Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2014, Article ID 803518, 10 pages
http://dx.doi.org/10.1155/2014/803518

*Research Article*

# Fuzzy-Based Adaptive Hybrid Burst Assembly Technique for Optical Burst Switched Networks

**Abubakar Muhammad Umaru, Muhammad Shafie Abd Latiff, and Yahaya Coulibaly**

*Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia*

Correspondence should be addressed to Abubakar Muhammad Umaru; amumaru@yahoo.com

Received 19 June 2014; Revised 4 October 2014; Accepted 4 October 2014; Published 3 November 2014

Academic Editor: Rui Zhang

Copyright © 2014 Abubakar Muhammad Umaru et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The optical burst switching (OBS) paradigm is perceived as an intermediate switching technology for future all-optical networks. Burst assembly that is the first process in OBS is the focus of this paper. In this paper, an intelligent hybrid burst assembly algorithm that is based on fuzzy logic is proposed. The new algorithm is evaluated against the traditional hybrid burst assembly algorithm and the fuzzy adaptive threshold (FAT) burst assembly algorithm via simulation. Simulation results show that the proposed algorithm outperforms the hybrid and the FAT algorithms in terms of burst end-to-end delay, packet end-to-end delay, and packet loss ratio.

## 1. Introduction

Optical burst switching (OBS) [1] is envisioned as the intermediate next-generation optical switching paradigm that is capable of providing huge bandwidth. Because of its statistical multiplexing capability, OBS has better resource utilization over optical circuit switching (OCS) [2]. Furthermore, OBS can work efficiently in a buffer-less environment unlike optical packet switching (OPS) that requires optical memory which is still technologically immature [3]. The OBS architecture (Figure 1) consists of edge (ingress/egress) and core nodes which are interconnected by high speed multichannel wavelength-division multiplexing fibre links. The edge node is responsible for burst assembly/disassembly, offset-time computation, signalling, and routing and wavelength assignment, while scheduling and contention resolution are performed at the core node [4]. In OBS, data and control channels are decoupled, thereby allowing data and control packets to be transmitted on separate channels. This feature allows OBS to eliminate the need for optical buffers at the core nodes. Also, stringent synchronization requirement between a data burst and its control packet is reduced [5]. OBS requires optical switching devices with high response rate in order to utilize the huge bandwidth provided by the optical fiber. Moreover, these optical devices should have low switching power [6] and low insert loss, should be compact in size and easy to integrate, and should not be affected by polarization [7].

Comprehensive reviews of different aspects of OBS have been conducted in [8–12] and the following issues have been identified: burst assembly, burst contention, quality of service (QoS) provisioning, routing and wavelength assignment, and core node scheduling. Contention is a major issue because it leads to loss, increased delay, and poor network throughput. Several contention avoidance and resolution techniques have been proposed in OBS literatures [13]. Avoidance techniques are used at the edge nodes to prevent contention while resolution techniques are applied at the core nodes to resolve contention when it occurs. Also, resolution techniques such as wavelength converters and fiber delay lines require additional hardware. Other resolution techniques such as burst segmentation incur additional processing overhead [14] while deflection routing increases delay.

Avoidance techniques utilize electronic memory at the edge in order to prevent contention from occurring at the network core nodes. Therefore, the process by which traffic is injected into an OBS network has an effect on the performance of the network. Thus, a careful selection of a burst assembly technique can regulate the congestion level of a network, thereby reducing the occurrence of contention and its effect on the network. Hence, for the aforementioned

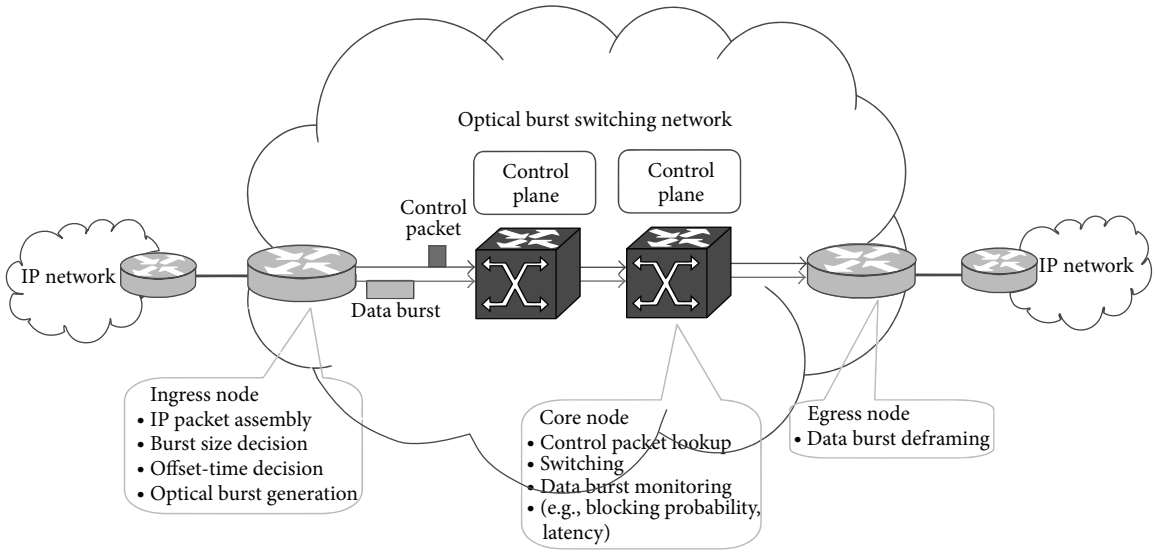



Figure 1: An OBS network architecture [15, Figure 2].

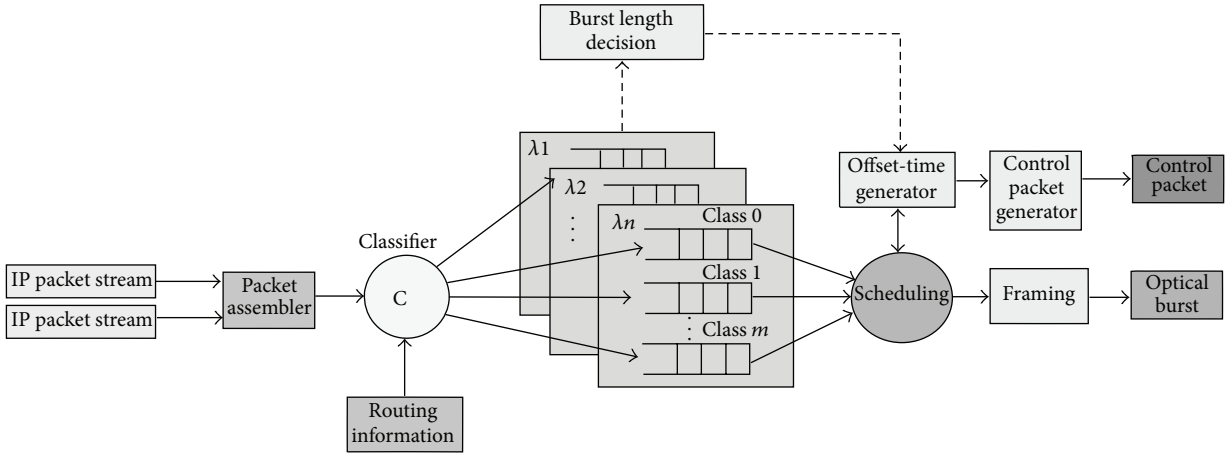



Figure 2: A functional model of an ingress node [15, Figure 3].

reason, burst assembly is a suitable candidate for congestion reduction.

Taking into account the benefits of fuzzy logic as discussed in [16–18], this paper proposes and evaluates a fuzzy-based adaptive hybrid burst assembly (FAHBA) algorithm that aims to reduce the average end-to-end delay experienced by burst and packet in an OBS network.

The paper is organized as follows: Section 2 presents a review of related studies. The proposed burst assembly algorithm is described in Section 3; Section 4 describes the simulation setup and presents the analysis and discussion of the results. Finally, the paper is concluded in Section 5.

## 2. Review of Related Studies

The burst assembly process starts at an ingress node upon the arrival of a client data. In the case of an IP packet, the routing module reads the IP header and determines its destination. The packet is passed through the classifier which forwards the packet to the appropriate destination buffer (DB). Upon the arrival of the first bit of data at any empty DB, timer counter, burst length counter, or both are initialized. The counters will continue counting until either the time threshold or burst length threshold is reached. When either threshold is reached, a burst and its corresponding control packet (BCP) are generated and all threshold counters are reinitialized again. The BCP is sent ahead of its burst to make the necessary reservations while the burst waits at the ingress node output buffer. After a period of time also known as the offset time, the burst is then transmitted into the OBS network. A functional model of an ingress router is shown in Figure 2.

The two basic burst assembly algorithms in OBS are the timer algorithm [19] and the length or threshold-based algorithm [20]. The timer algorithm uses a timer for burst assembly while the threshold-based algorithm uses the number of packets as the threshold for burst generation. Both assembly

algorithms are simple to implement. However, at low loads packets inside the bursts generated using the threshold-based assembly algorithm experience high end-to-end delay, while, at high loads, many fixed-size bursts will be injected into the network. Similarly, the timer burst assembly algorithm will generate average-size bursts under low loads while, at high loads, it will generate large but variable-size bursts. Therefore, under high loads, either algorithm will increase the burst blocking rate in the core node. Generally, fixed-size bursts are always generated for the threshold-based algorithm while variable-size bursts are periodically generated for the timer assembly algorithm.

The min-burst-length-max-assembly-period (MBMAP) [21] algorithm was proposed to address the limitations of the timer and threshold-based assembly algorithms. MBMAP is a hybrid algorithm that uses the burst generation conditions used by both the timer and the length algorithms. Even though MBMAP solved the problems of the basic algorithms, it does not consider the dynamic nature of the incoming traffic. That is, at certain loads, the MBMAP will perform exactly like the basic assembly algorithms.

The threshold-based mixed assembly algorithm [22] aggregates packets of different classes into the same burst. High priority packets are stored at the head while low priority packets are stored at the tail burst. When contention occurs, the tail dropping segmentation policy is employed in order to provide a resource for the contending burst. This scheme supports only two classes of traffic in a burst and in addition it still suffers from the drawbacks of the length threshold algorithm. Reference [23] is similar to the work proposed in [22] except that the high priority packets are placed at the tail of the burst. The work in [23] suffers from out-of-order packet delivery problem which increases delay.

Authors in [24, 25] have 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. These algorithms were used to study the performance of core scheduling under different types of traffic. Even though the algorithms are adaptive, additional delay is still incurred due to the large burst sizes that are generated.

A traffic prediction based assembly algorithm is the mixed-threshold burst assembly (MTBA) [26] algorithm. MTBA uses a traffic prediction to calculate the expected length of a burst. The predicted length is then added to the BCP. The BCP 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 in OBS network 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.

Authors in [27] also used traffic prediction techniques to propose a set of burst assembly schemes that work together with a fast reservation protocol (FRP). The assembly schemes aim to reduce the queuing delay while the FRP works towards reducing the burst end-to-end delay. The predicted result is used as a criterion to assemble 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. Therefore, the assembly process does not need to wait for the burst to be assembled before sending the BCP. These approaches reduce the burst assembly and reservation times and improve packet end-to-end delay. However, the predicted values may be inaccurate and therefore lead to poor resource utilization.

The fuzzy adaptive threshold algorithm (FAT) [28] is based on the length threshold burst assembly algorithm. FAT takes into account the amount of incoming traffic in order to adjust the length threshold of the assembler. FAT has the capability to intelligently adjust the assembly threshold. However, at low loads, bursts generated using FAT will experience high delay. And at high loads, FAT generates large bursts that increase the blocking probability.

Authors of [29] have proposed an adaptive classified cloning and aggregation scheme (ACCS) for high priority traffic that aims to provide better performance in terms of loss and end-to-end delay for high priority traffic. ACCS uses network loss-rate information to adaptively adjust the hybrid burst assembly thresholds. 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. Hence, low priority packets are dropped.

From the above review it can be concluded that current assembly algorithms still suffer from high end-to-end delay. Therefore, this paper proposes a new assembly algorithm to address end-to-end delay issue in OBS network. The proposed algorithm is described in Section 3.

## 3. Materials and Method

In hybrid burst assembly (HBA) algorithm, the timer and the length thresholds are fixed. These thresholds are used to generate a burst when either of the threshold conditions is satisfied in the assembler. Therefore, in the proposed method, the fixed threshold values are made adaptive subject to the incoming offered load and the bandwidth capacity of the output channel. The HBA process is modelled as a multiple-input-multiple-output (MIMO) fuzzy logic control process in which three control variables have been selected. The selected control variables are the timer, the length, and the assembler offered load. The assembler offered load (or load) is the aggregated traffic at an assembler within a time interval. And it can range from microseconds to seconds.

The load, the timer, and the length control variables are the inputs to the fuzzy logic controller, while a new timer and new length values are the outputs of the controller. The controller contains the inbuilt intelligence required to adapt the threshold values of the HBA and it is executed once at the end of every cycle. A cycle is defined as a period of time set by a microtimer. The microtimer is independent of the underlying HBA timer. Similarly, another timer also known as a macrotimer is also executed periodically to reset the load counter variable. Whenever the fuzzy logic controller is executed, it produces a new set of timer and length threshold values which will be used for the next cycle of the burst assembly process. Algorithm 1 shows the pseudocode of the fuzzy-based adaptive hybrid burst assembly algorithm.

```
(1) Begin Initialization
(2) Timer = t;                                   //t is the timer threshold in seconds
(3) Length = l;                                  //l is the length threshold in bytes
(4) Load = 0;                                    //Load at the assembler
(5) MinBurstLength = m;                          //m is minimum burst size
(6) newTimer = newLength = 0;
(7) perSecondTraffic = 0;                        //Load accumulator
(8) setFuzzyMicroTimer(k);                       //k seconds micro-timer
(9) setFuzzyMacroTimer(K);                       //K seconds macro-timer
(10) setTimer(Timer);                            //Set the assembler Timer threshold
(11) setLength(Length);                          //Set the assembler Length threshold
(12) End Initialization
(13) while (IsTrafficStillArriving?) do
(14)      perSecondTraffic = perSecondTraffic + packetSize;
(15)      if ((getTimer() == Timer) OR (getLength() == Length)) then
(16)           Call hybridBurstAssembler() Return burst;          //generate a burst
(17)           setTimer(Timer);                                   //reset the timer
(18)           setLength(Length);                                 //reset the length counter
(19)      end if
(20)      if (getFuzzyMicroTimer() == k) then
(21)           setFuzzyMicroTimer(k);
(22)           Load = perSecondTraffic;
(23)           Call FuzzyEngine(Timer, Length, Load) Return newTimer, newLength;
(24)           if (newLength ≤ MinBurstLength) then
(25)                Length = MinBurstLength;
(26)                Timer = newTimer;
(27)           else
(28)                Length = newLength;
(29)                Timer = newTimer;
(30)           end if
(31)      end if
(32)      if (getFuzzyMacroTimer() == K) then
(33)           setFuzzyMacroTimer(K);                             //reset the fuzzy macro-timer
(34)           Load = perSecondTraffic;
(35)           perSecondTraffic = 0;                              //reset to zero
(36)      end if
(37) end while
```

Algorithm 1: Fuzzy-based adaptive hybrid burst assembly (FAHBA) algorithm.

The fuzzy rules are developed based on experiment and they are stored in the fuzzy logic controller. The controller uses fuzzy logic operations in conjunction with the rules and inputs to compute the appropriate timer and length threshold values. These new threshold values are used to control the burst generation process for the underlying hybrid burst assembler.

Therefore, by using fuzzy logic, the new assembler can handle uncertainties associated with the highly variable nature of input traffic. Generally, FAHBA uses the fuzzy inputs (timer, length, and load), the fuzzy rules, and the bandwidth of the outgoing channel to compute the new set of threshold values using fuzzy logic. The fuzzy logic control process consists of the following three steps.

*Step I*. Fuzzification converts the crisp input and output control variables and values to their equivalent fuzzy (linguistic) variables and values using the appropriate membership functions. The crisp input control variables are timer, length, and load and, for simplicity, we maintain the same names of the crisp variables for their fuzzy equivalent. Similarly, the crisp output control variables are newTimer and newLength and again for simplicity we maintain the same names of the crisp variables for their fuzzy equivalent. The triangular membership function is used in this study. Table 1 shows the summary of the input and output variables. Figure 3 shows a graphical representation of the input and output linguistic variables and their membership functions.

*Step II*. Fuzzy inferencing implies the process of making a fuzzy decision based on the fuzzy input values. The inferencing process involves the computation of input fuzzy values, their aggregation, and then the activation of the affected fuzzy rules. Finally, the fuzzy rules that have been activated are accumulated to produce a single crisp output for each of the output variables in step III. The 27 fuzzy rules used for this study are shown in Table 2. Each line in the table represents a fuzzy rule with its corresponding inputs and outputs. Fuzzy

Table 1: Crisp and fuzzy input and output variables.

| | Crisp variable | Fuzzy variable | Fuzzy values/terms set |
|---|---|---|---|
| Input | Timer | Timer | (Sht: short, Avg: average, and Lng: long) |
| | Length | Length | (Sml: small, Mid: middle, and Big: big) |
| | Load | Load | (Low: low, Med: medium, and Hig: high) |
| Output | newTimer | newTimer | (Sht: short, Avg: average, and Lng: long) |
| | newLength | newLength | (Sml: small, Mid: middle, and Big: big) |

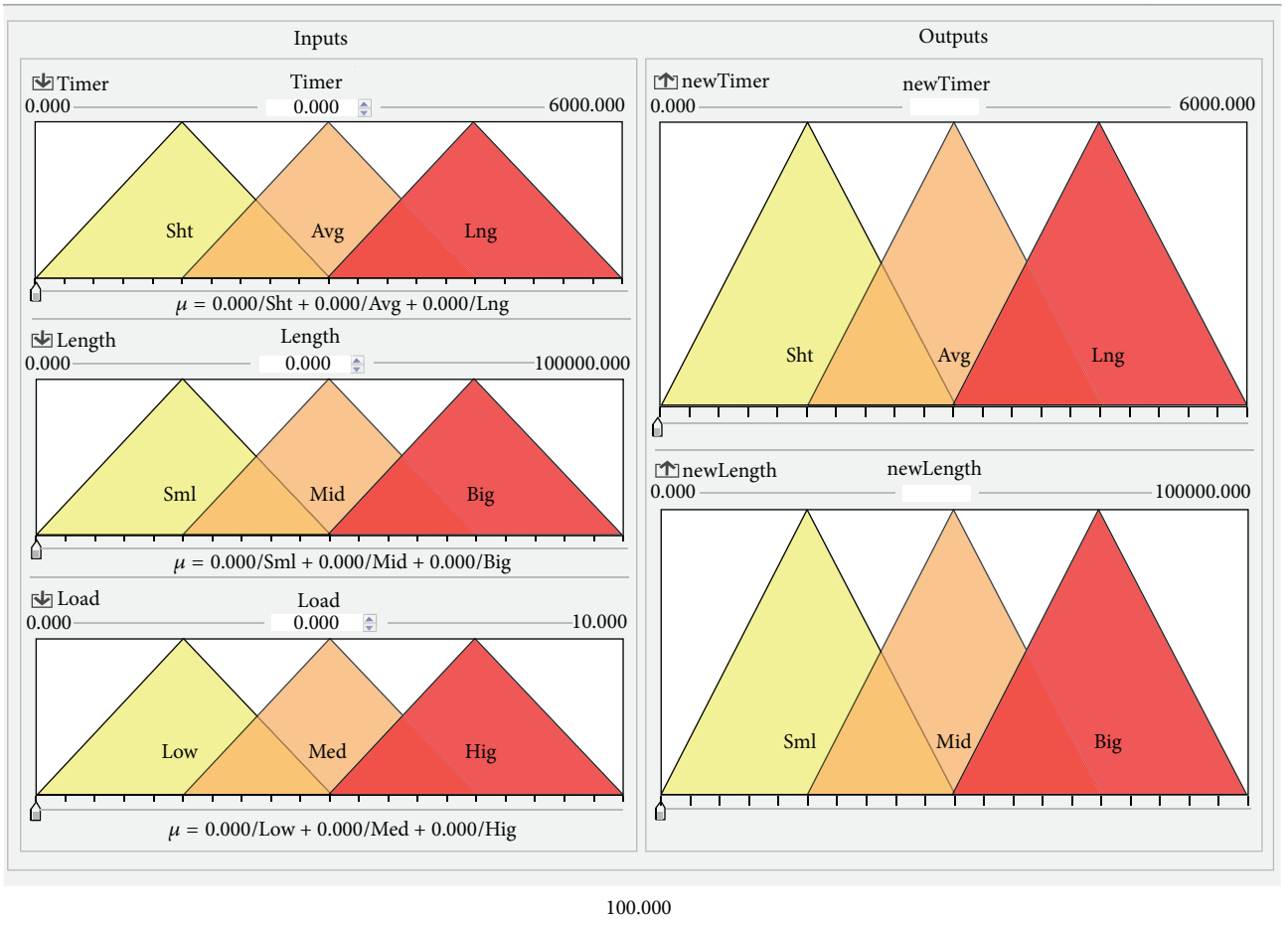



Figure 3: Input and output linguistic variables and their membership functions.

rules have the following syntax: IF (Load AND Timer AND Length) THEN (newTimer, newLength).

*Step III*. Defuzzification: in this step, the computed output fuzzy values are converted into crisp values using a defuzzification technique. The two output crisp values are the newTimer and newLength and they are used to control the burst assembly process.

## 4. Simulation Setup, Results, and Discussion

In order to evaluate the new algorithm, a simulation environment was set up in Omnet++ [30] using OBSModules [31] and fuzzylite [32]. OBSModules provides the OBS network simulation environment while fuzzylite provides the fuzzy logic control library. The node configuration and network simulation parameters are given in Table 3. The NSFNET and COST239 networks consist of 14 and 10 bidirectional source/destination pairs, respectively. All the nodes in both topologies were configured to transmit and receive uniformly distributed traffic. Traffic with exponential interarrival time was used to generate the network offered load according to the formula described in the following as in [33]:

$$L_{\text{offered}} = \frac{\sum_{i=1}^{n} L_i}{\sum_{j=1}^{m} C_j}; \quad (1)$$

(i) $L_{\text{offered}}$ is the network offered load;

Table 2: Fuzzy rules.

| Rule number | Load | Timer | Length | newTimer | newLength |
|---|---|---|---|---|---|
| 1 | Low | Sht | Sml | Avg | Sml |
| 2 | Low | Sht | Mid | Sht | Sml |
| 3 | Low | Sht | Big | Sht | Sml |
| 4 | Low | Avg | Sml | Sht | Sml |
| 5 | Low | Avg | Mid | Sht | Sml |
| 6 | Low | Avg | Big | Sht | Sml |
| 7 | Low | Lng | Sml | Sht | Sml |
| 8 | Low | Lng | Mid | Sht | Sml |
| 9 | Low | Lng | Big | Sht | Sml |
| 10 | Med | Sht | Sml | Sht | Sml |
| 11 | Med | Sht | Mid | Avg | Mid |
| 12 | Med | Sht | Big | Avg | Mid |
| 13 | Med | Avg | Sml | Avg | Sml |
| 14 | Med | Avg | Mid | Avg | Sml |
| 15 | Med | Avg | Big | Sht | Sml |
| 16 | Med | Lng | Sml | Avg | Sml |
| 17 | Med | Lng | Mid | Avg | Sml |
| 18 | Med | Lng | Big | Avg | Mid |
| 19 | Hig | Sht | Sml | Lng | Mid |
| 20 | Hig | Sht | Mid | Lng | Mid |
| 21 | Hig | Sht | Big | Lng | Mid |
| 22 | Hig | Avg | Sml | Avg | Big |
| 23 | Hig | Avg | Mid | Lng | Mid |
| 24 | Hig | Avg | Big | Lng | Mid |
| 25 | Hig | Lng | Sml | Avg | Mid |
| 26 | Hig | Lng | Mid | Avg | Mid |
| 27 | Hig | Lng | Big | Avg | Mid |

Table 3: Simulation parameters and values.

| Number | Parameter | Value |
|---|---|---|
| 1 | Network topologies | NSFNET, COST239 |
| 2 | Number of channels per link | 4 (3 data and 1 control) |
| 3 | Bandwidth per channel (Gbps) | 1 |
| 4 | Packet size (bytes) | 1250 |
| 5 | Packet interarrival time | Exponential |
| 6 | BCP processing time (us) | 10 |
| 7 | Fuzzy microtimer (s) | |
| | $k$ | 0.1 |
| 8 | Fuzzy macrotimer (s) | |
| | $K$ | 1 |
| 9 | Timer threshold (s) | |
| | Minimum | 0.001 |
| | Maximum | 0.006 |
| | Initial ($t$) | 0.004 |
| 10 | Burst threshold (bytes) | |
| | Minimum ($m$) | 1500 |
| | Maximum | 100000 |
| | Initial ($l$) | 60000 |
| 11 | Offered load | |
| | Minimum | 0.1 |
| | Maximum | 1 |
| | Increment | 0.1 |
| 12 | Fuzzy logic controller | |
| | T-norm | Algebraic-product |
| | S-norm | Algebraic-sum |
| | Activation | Minimum |
| | Accumulation | Maximum |
| | Inference engine | Mamdani |
| | Defuzzification | Centre of gravity (CoG) |

(ii) $L_i$ is the amount of traffic generated by a single user in a unit time;

(iii) $C_j$ is the capacity of a single link $j$ (out of $m$) in the network;

(iv) $m$ is the number of links in the network;

(v) $n$ is the number of active users in the network.

Offered load is incremented in steps of 0.1 for every point of measurement. Latest available unused channel scheduling with full wavelength conversion is used in the core node. Our evaluation metrics are average number of bursts sent, average packet end-to-end delay, average burst end-to-end delay, and average packet loss ratio (PLR). The average burst end-to-end delay is the burst delay from the ingress node to the egress node. The packet end-to-end delay consists of queuing delay, burst end-to-end delay, and burst disassembly delay. FAHBA and FAT are used to represent the proposed and the existing fuzzy burst algorithms, respectively, while HBA represents the traditional burst assembly algorithm.

Figures 4 to 7 show the plots of burst end-to-end delay, packet end-to-end delay, packet loss ratio, and the number of bursts that have been generated and sent into the COST239 network.

Figure 4 shows the average burst end-to-end delay versus offered load. At low loads, from 0.1 to 0.4, the traffic arriving at the HBA gradually increased. For the duration of the low loads, the time threshold value for HBA was still in effect, thereby generating small to medium size bursts which have average transmission duration over the network. However, when the load increased to the medium range (loads 0.5 to 0.7), the HBA length threshold condition is rapidly satisfied to generate bursts. As the load entered its high region (from 0.8 to 1), the delay increased steeply because of the fixed-size bursts that were rapidly generated and transmitted into the network. This puts the network into a congestion state with burst competing for resources, thereby increasing burst end-to-end delay. The burst end-to-end delay for FAHBA is lower than that of the HBA because of the intelligence that is built into the fuzzy controller of the assembler. FAHBA adapts the burst generation conditions based on the arriving loads at the ingress node and the available channel bandwidth. The fuzzy

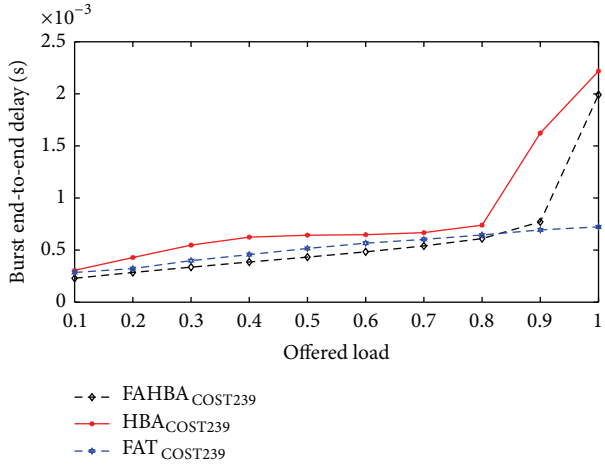



Figure 4: Burst end-to-end delay versus offered load.

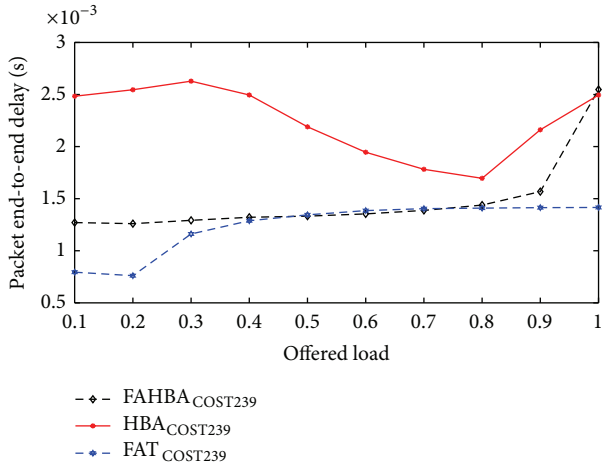



Figure 5: Packet end-to-end delay versus offered load.

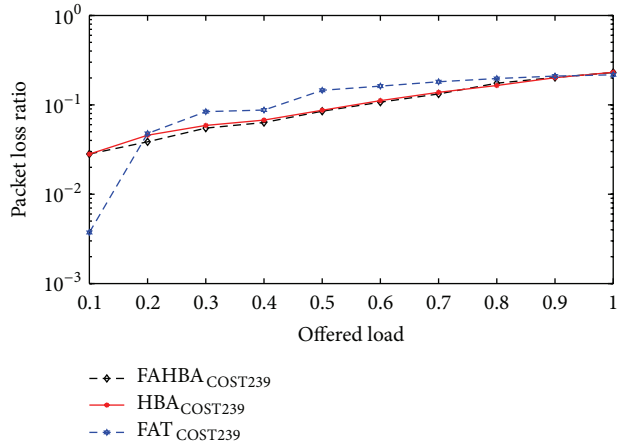



Figure 6: Packet loss ratio versus offered load.

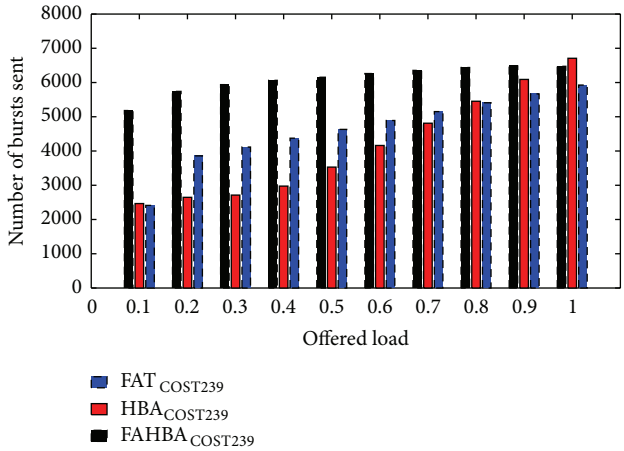



Figure 7: Number of bursts sent versus offered load.

rules satisfy the creation of average-size bursts which have short transmission duration. The burst end-to-end delay for FAHBA gradually increased as the load increased from loads 0.1 to 0.8. But at loads 0.9 to 1, the FAHBA rules satisfy the generation of large bursts which lead to longer transmission delay and congestion on the network. Even with this behaviour, the FAHBA has lower burst end-to-end delay when compared with that of HBA. As for the FAT algorithm, the burst size kept on increasing as the traffic increased. Furthermore, at loads 0.9 and 1, FAT has lower delay due to the fact that it satisfies the upper limit of the burst size threshold which enables it to have fixed transmission duration.

Figure 5 shows the plot of average packet end-to-end delay versus offered load. From loads 0.1 to 0.4, packets transmitted using HBA experience very high delay due to the high queuing delay at the ingress node. Such high delay is a result of low traffic that does not satisfy the length threshold and, therefore, the fixed time threshold must be satisfied in order to generate the burst. The HBA satisfies the length threshold starting from loads 0.5 to 0.8 which signifies that the packets experienced low queuing delays at the ingress nodes. However, at high loads from 0.9 to 1, the packets that arrive at the ingress node experience higher delay because of the fact that they were buffered for a longer time. As for FHBA, the fuzzy rule satisfies early burst generation by adapting the assembler thresholds to satisfy the load at the ingress node. At load 1, FAHBA generates very large bursts in order to satisfy the fuzzy rules. And as a result of that the packets experienced higher end-to-end delay. The good performance exhibited by FAT at loads starting from 0.1 to 0.4 is because FAT continuously adapts its threshold for every burst that was generated. Hence, in this case, FAT was able to detect the low nature of the traffic and then adapt the burst size such that the packets experienced lower queuing delay. On the other hand, as the load increased FAT generated larger bursts and for this reason the packets experienced higher queuing delay.

Figure 6 shows the plot of packet loss ratio (PLR) versus offered load. Both FAHBA and HBA exhibit a similar pattern of loss. However, from low load to medium load ranging from 0.1 to 0.7, FAHBA has lower PLR when compared to HBA because it generated medium bursts that have lower blocking probability. As the load increased from 0.8 to 1, the burst sizes for both algorithms also increased, thereby increasing their

burst blocking probabilities. However, in the case of FAHBA, it was able to maintain the same loss ratio while reducing delay. FAT exhibited an initial low packet loss ratio at low traffic loads. This is due to FAT's initial settings which made it generate small burst. Also FAT's performance at this low load traffic is coupled with the fact that network resources were readily available. However, as the traffic load increased, FAT generated bigger bursts which led to more packet loss when contention occurred.

Figure 7 shows the number of bursts sent into the network and it proves that, for all loads, the number of bursts generated by FAHBA is always higher than that of HBA and FAT except at maximum load of 1. The high number of bursts generated by FAHBA is a result of the size of burst it generated. This performance of FAHBA is attributed to the fact that the algorithm uses fuzzy rules to generate bursts, which results in variable thresholds and burst sizes that satisfy the traffic condition at the assembler. In the case of HBA, burst assembly thresholds are constants. However, at maximum load of 1, the HBA rapidly satisfies its length threshold condition, thereby making it possible to generate more numbers of bursts than FAHBA. As for FAT, it generated bursts of bigger sizes as the load increased. Hence, fewer number of bursts were sent into the network by FAT.

Figures 8 to 11 show the plots of burst end-to-end delay, packet end-to-end delay, packet loss ratio, and the number of bursts that have been generated and sent into the NSFNET. The plot of burst end-to-end delay against offered load is shown in Figure 8. HBA has higher delay from loads 0.1 to 0.5 and this is because of the timer threshold condition of the HBA that was satisfied. As the load increased, the burst sizes also increased, thereby increasing the burst transmission time over the network. But starting from loads 0.6 to 1, the length threshold of the HBA was satisfied. Therefore, fixed-size bursts were generated and injected into the network, thereby causing the network congestion level to increase. As for the FAHBA, the adaptive intelligence of the fuzzy logic controller which is based on fuzzy rules keeps producing suitable threshold values for the assembler. In this case, the timer threshold condition is mostly satisfied in the FAHBA. Hence, in FAHBA, short bursts with shorter transmission times were generated. However, this is not the case in FAT which keeps on increasing the size of bursts as the traffic increases, thereby generating bursts with longer transmission delays.

Figure 9 shows the plot of packet end-to-end delay versus offered load. Packets generated using the HBA algorithm experience higher delay at low loads ranging from 0.1 to 0.4. This situation arises due to the fact that packets must wait for the time threshold condition to be satisfied before any burst can be generated. However, when the load increased from 0.5 to 1, the packets arrive at the assembler so quickly that they satisfy the length threshold condition of the HBA algorithm. Therefore, these packets experience less delay due to the rapid generation of bursts by the HBA. The FAHBA algorithm shows a lower delay because of its ability to satisfy the generation of bursts using either threshold. A similar explanation holds true for FAT as it is discussed for Figure 5.

Figure 10 shows the packet loss ratio versus offered load. At load 0.1, FAHBA has lower PLR when compared to HBA.

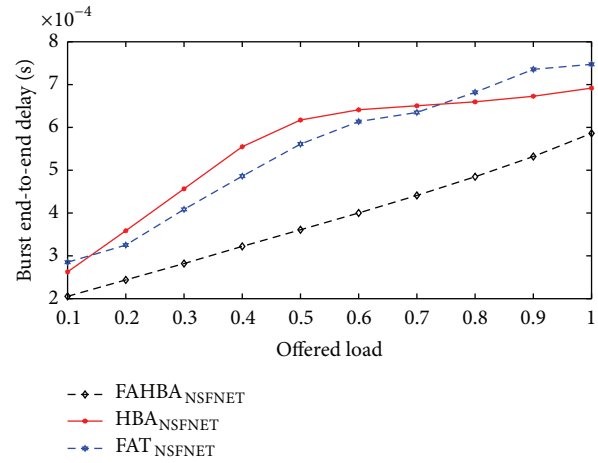



Figure 8: Burst end-to-end delay versus offered load.

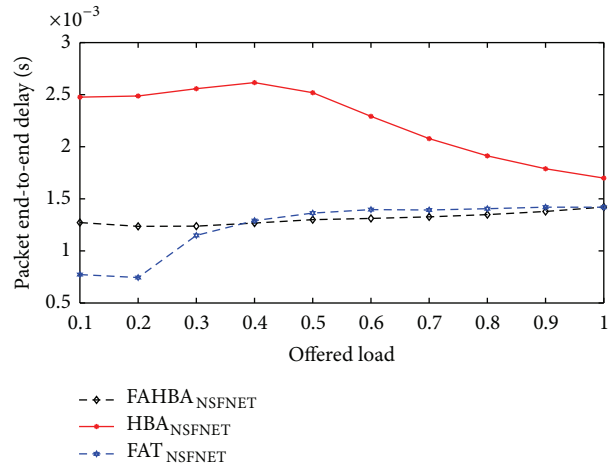



Figure 9: Packet end-to-end delay versus offered load.

And this is because of the small burst sizes that were generated. Both algorithms exhibit a similar pattern of loss when the offered load increased from loads 0.2 to 1. The burst sizes for both algorithms also increased, thereby increasing their burst blocking probabilities. However, in the case of FAHBA, it was able to maintain the same loss ratio while reducing delay. As for FAT, it exhibits lower packet loss ratio under low and medium traffic. However, as the size of bursts generated by FAT grows bigger which is due to the increasing traffic, the network goes into a state of congestion with many bursts contending with each other. This consequently results in high contention and thereby loss.

Figure 11 shows the number of bursts sent into the network by FAT, FAHBA, and HBA algorithms. The figure demonstrates that, for all loads, the number of bursts generated by FAHBA is always higher than those generated by FAT and HBA. The high performance of FAHBA is attributed to the fact that it uses fuzzy logic to adjust the threshold of the underlying burst assembler, thereby generating bursts of suitable sizes that satisfy the traffic condition. FAT generated bigger bursts as the traffic increased and this is the reason why

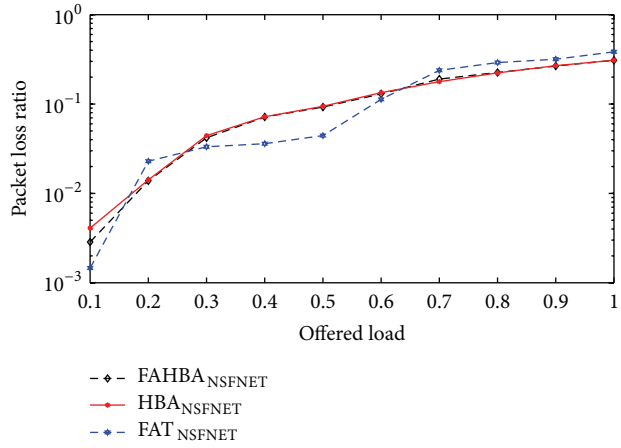



Figure 10: Packet loss ratio versus offered load.

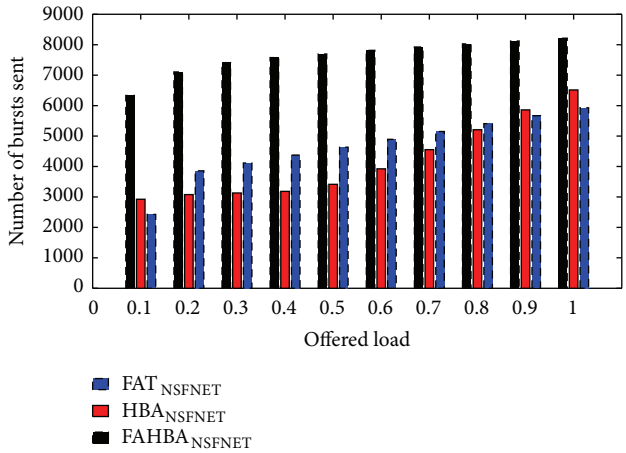



Figure 11: Number of bursts sent versus offered load.

fewer bursts were generated and sent into the network. In the case of HBA, the thresholds are constant.

## 5. Conclusion

In this paper, a new intelligent hybrid burst assembly algorithm for optical burst switched network is proposed. The burst assembly process is modelled as a fuzzy logic control process. Fuzzy logic is used in conjunction with hybrid burst assembly algorithm to minimize end-to-end delay while maintaining packet loss ratio in OBS networks. Through simulation, the proposed algorithm was evaluated against the traditional hybrid and FAT burst assembly algorithms on COST239 and NSF network topologies. In the future, service differentiation will be incorporated in order to improve the new algorithm.

## Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

## Acknowledgments

This research is supported by Universiti Teknologi Malaysia and Ministry of Education, Malaysia, through the Commonwealth Scholarship and Fellowship Plan 2011, and the Fundamental Research Grant Scheme (FRGS:R.J130000.78-28.4F324).

## 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, no. 1, pp. 69–84, 1999.

[2] Y. Chen, C. Qiao, and X. Yu, "Optical burst switching: a new area in optical networking research," *IEEE Network*, vol. 18, no. 3, pp. 16–23, 2004.

[3] J. P. Jue and V. M. Vokkarane, *Optical Burst Switched Networks*, Springer, 1st edition, 2005.

[4] M. Maier, *Optical Switching Networks*, Cambridge University Press, New York, NY, USA, 1st edition, 2008.

[5] A. K. Garg, "An optimal burst assembly approach employing traffic shaping (OBATS) for OBS," *Optik*, vol. 124, no. 22, pp. 5657–5659, 2013.

[6] Z. Zang, "All-optical switching in Sagnac loop mirror containing an ytterbium-doped fiber and fiber Bragg grating," *Applied Optics*, vol. 52, no. 23, pp. 5701–5706, 2013.

[7] Z. Zang and Y. Zhang, "Analysis of optical switching in a $Yb^{3+}$-doped fiber Bragg grating by using self-phase modulation and cross-phase modulation," *Applied Optics*, vol. 51, no. 16, pp. 3424–3430, 2012.

[8] H. Kaur and R. S. Kaler, "Burst assembly and signaling protocols in OBS," in *Proceedings of the National Conference on Challenges and Opportunities in Information Technology (COIT '07) RIMT-IET*, Mandi Gobindgarh, India, 2007.

[9] X. Yu, J. Li, X. Cao, Y. Chen, and C. Qiao, "Traffic statistics and performance evaluation in optical burst switched networks," *Journal of Lightwave Technology*, vol. 22, no. 12, pp. 2722–2738, 2004.

[10] J. Li, C. Qiao, and Y. Chen, "Recent progress in the scheduling algorithms in optical-burst-switched networks [Invited]," *Journal of Optical Networking*, vol. 3, no. 4, pp. 229–241, 2004.

[11] 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*, vol. 26, no. 3, pp. 315–336, 2013.

[12] T. Tachibana and S. Kasahara, "QoS-guaranteed burst transmission for VoIP service over optical burst switching networks," *Journal of Optical Networking*, vol. 6, no. 8, pp. 991–1002, 2007.

[13] A. G. P. Rahbar and O. W. W. Yang, "Contention avoidance and resolution schemes in bufferless all-optical packet-switched networks: a survey," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 4, pp. 94–107, 2008.

[14] S. Askar, G. Zervas, D. K. Hunter, and D. Simeonidou, "A novel ingress node design for video streaming over optical burst switching networks," *Optics Express*, vol. 19, no. 26, pp. B191–B196, 2011.

[15] S.-Y. Oh, H. H. Hong, and M. Kang, "A data burst assembly algorithm in optical burst switching networks," *ETRI Journal*, vol. 24, no. 4, pp. 311–322, 2002.

[16] T. J. Schwartz, "Fuzzy systems in the real world," *AI Expert*, vol. 5, pp. 28–36, 1990.

[17] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller. II," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 2, pp. 419–435, 1990.

[18] J. M. Garibaldi and R. I. John, "Choosing membership functions of linguistic terms," in *Proceedings of the 12th IEEE International conference on Fuzzy Systems (FUZZ '03)*, pp. 578–583, IEEE, May 2003.

[19] A. Ge, F. Callegati, and L. S. Tamil, "On optical burst switching and self-similar traffic," *IEEE Communications Letters*, vol. 4, no. 3, pp. 98–100, 2000.

[20] V. M. Vokkarane, K. Haridoss, and J. P. Jue, "Threshold-based burst assembly policies for QoS support in optical burst-switched networks," in *Proceedings of the Optical Networking and Communications (OptiComn '02)*, pp. 125–136, Boston, Mass, USA, July 2002.

[21] X. Cao, J. Li, Y. Chen, and C. Qiao, "Assembling TCP/IP packets in optical burst switched networks," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '02)*, vol. 3, pp. 2808–2812, November 2002.

[22] 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*, pp. 250–256, 2003.

[23] Z. Zhang, F. Cheng, J. Wang, J. Luo, Q. Zeng, and X. Xuan, "A new burst assembly and dropping scheme for service differentiation in optical burst switched networks," in *Proceedings of the Asia-Pacific Optical and Wireless Communications: Optical Transmission, Switching, and Subsystems (APOC '03)*, pp. 236–245, Wuhan, China, November 2003.

[24] B. Kantarci and S. Oktug, "Adaptive threshold based burst assembly in OBS networks," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE '06)*, pp. 1419–1422, Ottawa, Canada, May 2006.

[25] A. Gupta, R. S. Kaler, and H. Singh, "Investigation of OBS assembly technique based on various scheduling techniques for maximizing throughput," *Optik*, vol. 124, no. 9, pp. 840–844, 2013.

[26] H.-l. Liu and S. Jiang, "A mixed-length and time threshold burst assembly algorithm based on traffic prediction in OBS network," *International Journal of Computers Communications & Control*, vol. 2, pp. 87–93, 2012.

[27] 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, no. 2, pp. 132–148, 2013.

[28] J.-R. Yang, S.-I. Jia, and G. Wang, "Burst assembly algorithm based on fuzzy-adaptive-threshold," *Journal of Harbin Engineering University*, vol. 28, no. 6, 2007.

[29] 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 *Proceedings of the Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC '11)*, pp. 1–3, March 2011.

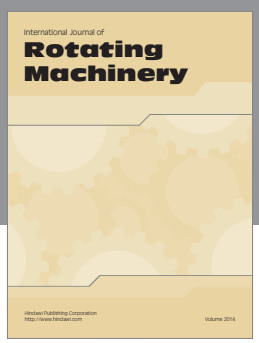
[30] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, p. 60, 2008.

[31] F. Espina, J. Armendariz, N. Garc, D. Morat, M. Izal, and E. Maga, "OBS network model for OMNeT++: a performance evaluation," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, Malaga, Spain, 2010.

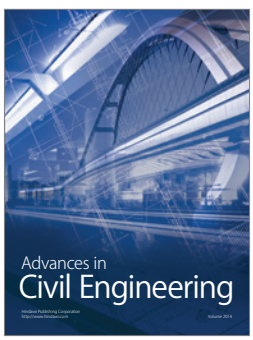
[32] J. Rada-Vilela, "fuzzylite: A fuzzy logic control library written in C++," 2013, http://www.fuzzylite.com/.

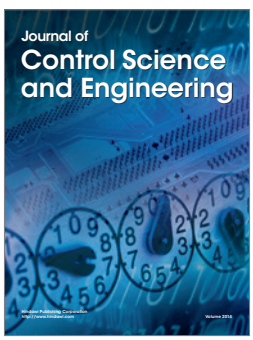
[33] P. Lenkiewicz, M. Hajduczenia, M. M. Freire, H. J. Da Silva, and P. P. Monteiro, "Estimating network offered load for optical burst switching networks," in *NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, vol. 3976 of *Lecture Notes in Computer Science*, pp. 1062–1073, Springer, Berlin, Germany, 2006.









Hindawi
Submit your manuscripts at
http://www.hindawi.com