filters to predict the traffic offered to the edge node during a time interval and the result is used as a criterion to assemble a burst. Additionally, the FRP uses another prediction filter to calculate the expected burst length and assembly duration in order to send the BCP to make an early reservation. In this case, the edge node does not need to wait for the burst to be assembled before it generates and sends the BCP. Hence, this approach reduces burst assembly and reservation times and improves packet end-to-end delay. However, the predicted values may be inaccurate and therefore may lead to poor resource utilization. The mixed-threshold burst assembly (MTBA) algorithm [29] uses traffic prediction to calculate the expected burst length. The predicted length is then added to the BCP which is immediately transmitted into the network in order to make early reservation before the burst is generated. This approach improves the end-to-end delay performance of high priority bursts since it does not wait for the burst generation process to complete before sending the BCP. However, poor resource utilization may occur if the generated burst size is less than the predicted length. Similarly, References [30, 31, 33,26] employed different prediction mechanisms to further minimise delay but they still suffer from the same problems of the aforementioned prediction based algorithms.In feedback-based bustifiers, network state information such as network offered load, burst loss, delay, congestion level or other network performance measurements are collected from the network and they are used to adjust the burst assembly parameters. They are mainly used to control the congestion level on the network in order to minimize burst contention. References [34, 35] proposed an adaptive hybrid burst assembly algorithm that adjusts the timer and size threshold values of the assembler using the congestion information of links incident to the ingress node. Although the algorithms are adaptive, additional delay is incurred due to the large burst sizes that are generated at high loads. In addition, the hard limits set on the possible threshold values makes it inflexible, therefore, limiting the possible range of threshold values it can support and hence affecting its performance.

Reference [36] proposed the Intelligent segment optical burst switching (ISOBS) assembly algorithm that aggregates IP packets into small fixed-size segments. A group of segments are then considered as a burst. The burst size and the number of packets inside the segments change according to the input traffic intensity. And

since, the burst is divided into several segments of equal sizes, there is a short duration between the transmission of successive segments of the same burst. In the event of contention, only the contending segments known as the contenting region are affected. A major advantage of ISOBS is that it does not require the conventional burst segmentation mechanism at the core node. However, ISOBS suffers from the following: firstly, poor resource utilization may occur due to the additional void created between segments. Secondly, inaccurate time synchronization between segments may lead to further loss. Thirdly, it requires a specialized core node scheduling algorithm that is capable of identifying and scheduling related segments.

Reference [37] proposed the adaptive classified cloning and aggregation scheme (ACCS) for high priority traffic. ACCS uses network loss-rate information to adaptively adjust the hybrid burst assembly thresholds. ACCS aims to provide better performance in terms of loss and end-to-end delay for high priority traffic. However, at high loads, ACCS cannot handle the input traffic due to the limitation imposed by the bandwidth at the outgoing link of the edge node, and for this reason, low priority packets are dropped. Reference [19] proposed the data-length time-lag product (DTP) burst assembly algorithm which uses the real-time traffic information together with the burst injection rate to adjust the burst size threshold. Furthermore, DTP uses the result of the product of the assembly time and assembled burst length as the new threshold for assembling the next burst.

DTP aims to provide guaranteed burst assembly delay while at the same time lowering the blocking probability by generating bursts of average sizes. A major limitation of the DTP algorithm is that it executes after every burst is assembled, hence there is an overhead on the continuous execution of the algorithm for every burst that is generated. The link loss-rate burst assembly (LLRBA) and path loss-rate burst assembly (PLRBA) [38] algorithms are proposed to address the issue of contention in OBS network. LLRBA and PLRBA are adaptive by design and they respectively collect link and path loss rate information in order to determine the congestion level on the network before generating a burst. They are mainly used to control the congestion level on a network. Even though, both LLRBA and PLRBA minimize the contention on the network, they increase delay due to the latency incurred when collecting network state information. The Intelligent-based Burst Assembly Techniques are another class of burst assembly algorithms proposed to address most of the aforementioned problems. Burst assemblers in this category use soft computing techniques to intelligently adjust the threshold value/s of the burst assembler. By intelligence, it means that the burst assembly algorithms have the in-built capability to learn, understand and make informed decisions based on the information gathered from their environment.

The fuzzy adaptive threshold algorithm (FAT) [39] is based on the length-threshold assembly algorithm and fuzzy logic technique. FAT uses fuzzy logic together with the incoming traffic to dynamically adjust the length-threshold of the assembler.

FAT suffers from high frequency of execution because it is run every time a burst is generated. Reference [40] proposed a burst assembly algorithm using control theory techniques to measure burst loss ratio performance on an OBS network. Such approach requires complex mathematical principles in order to analyse and design such controllers. Authors in [41-45] have proposed a family of Fuzzy-based adaptive burst assembly algorithms to

address the problem of burst loss ratio and delay.

3. Fuzzy-based Adaptive Length Burst Assembly Algorithm (FALBA) Design

This Section describes the detailed design and implementation phases of the fuzzy-based adaptive length burst assembly algorithm whose main goal is to improve burst loss and delay performance in an OBS network.

In conventional threshold-based burst assembly algorithm, only a single parameter is considered for burst creation which is the burst length or threshold (*pb_length*). However, in the proposed method, the threshold-based burst assembly process is modelled as a control problem in which the burst length threshold parameter needs to be adjusted subject to incoming traffic and the bandwidth of the outgoing channel. As such, the proposed technique is modelled as a Multiple-Input-Single-Output (MISO) fuzzy logic controller. The controller accepts two inputs and then produces a single output as shown in Fig. 2. At this stage, two inputs and one output control variables have been identified.

The two input control variables are the burst length threshold and the incoming traffic (also known as offered load at the burst assembler) and they are denoted by *Length* and *Load* respectively. The output of the fuzzy logic controller is the new value of the burst length threshold (*newLength*) which will be used for the next cycle of the burst assembly process.

The assembler offered load which is denoted by *Load*, is the total amount of traffic received by the assembler starting from the moment the assembly process started. *BW* is the bandwidth of the outgoing channel in bits. Equation 4 shows how the offered load (*Load*) is calculated prior to passing it to the controller.

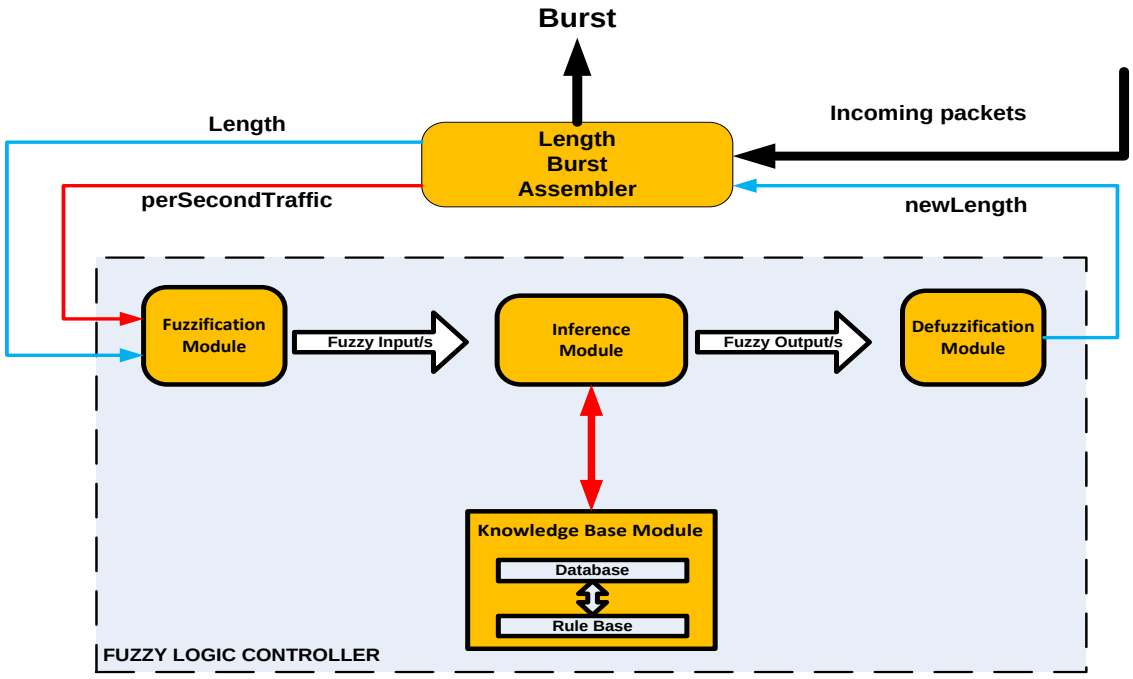

Figure 2: Block Diagram of the proposed Fuzzy logic controller

The proposed design assumes the following:

i. There is a dedicated burst assembler for every destination
ii. Functions such as burst control packet generation and burst scheduling takes place at their appropriate times
iii. All links have the same number of wavelengths/ channels
iv. All wavelengths have the same bandwidth capacity

$$\text{Load}=\begin{cases} 0, & \text{Load}>\text{BW} \\ \text{Load}+\text{packetSize}, & \text{Load}\leq\text{BW} \end{cases} \quad (4)$$

The main objective of this design is to minimizes burst loss and end-to-end delay by using the built-in intelligence of fuzzy logic to adjust the burst length threshold based on the incoming traffic and the bandwidth of the outgoing channel.

The incoming traffic load is intelligently mapped to the channel bandwidth such that by using the current value of the burst length threshold together with the mapped traffic load, a new burst length threshold value is produced using fuzzy logic reasoning.

Fuzzy parameters settings such as fuzzy rules, membership functions, aggregation method and deffuzification technique support the algorithm in achieving this objective. Fig. 3 shows the proposed fuzzy-based adaptive length burst assembly algorithm while the definition of variables and functions are given below:

i. ***i:*** Burst assembler index
ii. ***packetQueue (i):*** Data accumulated for the i-th burst queue (bytes).
iii. $\boldsymbol{b_{length}}$***:*** Burst length counter.
iv. ***packetSize:*** the size of packet in bytes.
v. ***Length:*** Length threshold used by the FLC (Fuzzy Logic Controller)in bytes.
vi. ***l:*** Initial length threshold in bytes.
vii. ***Load:*** Accumulated traffic at the assembler to be used by the FCL.
viii. ***newLength:*** New burst length threshold produced after executing the FCL.
ix. ***perSecondTraffic:*** Load counter in bits (to be used by *Load* variable).
x. ***BW:*** Channel bandwidth in bits.
xi. ***AssembleBurst(packetQueue(i)):*** Assembles the packets in packetQueue(i) into a burst.
xii. ***FuzzyEngine(Length,Load):*** This function execute the FLC using the Length and Load variables as inputs. The function returns a single value that serves as the output of the FLC. The output that is produced is then used as the new Length threshold for the next cycle of the burst assembly process.

The following subsections describe the fuzzification, knowledge-base, fuzzy inference engine, defuzzification, and testing, tuning and simulation parameters for the fuzzy-based adaptive burst length burst assembly algorithm.

Begin FALBA

$//\boldsymbol{Initialization\ Phase}$

$Length = l;\ l\ is\ the\ predefined\ burst\ length$

$Load = 0;$

$newLength = 0;$

$perSecondTraffic = 0;$

$BW = x\ Gbps;$

$b_{length} = 0;$

$//\boldsymbol{Execution\ Phase}$

1: $\boldsymbol{while}\ (Is\_Traffic\_Still\_Arriving?)\ \boldsymbol{do}$

2: $perSecondTraffic = perSecondTraffic + (packetSize * 8);$

3: $\boldsymbol{if}\ (perSecondTraffic \geq BW)\ \boldsymbol{then}$

4: $Load = perSecondTraffic;$

5: $perSecondTraffic = 0;$

6: $\boldsymbol{end\ if}$

7: $Load = perSecondTraffic;$

8: $\boldsymbol{if}\ b_{length} \geq Length\ \boldsymbol{then}$

9: $AssembleBurst(packetQueue(i));$

10: $newLength = FuzzyEngine(Length, Load);$

11: $Length = newLength;$

12: $b_{length} = 0;$

13: $\boldsymbol{else}$

14: $b_{length} = b_{length} + packetSize;$

15: $\boldsymbol{end\ if}$

Figure 3: Fuzzy-based Adaptive Length Burst Assembly Algorithm

3.1. Fuzzification

In fuzzy control systems, crisp control variables need to be converted into their equivalent fuzzy variables in order to control the system under study. Therefore, in this section, the inputs and outputs variables of the fuzzy logic controller are identified and transformed into a form that can be recognized by the controller using the following fuzzification steps:

a. Identification of all inputs and outputs and their operational ranges: In this paper, two-inputs and one-output control variables have been identified. The two input variables are the Length and Load variables whereas the output variable is the new Length. Table shows the control variables and the operational ranges.
b. Decide the number of fuzzy partitions for every input and output variable: This study have divided the universe of discourse of every fuzzy variable into three partitions. Thus, the number of fuzzy partitions, their names and labels are shown in

	Fuzzy Variable	**Number of Fuzzy Partitions**	**Partition Name**	**Partition Label**
Input	Length	3	Small	Sml

			Middle	Mid
			Big	Big
	Load	3	Low	Low
			Medium	Med
			High	Hig
Output	newLength	3	Small	Sml
			Middle	Mid
			Big	Big

c. .

d. Decide type of membership function to be used for every fuzzy partition: This is the final phase of the fuzzification process. Here, the appropriate membership functions are selected to define every fuzzy partition that have been described in the previous step. Table 3 shows the type of membership function that have been used for all the fuzzy partitions of all the fuzzy variables.

In summary, fuzzification converts the crisp input and output control variables and their values into their equivalent fuzzy variables and values using the appropriate membership functions. This is a crucial phase in the development of the fuzzy controller.

3.2. Knowledge Base

The fuzzy knowledge base consist of the database and the rule base. The database is used to store the information needed for the smooth operation of the controller. Such information include the names of the fuzzy variables, number of partitions for every variable, type of membership function used for every partition, fuzzy rules, fuzzy operators and any other information required for the smooth functioning of the fuzzy logic controller. Most of these information have been defined in the fuzzification phase and the remaining will be defined in this phase, the inference phase and the defuzzification phase. The rule base is usually made up of a number of fuzzy rules that are expressed in the form: **IF** <CONDITIONS> **THEN** <ACTIONS>. For this study, all rules have the same structure: **IF** (Load **AND** Length) **THEN** (newLength). Now, the number of fuzzy rules need to be determined. However, the number of fuzzy rules is directly related to the number of partitions of the input linguistic variables. In this case, there are two input linguistic variables and each of them has three partitions. Then, the product of the number of partitions of both input variables gives the total number of rules that can be generated. Hence, in this case, nine fuzzy rules are used to represent a set of rules. Furthermore, three different sets of rules have been generated. These sets of rules have been labelled as $FALBA_0$, $FALBA_1$ and $FALBA_2$ and they represent the first, second, and third set of rules respectively. They have also been used to produce the results used in this study. The rules set configurations for $FALBA_0$, $FALBA_1$ and $FALBA_2$ are shown below.

Table 1: Inputs, Output variables, and their operating range

	Crisp Variable	**Fuzzy Variable**	**Operating Range**
Input	Length	Length	0 -100000 (bytes)
	Load	Load	0 – 1000000000 (bps)
Output	Length	newLength	0 -100000 (bytes)

Table 1: Partition information of the fuzzy variables

	Fuzzy Variable	Number of Fuzzy Partitions	Partition Name	Partition Label
Input	Length	3	Small	Sml
			Middle	Mid
			Big	Big
	Load	3	Low	Low
			Medium	Med
			High	Hig
Output	newLength	3	Small	Sml
			Middle	Mid
			Big	Big

Table 3: Choice of type of membership function for every fuzzy partition

	Fuzzy Variable	Fuzzy Partition	Type of Membership Function
Input	Length	Small	Triangular
		Middle	Triangular
		Big	Triangular
	Load	Low	Triangular
		Medium	Triangular
		High	Triangular
Output	newLength	Small	Triangular
		Middle	Triangular
		Big	Triangular

Rule set 0: $FALBA_0$

Rule 1:	IF (Load is Low AND Length is Sml)	THEN	(newLength is Mid);
Rule 2:	IF (Load is Low AND Length is Mid)	THEN	(newLength is Mid);
Rule 3:	IF (Load is Low AND Length is Big)	THEN	(newLength is Mid);
Rule 4:	IF (Load is Med AND Length is Sml)	THEN	(newLength is Mid);
Rule 5:	IF (Load is Med AND Length is Mid)	THEN	(newLength is Big);
Rule 6:	IF (Load is Med AND Length is Big)	THEN	(newLength is Big);
Rule 7:	IF (Load is Hig AND Length is Sml)	THEN	(newLength is Big);
Rule 8:	IF (Load is Hig AND Length is Mid)	THEN	(newLength is Big);
Rule 9:	IF (Load is Hig AND Length is Big)	THEN	(newLength is Big);

Rule set 1: $FALBA_1$

Rule 1:	IF (Load is Low AND Length is Sml)	THEN	(newLength is Mid);
Rule 2:	IF (Load is Low AND Length is Mid)	THEN	(newLength is Sml);

Rule 3:	IF (Load is Low AND Length is Big)	THEN	(newLength is Sml);
Rule 4:	IF (Load is Med AND Length is Sml)	THEN	(newLength is Big);
Rule 5:	IF (Load is Med AND Length is Mid)	THEN	(newLength is Big);
Rule 6:	IF (Load is Med AND Length is Big)	THEN	(newLength is Big);
Rule 7:	IF (Load is Hig AND Length is Sml)	THEN	(newLength is Big);
Rule 8:	IF (Load is Hig AND Length is Mid)	THEN	(newLength is Big);
Rule 9:	IF (Load is Hig AND Length is Big)	THEN	(newLength is Big);

Rule set 2: $FALBA_2$

Rule 1:	IF (Load is Low AND Length is Sml)	THEN	(newLength is Sml);
Rule 2:	IF (Load is Low AND Length is Mid)	THEN	(newLength is Sml);
Rule 3:	IF (Load is Low AND Length is Big)	THEN	(newLength is Sml);
Rule 4:	IF (Load is Med AND Length is Sml)	THEN	(newLength is Mid);
Rule 5:	IF (Load is Med AND Length is Mid)	THEN	(newLength is Mid);
Rule 6:	IF (Load is Med AND Length is Big)	THEN	(newLength is Mid);
Rule 7:	IF (Load is Hig AND Length is Sml)	THEN	(newLength is Big);
Rule 8:	IF (Load is Hig AND Length is Mid)	THEN	(newLength is Big);
Rule 9:	IF (Load is Hig AND Length is Big)	THEN	(newLength is Mid);

3.3. Fuzzy Inference Engine

Fuzzy inferencing implies the process of making a fuzzy decision based on the fuzzy input values. The inferencing process involves the evaluation of input fuzzy values, the activation of the affected rules and their accumulation. Finally, the fuzzy rules that have been activated are aggregated to produce a single crisp output for each of the output variables using a deffuzification technique. Tabl 4 shows two configurations of the fuzzy inference engine that have been used for this study.

Table 4: Fuzzy Inference Engine Configuration

No.	**Parameter**	**Configuration 1**	**Configuration 2**
1	T-Norm (AND)	Minimum	Minimum
2	S-Norm (OR)	Maximum	Maximum
3	Activation	Minimum	Minimum
4	Accumulation	Maximum	Algebraic-sum
5	Inference mechanism	Mamdani	Mamdani

3.4. Deffuzification

In this phase, the computed output fuzzy value is converted into a crisp value using a defuzzification technique.

This crisp value is the new threshold of the length burst assembler to be used in generating the next burst. This study used the Centroid/Centre of Gravity (CoG) and the Largest of Maximum (LoM) defuzzification techniques. The two techniques have been used to tune the performance of the fuzzy logic controller.

3.5. Testing and Tuning

The testing and tuning of a fuzzy logic controller can be done in a number of ways as described by Ilyas and his colleagues [46]. In this study, the tuning involved changing only one of the following settings at a time: the rules in the rule base; the aggregation method; and the defuzzification technique. From the combinations of the above settings, 12 unique configurations were formed as described in Table 5 to show the performance of the FALBA algorithm.

Table 5: FALBA algorithm tuning configurations

Configuration		Rules Set	Aggregation Method		Defuzzification Technique	
Tuning Number	Label		Max	Sum	CoG	LoM
1	$FALBA_{0CM}$	0	Yes		Yes	
2	$FALBA_{1CM}$	1	Yes		Yes	
3	$FALBA_{2CM}$	2	Yes		Yes	
4	$FALBA_{0CS}$	0		Yes	Yes	
5	$FALBA_{1CS}$	1		Yes	Yes	
6	$FALBA_{2CS}$	2		Yes	Yes	
7	$FALBA_{0LM}$	0	Yes			Yes
8	$FALBA_{1LM}$	1	Yes			Yes
9	$FALBA_{2LM}$	2	Yes			Yes
10	$FALBA_{0LS}$	0		Yes		Yes
11	$FALBA_{1LS}$	1		Yes		Yes
12	$FALBA_{2LM}$	2		Yes		Yes

3.5.1. Simulation Parameters

The simulation parameters used to evealuate the proposed FALBA algorithms are shown in Table 6.

Table 6: OBS Simulation Parameters and Settings

No.	**Parameter**	**Value**
1	Network Topologies	NSFNET
2	Number of channels per link	4 (3 data and 1 control)
3	Bandwidth per channel [BW] (Gbps)	1
4	Traffic Model	Poisson
5	Packet Size (Bytes)	1250
6	BCP processing Time (us)	10
7	Scheduling Scheme	LAUC
8	Signalling Scheme	JET
9	Wavelength Conversion	On
10	Burst Segmentation	Off
11	Deflection Routing	Off

No.	Parameter		Value
12	Optical Buffers (FDL)		Off
13	Burst Threshold (Bytes)	Minimum (m)	1500
		Maximum	100000
		Initial (l)	60000
14	Offered Load	Minimum	0.1
		Maximum	1
		Increment	0.1

3.5.2. Simulation Setup

The FALBA algorithm was designed and implemented using the tools and environment shown in Table 7. Fig. 4 shows the fuzzylite fuzzy logic controller design tool and the graphical representation of the FALBA algorithm's fuzzy logic controller.

Table 7: Design Tools and Implementation Environment

	Item	Description
Hardware	Processor type	Intel R Core^{TM} i5
	Processor speed	2.00 GHz
	Hard Disk Drive	1Terabyte
	Memory	8 Gigabyte
Software	Operating System	Fedora 18 (Linux)
	Network Simulator	Omnet++ 4.2.2 [47]
	OBS Framework	OBSModules [48]
	Fuzzy Logic Library	Fuzzylite version 5.0 [49]
	Compiler	GCC

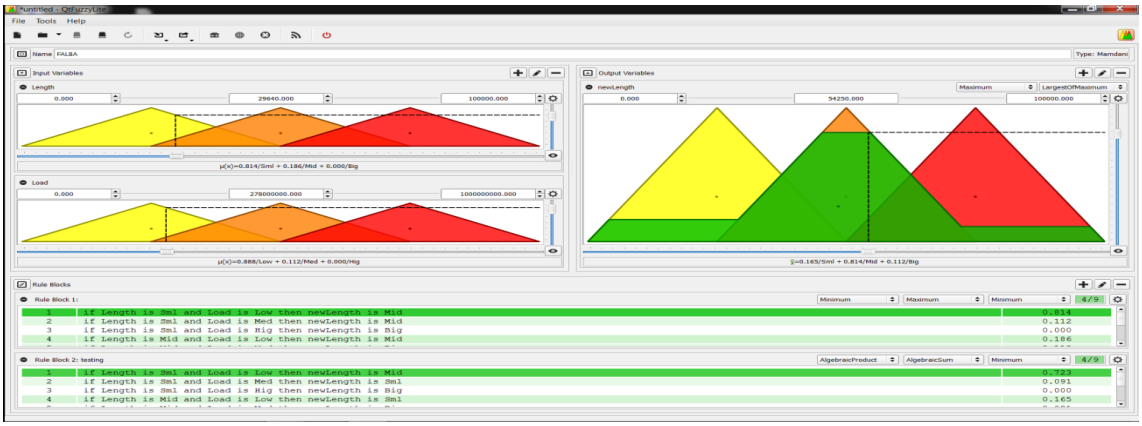

Figure 4: A sample operation of the FALBA FLC showing the Input and Output variables, the Membership Functions and some rules that have been fired to produce a new output.

4. Results and Discussion

This section analyses and discusses the results of the fuzzy-based adaptive length burst assembly (FALBA) algorithm in terms of the burst loss ratio (BLR) and burst end-to-end delay. The algorithm was compared against the fuzzy adaptive threshold (FAT) algorithm [39] in order to evaluate its performance. In the following sections, only the rules sets were changed while the deffuzification techniques and accumulation methods remained constant. Therefore, the results are grouped into four sets with each group having three different results of the FALBA algorithm. These groupings resulted from several tuning of the algorithm. Hence, the plots for FALBA have labels with meanings derived from Table 5. In the following plots, the algorithm is labelled as $FALBA_{XYZ.}$ Where FALBA denotes the name of the algorithm (FALBA) which is followed by the rules set number which is denoted by X , then followed by the deffuzification technique which is denoted by Y and finally the accumulation method which is denoted by Z. For example, $FALBA_{0CM}$ implies algorithm FALBA was executed with configuration 0, C, and M; where 0, C, and M are the rules set 0, Centroid deffuzification technique and the largest of Maximum accumulation method respectively.

4.1. Loss and Delay for Tunings 1, 2 and 3

Fig. 5 and Fig. 6 show the plots of burst loss ratio (BLR) and burst end-to-end delay as a function of normalized offered load for three configurations of FALBA. The configuration consists of three sets of rules: 0, 1 and 2; whereas the centriod and maximum are the deffuzification technique and accumulation method used respectively. Fig. 5 depicts the effect of different fuzzy rules sets on the performance of BLR. It shows that at loads from 0.1 to 0.4, all the configurations of FALBA have good loss performance. However, as the load increases from 0.5 to 1 inclusive, the different behaviours of the algorithms manifested. The figure shows that at all loads, $FALBA_{0CM}$ exhibits the best loss perfomance for this configuration compared to other FALBA configuration and FAT algorithm. The low BLR of $FALBA_{OCM}$ is attributed to its ability to adapt the burst length threshold such that longer burst size thresholds are produced by the fuzzy logic controller. Thus, generating bigger burst size lead to contention reduction at the core nodes because fewer number of bursts are transmitted throught the core nodes which results in low BLR. Fig. 6 shows the effect of different fuzzy rules sets on the performance of burst end-to-end delay. At low loads starting from 0.1 to 0.3, the FAT algorithm exhibits a better delay perfromance than the FALBA algortihms. However, from loads 0.4 through load 1, different FALBA configurations showed better perfromances than the FAT algorithm. The higher delay incured by $FALBA_{0CM}$ is as a result of the large bursts generated by the algorithm which leads to longer processiing delay at the core nodes. However, such bursts have lower burst loss ratio as shown in Fig. 5.

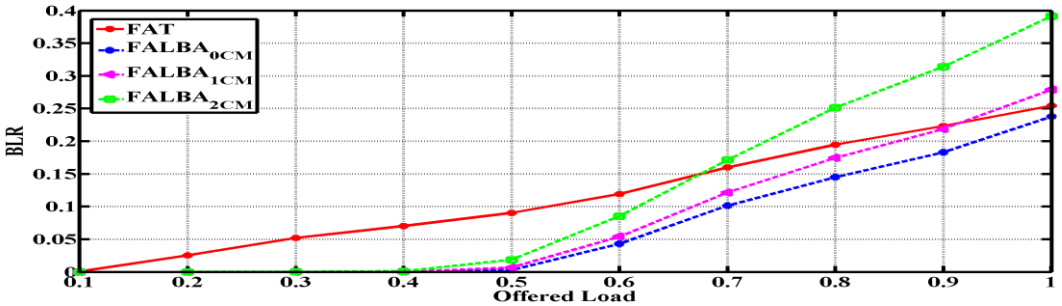

Figure 5: BLR versus offered load for centriod deffuzification technique and maximum accumulation method.

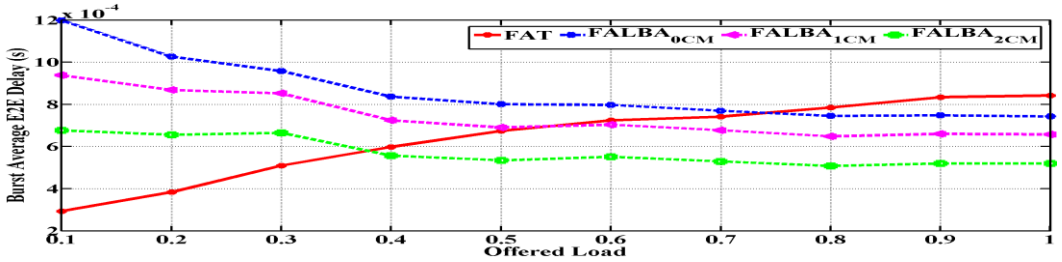

Figure 6: Burst end-to-end delay versus offered load for centriod deffuzification technique and maximum accumulation method.

4.2. Loss and Delay for Tunings 4, 5 and 6

Fig. 7 and Fig. 8 show the plots of BLR and burst end-to-end delay as a function of normalized offered load for tunings 4, 5 and 6 as shown in Table 5. The configuration consists of three sets of rules: 0, 1 and 2; with centriod and sum being the deffuzification technique and accumulation method respectively. Fig. 7 depicts the effect of centroid deffuzification technique and sum accumulation method on different fuzzy rules sets and their overall effect on the performance of BLR. As in the case of $FALBA_{0CM}$, $FALBA_{0CS}$ exhibits the best loss performance. The causes for this performance are the same as in Section 4.1. The other results also follow the same loss pattern as in tunings 1,2 and 3 described in Table. This shows that the change in accumulation method from maximum to sum while the results of centroid deffuzification technique remain unchanged has little effect on the burst loss performance. Similarly, the same interpretations for tunings 1, 2 and 3 given earlier holds true for the burst end-to-end delay plots shown in Fig. 8.

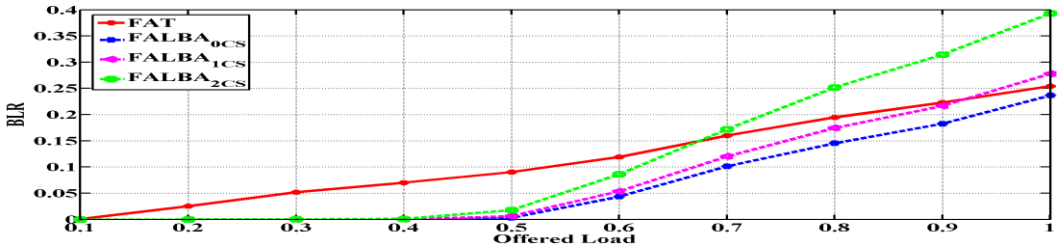

Figure 7: Burst loss ratio versus offered load for centriod deffuzification technique and sum accumulation method.

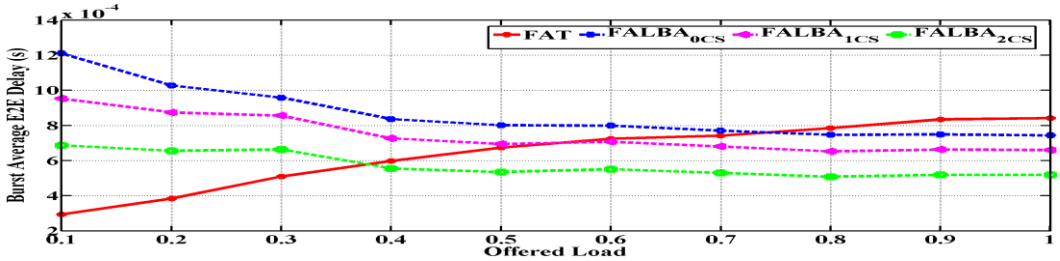

Figure 8: Burst end-to-end delay versus offered load for centriod deffuzification technique and sum accumulation method.

4.3. Loss and Delay Analysis for Tunings 7, 8 and 9

Fig. 9 and Fig. 10 show the plots of BLR and burst end-to-end delay as a function of normalized offered load for tuninngs 7, 8 and 9 as shwon in Table 5. The configuration consists of three sets of rules: 0, 1 and 2; with largest of maximum and maximum being the deffuzification technique and accumulation method respectively. Fig. 9 shows the effect of the largest of maximum deffuzification technique and the maximum accumulation method on different fuzzy rules sets and their overall effect on performance of BLR. The figure shows that the BLR of the FALBA algorithms have been considerably reduced. This improved BLR performance is attributed to the sensitivity of the FLC for producing bursts of suitable sizes when the largest of maximum deffuzification technique and the maximum method are used. The rule set configuration of $FALBA_{0LM}$ favours the generation of the largest bursts when compared to $FALBA_{1LM}$, $FALBA_{2LM}$ and FAT. Fig. 10 shows the bursts average end-to-end delay expereicned by the FALBA algorithms using three sets of fuzzy rules. The figurs shows that the FAT algorithm exhibited the best BLR perfomance at loads between 0.1 and 0.4 inclusive. However, as the load increased, $FALBA_{2LM}$ exhibited the best perfromance for the rest of the loads. This show that $FALBA_{2LM}$ produced bursts sizes with the smallest thresholds among its peers. Therefore, the bursts experienced the least delay when compared with/to the other FALBA configuratons and the FAT algortihm.

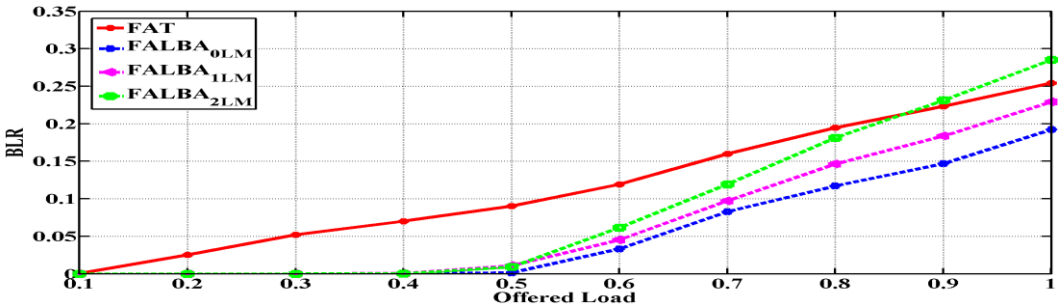

Figure 9: BLR versus Offered Load for largest of maximum deffuzification technique and maximum accumulation method.

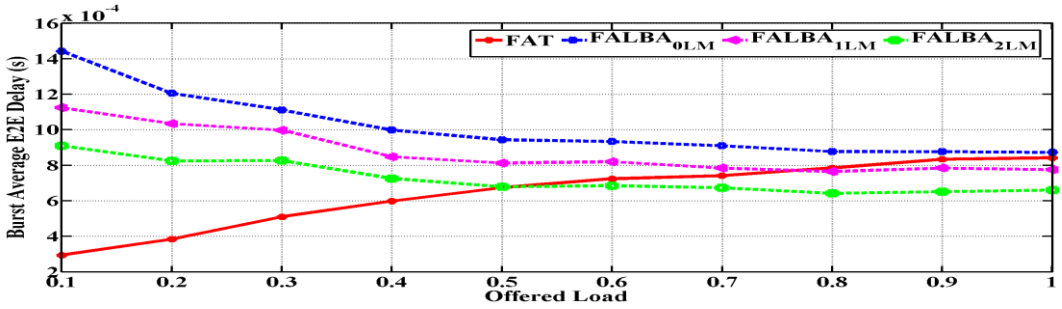

Figure 10: Burst end-to-end delay versus offered load for largest of maximum deffuzification technique and maximum accumulation method.

4.4. Loss and Delay Analysis for Tunings 10, 11 and 12

Fig. 11 and Fig. 12 respectively show the plots of BLR and burst end-to-end delay as a function of normalized offered load for tunings 10, 11 and 12 as shown in Table 5. The configuration consists of three sets of rules: 0, 1 and 2; with largest of maximum and sum being the deffuzification technique and accumulation method respectively. Fig. 11 shows the BLR performances for $FALBA_{0LS}$, $FALBA_{1LS}$, $FALBA_{2LS}$ and FAT algorithms. The figure shows that the change in the accumulation method has little effect on the BLR performance for $FALBA_{0LS}$. However, other configuration of FALBA have shown a drop in perfromance when the load is high. Even with these drop in BLR performance, the FALBA configurations still performed better than the FAT algorithm except for $FALBA_{2LS}$ which shows a drop in perfromance at high loads ranging from 0.9 and 1. Fig. 12 shows a significant improvement in the delay performance of $FALBA_{2LS}$ which is at the expense of its high BLR. This improved deley performance can be attributed to the short bursts it generates.

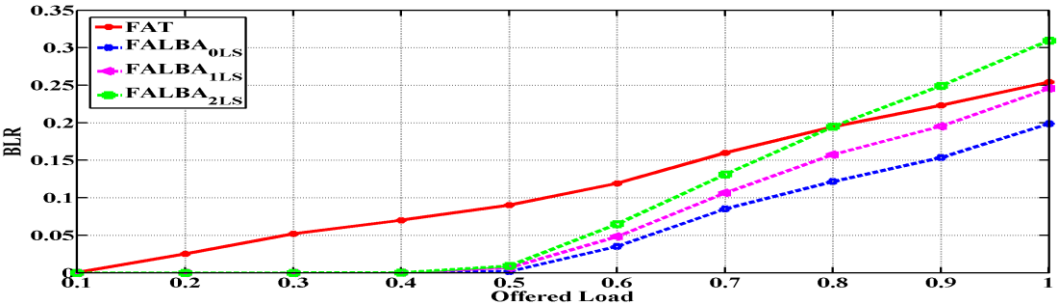

Figure 11: BLR versus offered load for largest of maximum deffuzification technique and sum accumulation method.

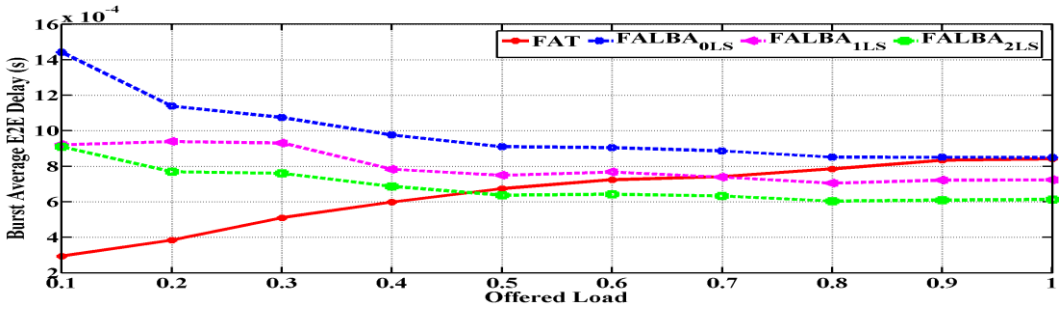

Figure 12: Burst end-to-end delay versus offered load for largest of maximum deffuzification technique and sum accumulation method.

5. Conclusion

In this paper, an intelligent burst assembly algorithm that is based on fuzzy logic has been designed and implemented for optical burst switched network. Specifically, this paper describes the detailed design and implementation of the fuzzy-based adaptive length burst assembly (FALBA) algorithm. The FALBA algorithm has been modelled as a fuzzy control process in which the burst length threshold of the conventional threshold-based burst assembly algorithm is made adaptive subject to the network offered load, current value of the predefined burst length threshold and the bandwidth of the outgoing channel. Furthermore, it describes the testing and tuning configurations of the algorithm and the additional simulation parameters used to generate the results. It was shown that the FALBA algorithm can be tuned intelligently using fuzzy logic. Hence, $FALBA_{0LM}$ has the best BLR performance when compared to its other configurations and the FAT. However, with respect to delay, FAT only outperforms all configurations of FALBA at low loads (0.0-0.4) but the performance of FAT decreases as the load (0.4-1.0) increases. Therefore, at high loads (0.4-1.0) $FALBA_{2CS}$ has the best delay performance. In conclusion, $FALBA_{0LM}$ should be used for loss-sensitive applications while $FALBA_{2CS}$ should be used for delay sensitive applications.

References

[1] C. Qiao and M. Yoo, "Optical burst switching (OBS)–a new paradigm for an Optical Internet," Journal of high speed networks, vol. 8, pp. 69-84, 1999.

[2] B. Mukherjee, Optical WDM Networks (Optical Networks): Springer, 2006.

[3] S. Y. Oh, H. H. Hong, and M. H. Kang, "A data burst assembly algorithm in optical burst switching networks," ETRI Journal, vol. 24, pp. 311-322, Aug 2002.

[4] M. Maier, Optical switching networks vol. 324: Cambridge University Press Cambridge, 2008.

[5] P. K. Chandra, A. K. Turuk, and B. Sahoo, "Survey on optical burst switching in WDM networks," in Industrial and Information Systems (ICIIS), 2009 International Conference on, 2009, pp. 83-88.

[6] F. Farahmand, V. M. Vokkarane, J. P. Jue, J. J. P. C. Rodrigues, and M. M. Freire, "Optical burst switching network: A multi-layered approach," Journal of High Speed Networks, vol. 16, pp. 105-122, 2007.

[7] J. P. Jue, W. H. Yang, Y. C. Kim, and Q. Zhang, "Optical packet and burst switched networks: a review," IET Communications, vol. 3, pp. 334-352, 2009.

[8] T. F. Fernandez, "Challenges, Issues and Research directions in Optical Burst Switching," International

Journal of Computer Applications Technology and Research, vol. 2, pp. 131-136.

[9] H. Kaur and R. Kaler, "Burst Assembly and Signaling Protocols in OBS," in Proceedings of National Conference on Challenges and Opportunities in Information Technology COIT-2007 RIMT-IET, Mandi Gobindgarh, India, pp. 268-272.

[10] J. Li, C. Qiao, and Y. Chen, "Recent progress in the scheduling algorithms in optical-burst-switched networks [Invited]," J. Opt. Netw., vol. 3, pp. 229-241, 2004.

[11] R. Adgaonkar and S. Sharma, "A Review of Burst Scheduling Algorithm in WDM Optical Burst Switching Network," International Journal of Computer Science Issues(IJCSI), vol. 8, 2011.

[12] N. Akar, E. Karasan, K. G. Vlachos, E. A. Varvarigos, D. Careglio, M. Klinkowski, et al., "A survey of quality of service differentiation mechanisms for optical burst switching networks," Optical Switching and Networking, vol. 7, pp. 1-11, 2010.

[13] C. Yahaya, M. S. Abd Latiff, and A. B. Mohamed, "A review of routing strategies for optical burst switched networks," International Journal of Communication Systems, pp. n/a-n/a, 2011.

[14] M. Yoo and C. Qiao, "Just-enough-time (JET): A high speed protocol for bursty traffic in optical networks," Montreal, Can, 1997, pp. 26-27.

[15] A. A. Yayah, Y. Coulibaly, A. S. Ismail, and G. Rouskas, "Hybrid offset-time and burst assembly algorithm (H-OTBA) for delay sensitive applications over optical burst switching networks," International Journal of Communication Systems, 2014.

[16] B. Shihada and P.-H. Ho, "Transport Control Protocol in Optical Burst Switched Networks: Issues, Solutions, and Challenges," IEEE Communications Surveys & Tutorials,, vol. 10, pp. 70-86, 2008.

[17] A. Ge, F. Callegati, and L. S. Tamil, "On Optical Burst Switching and Self-Similar Traffic," IEEE Communications Letters vol. 4, pp. 98-100, 2000.

[18] B. Kantarci, S. F. Oktug, and T. Atmaca, "Performance of OBS techniques under self-similar traffic based on various burst assembly techniques," Computer Communications, vol. 30, pp. 315-325, 2007.

[19] C. Yuan, Z. Zhang, Z. Li, Y. He, and A. Xu, "A unified study of burst assembly in optical burst switching networks," Photonic Network Communications, vol. 21, pp. 228-237, 2011.

[20] J. Yang, G. Wang, and S. Jia, "Improved adaptive-threshold burst assembly in optical burst switching networks," Chin. Opt. Lett., vol. 5, pp. 325-327, 2007.

[21] V. M. Vokkarane, K. Haridoss, and J. P. Jue, "Threshold-based burst assembly policies for QoS support in optical burst-switched networks," presented at the The Convergence of Information

Technologies and Communications, 2002.

[22] V. M. Vokkarane, Q. Zhang, J. P. Jue, and B. Chen, "Generalized burst assembly and scheduling techniques for QoS support in optical burst-switched networks," in IEEE Global Telecommunications Conference, 2002. GLOBECOM'02., 2002, pp. 2747-2751.

[23] Z. Zhang, J. Luo, Q. Zeng, and Y. Zhou, "Novel threshold-based burst assembly scheme for QoS support in optical burst switched WDM networks," in Performance and Control of Next-Generation Communications Networks, 2003, pp. 250-256.

[24] X. Yu, Y. Chen, and C. Qiao, "A Study of traffic statistics of assembled burst traffic in optical burst-switched networks," 2002, pp. 149-159.

[25] X. Cao, J. Li, Y. Chen, and C. Qiao, "Assembling TCP/IP packets in Optical Burst Switched Networks," presented at the IEEE GLOBECOM'02, Taipei, China, 2002.

[26] M. Mangwala, B. B. Sigweni, and O. O. Ekabua, "Implementation of Efficient Burst Assembly Algorithm with traffic prediction," Computer Technology and Application, vol. 4, pp. 153-161, 2013.

[27] H. Kaur and R. S. Kaler, "Burst Assembly and Signaling Protocols in OBS," presented at the Proceedings of National Conference on Challenges and Opportunities in Information Technology COIT-2007 RIMT-IET, Mandi Gobindgarh, India, 2007.

[28] K. Seklou, A. Sideri, P. Kokkinos, and E. Varvarigos, "New assembly techniques and fast reservation protocols for optical burst switched networks based on traffic prediction," Optical Switching and Networking, vol. 10, pp. 132-148, Apr 2013.

[29] H.-l. Liu and S. Jiang, "A mixed-length and time threshold burst assembly algorithm based on traffic prediction in OBS network," Int. J. Sensing, Computing & Control, vol. 2, pp. 87-93, 2012.

[30] A. Sideri and E. A. Varvarigos, "New assembly techniques for optical burst switched networks based on traffic prediction," in Optical Network Design and Modeling, ed: Springer, 2007, pp. 358-367.

[31] J. Liu, N. Ansari, and T. J. Ott, "FRR for latency reduction and QoS provisioning in OBS networks," IEEE Journal on Selected Areas in Communications, vol. 21, pp. 1210-1219, 2003.

[32] T. Mikoshi and T. Takenaka, "Improvement of burst transmission delay using offset time for burst assembly in optical burt switching," in 7th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT) 2008, pp. 13-18.

[33] A. K. Garg, "Traffic prediction based burst assembly mechanism for OBS," Optik - International Journal for Light and Electron Optics, vol. 124, pp. 2017-2019, 8// 2013.

[34] B. Kantarci and S. Oktug, "Adaptive Threshold Based Burst Assembly in OBS Networks," presented at the IEEE Canadian Conference on Electrical and Computer Engineering, 2006.

[35] A. Gupta, R. S. Kaler, and H. Singh, "Investigation of OBS assembly technique based on various scheduling techniques for maximizing throughput," Optik - International Journal for Light and Electron Optics, vol. 124, pp. 840-844, 5// 2013.

[36] X. Yi-Yuan and Z. Jian-Guo, "An intelligent segmented burst assembly mechanism in optical burst switching networks," Chinese Physics Letters, vol. 25, p. 2535, 2008.

[37] S. Askar, G. Zervas, D. K. Hunter, and D. Simeonidou, "Adaptive classified cloning and aggregation technique for delay and loss sensitive applications in OBS networks," in Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference, 2011, pp. 1-3.

[38] B. Kantarci and S. Oktug, "Loss rate-based burst assembly to resolve contention in optical burst switching networks," IET communications, vol. 2, pp. 137-143, 2008.

[39] J.-r. YANG, S.-l. JIA, and G. WANG, "Burst assembly algorithm based on fuzzy-adaptive-threshold " Journal of Harbin Engineering University, vol. 6, p. 013, 2007.

[40] W. H. F. Aly, M. F. Zhani, and H. Elbiaze, "On controlling burst loss ratio inside an OBS network," presented at the IEEE Symposium on Computers and Communications, ISCC 2008.

[41] A. Muhammad Umaru, M. S. Abd Latiff, and Y. Coulibaly, "Fuzzy-Based Adaptive Hybrid Burst Assembly Technique for Optical Burst Switched Networks," Journal of Computer Networks and Communications, vol. 2014, p. 10, 2014.

[42] A. M. Umaru, M. S. A. Latiff, and Y. Coulibaly, "A novel fuzzy-based adaptive timer burst assembly algorithm for optical burst switching networks," Journal of Theoretical and Applied Information Technology, vol. 67, pp. 220-227, 2014.

[43] A. M. Umaru, C. Yahaya, and M. S. A. Latiff, "A Fuzzy-based Burst Assembly Approach to Reduce End-to-End Delay in OBS Networks," in 5th International Conference on Photonics, Kuala Lumpur, Malaysia, 2014, pp. 29-31.

[44] A. M. Umaru, M. S. Abd Latiff, and Y. Coulibaly, "Fuzzy-Based Adaptive Length Burst Assembly Technique for Loss Reduction in Optical Burst Switched Networks," Advanced Science Letters, vol. 22, pp. 2681-2685, 2016.

[45] B. Lakshmanan, S. Ramasamy, and S. Alavandar, "Adaptive Burst Assembly Algorithm for Reducing Burst Loss and Delay in OBS Networks.," Asian Journal of Information Technology, vol. 15, p. 10,

2016.

[46] A. Ilyas, S. Jahan, and M. Ayyub, "Tuning Of Conventional Pid And Fuzzy Logic Controller Using Different Defuzzification Techniques," International Journal of Scientific & Technology Research, vol. 2, 2013.

[47] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," presented at the Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, 2008.

[48] F. Espina, J. Armendariz, N. Garc, D. Morat, M. Izal, and E. Maga, "OBS Network Model for OMNeT++: A Performance Evaluation," presented at the Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, Torremolinos, Malaga, Spain, 2010.

[49] J. Rada-Vilela. (2013, 01-03-2013, URL: http://www.fuzzylite.com). fuzzylite: A fuzzy logic control library written in C++.