International Journal of Innovative
Computing, Information and Control
Volume **10**, Number **5**, October **2014**

ICIC International ©2014 ISSN 1349-4198
pp. 1669–1685

TOWARDS INTELLIGENT DATACENTER TRAFFIC MANAGEMENT: USING AUTOMATED FUZZY INFERENCING FOR ELEPHANT FLOW DETECTION

Manh Tung Pham[1], Kiam Tian Seow[1] and Chuan Heng Foh[2]

[1]School of Computer Engineering
Nanyang Technological University
No. 50, Nanyang Avenue, Singapore 639798, Republic of Singapore
{ pham0028; asktseow }@ntu.edu.sg

[2]Centre for Communication Systems Research
University of Surrey
Guildford, Surrey, GU2 7XH, United Kingdom
c.foh@surrey.ac.uk

Received August 2013; revised January 2014

Abstract. *Effective traffic management has always been one of the key considerations in datacenter design. It plays an even more important role today in the face of increasingly widespread deployment of communication intensive applications and cloud-based services, as well as the adoption of multipath datacenter topologies to cope with the enormous bandwidth requirements arising from those applications and services. Of central importance in traffic management for multipath datacenters is the problem of timely detection of elephant flows – flows that carry huge amount of data – so that the best paths can be selected for these flows, which otherwise might cause serious network congestion. In this paper, we propose FuzzyDetec, a novel control architecture for the adaptive detection of elephant flows in multipath datacenters based on fuzzy logic. We develop, perhaps for the first time, a close loop elephant flow detection framework with an automated fuzzy inference module that can continually compute an appropriate threshold for elephant flow detection based on current information feedback from the network. The novelty and practical significance of the idea lie in allowing multiple imprecise and possibly conflicting criteria to be incorporated into the elephant flow detection process, through simple fuzzy rules emulating human expertise in elephant flow threshold classification. The proposed approach is simple, intuitive and easily extensible, providing a promising direction towards intelligent datacenter traffic management for autonomous high performance datacenter networks. Simulation results show that, in comparison with an existing state-of-the-art elephant flow detection framework, our proposed approach can provide considerable throughput improvements in datacenter network routing.*
Keywords: Datacenters, Traffic management, Fuzzy logic

1. **Introduction.** Endowed with a precise mathematical formalism to accommodate uncertainty or resolve conflict arising from impreciseness and ambiguity, fuzzy logic enables approximate reasoning [1, 2, 3] that entails drawing inferences from fuzzy rules emulating human judgments based on domain expert knowledge. This human-like reasoning capability makes it particularly attractive for addressing engineering application problems, where it is impractical or impossible to precisely assess a situation in which the information accessible for quality decision making is inherently ambiguous or conflicting. Applications of fuzzy logic include medical decision supports [4, 5, 6], bioinformatics [7], finance [8, 9] and planning and management [10, 11]. In this paper, we present a novel application of

fuzzy logic to intelligent traffic management of datacenter networks – a problem of central importance in today's ubiquitous cloud computing environments.

In coping with the increasingly enormous bandwidth demands arising from scientific computing [12], communication intensive cloud-based services [13] and web search engine applications [14], new datacenter network designs with multiple paths between pairs of sources and destinations have been proposed to replace conventional hierarchical tree topologies [15]. The basic premise is that if flows can be distributed proportionally among available paths then hosts can communicate with other hosts at the maximum speed of their network interface cards (NICs), maximizing the aggregate bisection bandwidth [16]. Newly proposed architectures, such as Fat-tree [16], HyperX [17] and Flattened Butterfly [18], have been shown to provide much higher aggregate bisection bandwidth than conventional tree architectures, provided fine-grained routing techniques are employed for load balancing and distributing flows among available paths [19].

Until recently, static routing techniques, of which the most popular one is ECMP (Equal Cost Multipath) [20], are employed in multipath datacenters for load balancing. ECMP is an oblivious routing technique, which distributes flows using only flow hashing. It is simple to implement, requires low computational effort, and can deliver high aggregate bisection bandwidth when flows are uniform in size [21]. However, using ECMP, problems arise when flow sizes vary, which is often the case in datacenter traffics [22]. In this case, a relatively small number of elephant flows – flows that carry huge amount of data – often carry a large fraction of datacenter traffics [22]. Without considering flow size and the current network utilization, ECMP may inadvertently place two elephant flows onto the same congested path in the network, creating unnecessary long-lived collision.

To remedy ECMP but still retain its useful features, an intuitive approach is to only use EMCP to route non-elephant flows. Elephant flows would then need to be detected as soon as possible upon entering the network, and routed dynamically based on the current network utilization. In implementing this approach, however, a challenging problem is how to detect elephant flows in a timely manner. Detecting elephant flows too late may create unnecessary network congestion since ECMP might have already been inadvertently used to route them. This problem is further exacerbated by the lack of preciseness in the definition of elephant flows, namely, how big and when is a flow size considered "elephant"? As a simple illustration, if a network has a current bandwidth capacity of 100 Gbps, a 1 GB flow might be safely classified as "non-elephant", but if the same network has a bandwidth capacity of only 1 Gbps remaining, possibly due to high network utilization, a 1 GB flow should be better treated as "elephant".

Against this background, this paper develops a simple, yet effective fuzzy inference architecture called FuzzyDetec to tackle the challenging problem of timely detection of elephant flows. Through an innovative application of fuzzy logic, we demonstrate the role of incorporating human expertise and judgments as fuzzy rules to handle the impreciseness of flow size classification for elephant flow detection, and propose a close loop control framework for the adaptive detection of elephant flows. The proposed approach is simple, intuitive and easily extensible, providing a promising direction towards intelligent datacenter traffic management for autonomous high performance datacenter networks. Importantly, over Mahout [23], an existing state-of-the-art elephant flow detection framework, we experimentally show that FuzzyDetec can provide considerable throughput improvements in datacenter network routing.

To the best of our knowledge, this paper presents perhaps the first attempt that applies fuzzy logic reasoning to intelligent traffic management of datacenter networks. Apart from our work, to date, fuzzy logic has surprisingly found little application in the field

of datacenter networks, other than managing virtual computing resources in datacenters [24, 25] and selecting one from multiple datacenters to service applications [26].

The rest of the paper is organized as follows. Section 2 presents the motivation and main contributions of our work. In Section 3, we review relevant background and related work in datacenter network topologies, routing and traffic management. We then present the design of FuzzyDetec in Section 4. In laying a clear and uncluttered research foundation for FuzzyDetec, only the most basic and fundamental concepts and techniques in fuzzification, fuzzy inference and defuzzification are presented in this paper. To make this paper self-contained, these concepts and techniques are briefly reviewed in Section 4 as necessary. In Section 5, we experimentally evaluate FuzzyDetec. Finally, Section 6 concludes the paper.

2. Motivation and Contributions.

Our research is performed within the increasingly popular model of simple-switch/smart-controller datacenter, as proposed in the OpenFlow framework [27]. Within this model, non-elephant flows are handled by the state-of-the-art routing mechanism using ECMP modules implemented at every switch in the network, while notifications of elephant flow detections are directed to a center controller, which dynamically computes the best paths for these flows based on the current network utilization. In order to do so, the center controller is connected to all switches in the network and can poll statistics to estimate buffer and link utilizations as well as receive elephant flow notification.

Existing approaches in detecting elephant flows can be broadly classified as using one of the following strategies:

1. Modify and enable applications to identify whether their flows are elephants [28].
2. Use switches to maintain per-flow statistics, and the center controller to periodically poll these statistics and identify a flow as elephant when some statistical conditions are met [21].
3. Use the center controller to sample packets from individual switches and identify a flow as elephant after it has seen a sufficient number of packet samples from the flow [29].
4. Use every end-host to monitor the number of bytes in its buffer for every flow and identify a flow as elephant as soon as the number of bytes in its buffer is greater than a threshold that is set [23].

Modifying existing applications and forcing new applications to incorporate features to detect elephant flows, as proposed in the first strategy, are often impractical [23]. Polling per-flow statistics and sampling packets from switches, as proposed in the second and third strategy, respectively, incur high monitoring cost for the center controller, consume significant network communication bandwidth and switch resources, and might require long detection time. An elephant flow detection architecture called Mahout [23] has been developed to implement the fourth strategy. Compared with the rest, the work is promising for its demonstrated merits of shorter detection time, low monitoring cost and low consumption of switch resources, laying a good foundation for datacenter traffic management. However, Mahout is essentially an open loop architecture, without an automated decision-making module that can continually compute an appropriate threshold for elephant flow detection based on information feedback from the network. This is a key limitation of Mahout for continuous datacenter operation. Besides, threshold setting faces a dilemma: on the one hand, setting a threshold that is too low will cause too many flows to be recognized as elephants, overloading the center controller with tasks of computing paths for these flows; on the other hand, setting a threshold that is too high will cause too

many flows to be recognized as non-elephants, possibly creating long-lived but actually avoidable collisions in the network when only ECMP is deployed to distribute them.

We assert that a threshold for use in elephant flow detection algorithms should be dynamically set, based on important criteria such as current network utilization and center controller load. If network utilization is already high, for example, the cost incurred for dynamically routing elephant flows might exceed the benefits gained since there might be no path that could accommodate the flows. In this case, a large threshold value should be set, so that less flows are identified as elephant flows. Similarly, if the center controller is lightly loaded, it should consider more flows for routing instructions, and a small threshold value should be set.

Without a suitable formalism, incorporating the abovementioned criteria in threshold setting for elephant flow detection is difficult. This is because, being based mostly on human expertise and judgments, firstly, describing the problem in terms of the criteria and threshold decisions to take are imprecise or vague. Such vagueness of description arises in general due to fuzziness [1] in the semantic meanings of events and phenomena. For example, statements such as "current network utilization is high", "current controller load is low", "set a large elephant flow detection threshold" and "this flow is elephant" are vague because the meanings of high, low, large and elephant (or non-elephant) are not precise and depend on context. In datacenter operations, this context is characterized by the bisection bandwidth capability of the datacenter network, the computational capability of the center controller and the characteristics of the datacenter traffic patterns. Secondly, the criteria could be conflicting with each other. For example, when network utilization is high and controller load is low, it is hard to decide whether a low or a high threshold value should be set, since the former condition implies a high threshold setting while the latter implies a low threshold setting.

In an application context where criteria cannot be sharply defined, fuzzy logic provides a powerful conceptual formalism for reasoning, learning and decision making. The theory has been demonstrated to be effective in handling multiple conflicting criteria as well as their vagueness in a natural way [1]. In this paper, we propose a fuzzy logic based architecture called FuzzyDetec. The architecture closes the elephant flow detection loop in Mahout with a fuzzy logic inference module resident in the center controller. The module reasons based on simple fuzzy rules incorporating datacenter network criteria in a manner emulating human expertise, and computes appropriate elephant flow threshold decisions in accordance to information feedback on current network conditions. This threshold, which is communicated by the center controller to every end-host for elephant flow detection, is an aggregation of various criteria related to the current network environment. As an illustration, we present and justify several rules for setting a threshold value based on current network utilization and current controller load, and evaluate our approach with extensive simulations.

In equipping the center controller with a fuzzy logic inference module for automatically computing appropriate thresholds for elephant flow detection, we overcome Mahout's key limitation of an open loop architecture that only allows some pre-determined, static value for the elephant flow threshold. Closing the elephant flow detection loop, as in FuzzyDetec, is an essential step towards fully automating datacenter network operations. And, to the best of our knowledge, this work is the first attempt on classifying flows based on current datacenter network conditions. Since different treatments of elephant flows are certainly needed for different network conditions, we believe this is a promising direction for improving the performance of datacenters. Importantly, the proposed approach of using fuzzy logic based elephant flow detection is simple and intuitive, allows multiple criteria to be flexibly incorporated in the detection process, and is easily extensible.

3. Background and Related Work.

3.1. Multipath topologies.

Traditionally, datacenter topologies are hierarchical trees with a layer of racks of hosts at the bottom, with all hosts in a rack connecting directly to a Top of Rack (ToR) switch, and a layer of core switches at the top. ToR switches are connected to aggregation switches, and these switches are aggregated further, connecting to core switches [15]. Moving up the hierarchy, ToR switches are the smallest and cheapest with the lowest speed, while core switches are the densest in port numbers and most expensive, and have the highest speed (see Figure 1).

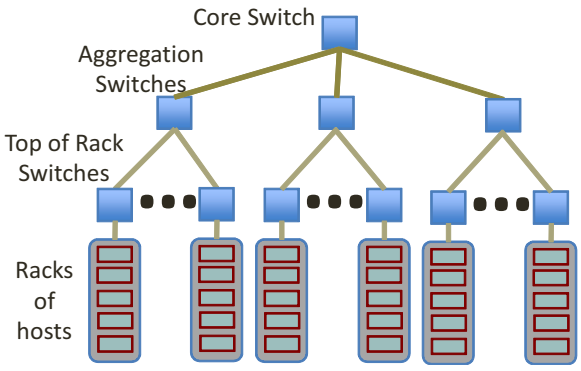


FIGURE 1. A conventional network topology for datacenters

Due to the high costs of port-dense and high speed switches, the over-subscription ratio[1] increases rapidly as we go up the hierarchy. At the bottom level, hosts typically have 1:1 over-subscription to other hosts in the same rack, allowing them to communicate intra-rack at the maximum speed of their NICs. However, up-links from ToRs over-subscription ratios are typically 1:5 to 1:20, and paths through the core switches can be 1:240 over-subscribed [30]. These large over-subscription ratios severely limit the communication bandwidth between hosts in different racks [30].

With the increasing deployment of communication intensive applications and cloud-based services in datacenters, new topologies, such as Fat-tree [16], HyperX [17] and Flattened Butterfly [18], are proposed to cope with the enormous bandwidth demands. Fat-tree topology [16], for example, enables commodity switches to be connected in a manner that maintains 1:1 over-subscription across the whole network, allowing hosts to potentially communicate with other arbitrary hosts at the maximum speed of their NICs (see Figure 2).

One common feature of the newly proposed datacenter topologies is that they are all designed with multiple data paths with equal length between every pair of source and destination switches, providing the possibility for managing network congestion and maximizing aggregate bisection bandwidth. To extract the best aggregate bandwidth from these multipath topologies, fine-grained routing techniques must be designed for load balancing, namely, distributing flows among available paths so that none of these paths are overloaded while others are underloaded. Traditional shortest path routing protocols, such as OSPF [31], are not suitable for this purpose since they might concentrate traffics going to a given destination in a single port of an aggregation switch and a single port of a core switch, even though other choices exist, causing avoidable network congestion [21].

[1] Over-subscription ratio is the ratio of the worst-case achievable aggregate bandwidth among end-hosts to the total bisection bandwidth of a particular network topology.

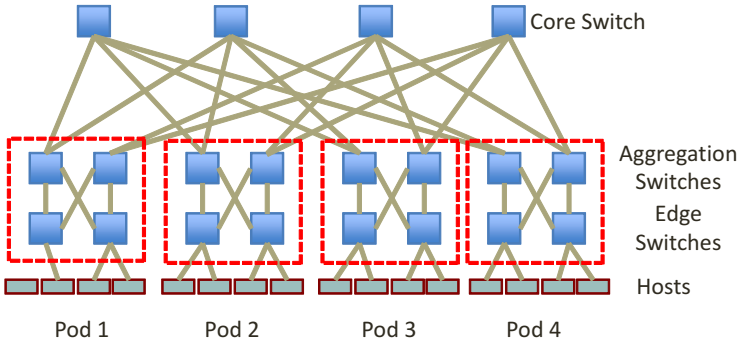


FIGURE 2. A 4-array fat-tree topology for datacenters. In a k-array fat-tree network there are $5k^2/4$ identical k-port switches, with $k^2/4$ core switches and the remaining k^2 switches are organized into k pods, each of which contains one layer of $k/2$ aggregation switches and one layer of $k/2$ edge switches. In each pod, each k-port edge switch is connected to $k/2$ hosts and $k/2$ aggregation switches in the same pod. The remaining $k/2$ ports of each aggregation switch is connected to $k/2$ core switches in an order similar to the one shown in the figure. By scarifying compact wiring, a fat-tree network ensures that the number of input links to any switch is equal to the number of links going out of it, thereby maintaining 1:1 over-subscription ratio across the whole network and allowing hosts to potentially communicate with arbitrary other hosts at the full speed of their NICs. See [16] for a detailed discussion of fat-tree topology.

3.2. **Static vs. dynamic routing.** In one extreme, load balancing can be done with minimal computational efforts using ECMP [20], a static, oblivious routing technique. In essence, an ECMP-enabled switch is configured with multiple paths for a given destination. When a packet with multiple candidate paths arrives, it is forwarded to a path that corresponds to a hash of selected fields in the packet header, modulo the number of paths. In this way, flows between a pair of source and destination switches are split across multiple paths. ECMP works well when flows are small and uniform in size [21]. When elephant flows are present, however, its key limitation is that two elephant flows might collide in their hash and route through the same path in the network, resulting in long-lived congestion.

In the other extreme, load balancing can be done in a completely dynamic fashion, with every new flow placed in the best path selected online by a routing algorithm, taking into account the current utilization of every link and buffer in the network. Such a routing algorithm is often run by a center controller, which is connected to all switches and can poll statistics from them to estimate the utilization of every link and buffer. This approach, however, requires high computational efforts and introduces unacceptable setup delays for latency-sensitive flows, such as those generated from interactive applications [21, 23].

Modern datacenter routing mechanisms for multipath topologies often combine both static and dynamic routings: Paths for elephant flows are computed online, while paths for non-elephant flows are selected using ECMP. It has been shown [21] that by detecting and dynamically placing elephant flows in carefully selected paths based on current network utilization, as much as 113 % higher aggregate throughput can be achieved as compared to using ECMP alone.

3.3. **Elephant flow detection.** Of central importance to effectively select and deploy routing algorithms for multipath topologies is the problem of timely detection of elephant

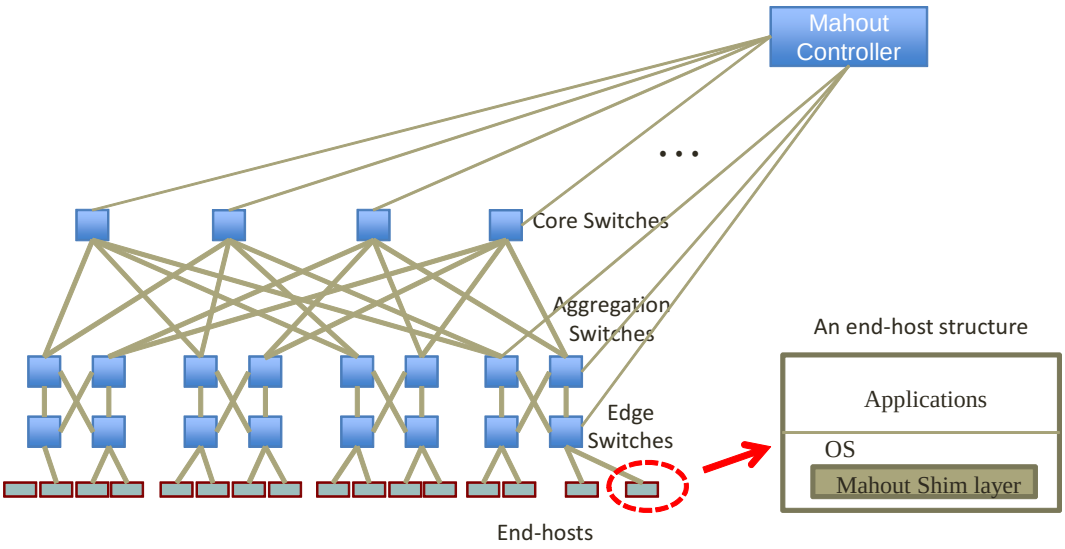


FIGURE 3. Mahout – An end-host based elephant flow detection system

flows. Curtis et al. [23] propose Mahout, an architecture for end-host based elephant flow detection which has proven to be superior to previous approaches [21, 29] in terms of early detection and low computational overload. The Mahout architecture, which subscribes to the popular simple switch/smart controller model as proposed in OpenFlow [27], is shown in Figure 3. In this architecture, end-hosts are responsible for the timely detection of elephant flows and a center controller connecting to every switch in the network is responsible for computing the best paths for newly detected elephant flows. An end-host detects elephant flows through what is called a shim layer implemented in its network stack, enabling the host to monitor the socket buffer of every flow originating from it. An end-host will identify a flow as an elephant as soon as the number of bytes in its buffer is greater than a threshold. Once an end-host detects an elephant flow, it sets the Differentiated Services (DS) Field bits of every packet in this flow to 00001100, to inform the edge switch that is directly connected to it (see Algorithm 1). The edge switch notifies the center controller of the arrival of a new elephant flow, which in turn computes the best path for this elephant flow, and installs flow-specific routing entry to every necessary switch in the network.

Algorithm 1: Pseudo-code for end-host shim layer

```
begin
    When sending a packet, if the number of bytes in buffer ≥ Threshold then
        Mark the packet as belonging to an elephant flow by setting its DS bit to 00001100;
```

However, a key limitation of Mahout for continuous datacenter operation is that it is essentially an open loop architecture, without an automated decision-making module that can continually compute an appropriate threshold for elephant flow detection based on information feedback from the network. Besides, threshold setting faces a dilemma: on the one hand, setting a threshold that is too low will cause too many flows to be recognized as elephants, overloading the center controller; on the other hand, setting a threshold that is too high will cause too many flows to be recognized as non-elephants, resulting possibly in long-lived but actually avoidable collisions in the network.

4. **Fuzzy Logic Based Elephant Flow Detection.** The proposed FuzzyDetec architecture is shown in Figure 4. In FuzzyDetec, the center controller is equipped with a fuzzy logic inference module that computes appropriate values for the elephant flow threshold based on current network conditions. It then communicates the computed value to every end-host in the datacenter which in turn inputs the new threshold value to Algorithm 1 for elephant flow detection. In equipping the center controller with a fuzzy logic inference module for automatic elephant flow threshold computation based on information feedback on current network conditions, we close the control loop in Mahout, an essential step towards fully automating datacenter network operations.

In the following, we describe the design of the fuzzy logic inference module. We present how $Threshold$ can be determined based on the current network utilization $curNetUtil$, and the current controller load $curCtrload$. The approach can easily be extended to incorporate new criteria other than network utilization and controller load.

4.1. **Linguistic variables and membership functions.** Ordinary Boolean logic deals with exact reasoning where a variable can only take a value of either true or false. Fuzzy logic is an extension of Boolean logic to deal with approximate reasoning. This is achieved by introducing linguistic variables that can take numerical values, and associating each of these variables with a collection of linguistic values. A linguistic value represents what is called a linguistic set containing a subrange of the numerical values defined for a linguistic variable. Every linguistic value of a linguistic variable is associated with a membership function that takes values between 0 and 1, denoting the degree that the linguistic value represents every specific numerical value in the linguistic set (that it represents) for the variable.

In FuzzyDetec, there are two input linguistic variables $curNetUtil$ and $curCtrload$, and one output linguistic variable $Threshold$. The objective of FuzzyDetec is to deduce a numerical value for $Threshold$ from the numerical values of $curNetUtil$ and $curCtrload$. The numerical values of $curNetUtil$ and $curCtrload$ are provided by the center controller and the numerical value of $Threshold$ is deduced by fuzzy inferencing (see Figure 4).

The linguistic variables $curNetUtil$ and $curCtrload$ can take linguistic values of either LOW, MEDIUM or HIGH, and $Threshold$ can take linguistic values of either SMALL, MEDIUM or LARGE. We define the network utilization $curNetUtil$ as the average of all link utilizations in the datacenter. The current controller load $curCtrload$ is estimated by the current number of elephant flows that the center controller has to compute routing

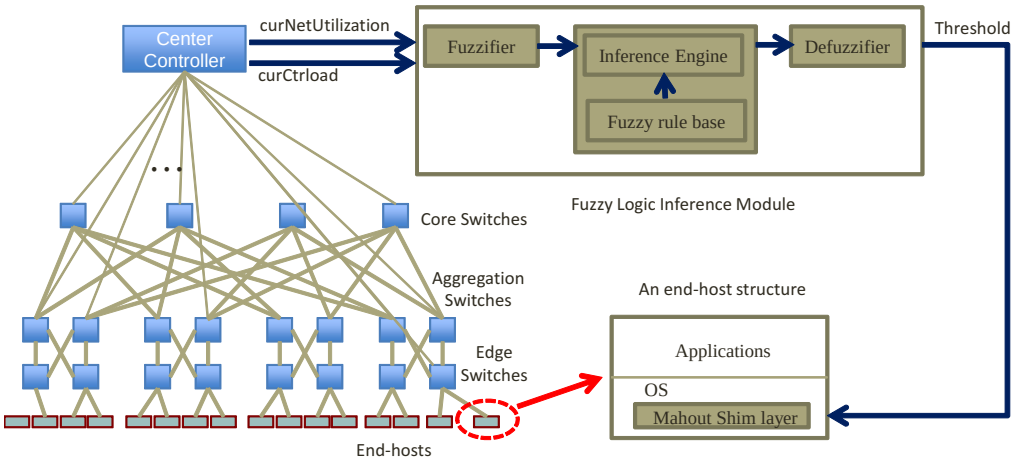
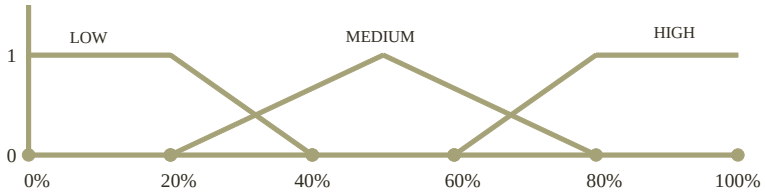
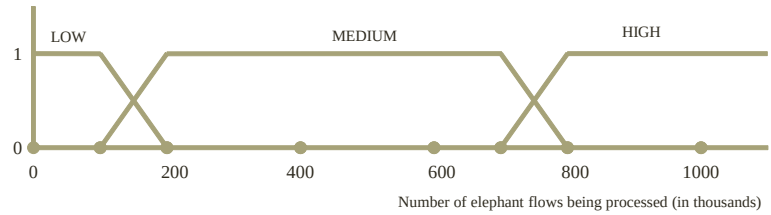


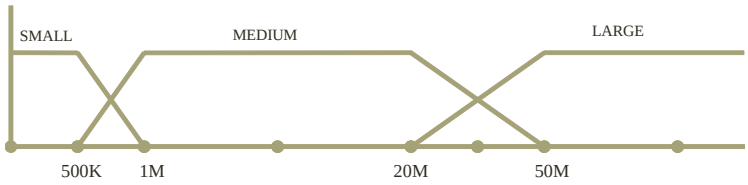
FIGURE 4. FuzzyDetec – A fuzzy logic based elephant flow detection system



(a) Membership functions for linguistic values of $curNetUtil$



(b) Membership functions for linguistic values of $curCtrload$



(c) Membership functions for linguistic values of $Threshold$

FIGURE 5. Illustration of membership functions for fuzzy values in FuzzyDetec

paths for. An example of membership functions for the linguistic values of $curNetUtil$, $curCtrload$ and $Threshold$ is shown in Figure 5.

The shape of a membership function emulates human expertise in a particular application context. Membership functions can often be constructed by addressing questions such as the following: To what degree is a 20% network utilization considered low and medium? To what degree is a 60% network utilization considered medium and high? By answering these questions, pairs of numerical values and the degrees that these values are represented by each linguistic value (low, medium or high) are defined, forming the respective linguistic sets and corresponding membership functions. The membership functions can be constructed using curve-fitting methods such as trapezoidal approximation. In fuzzy logic applications, it is a common practice to use trapezoidal and triangular shapes for membership functions due to their computational efficiency, such as those presented in Figure 5; but other shapes can also be used [1].

4.2. **Fuzzification and fuzzy rule base.** The center controller in FuzzyDetec collects statistics from switches in the datacenter and estimates numerical values of $curNetUtil$ and $curCtrload$. These values are then converted to linguistic sets in a process called fuzzification. This is done by a procedure called fuzzifier (see Figure 4) that takes the numerical value of a linguistic variable and converts it to a collection of linguistic sets using the membership functions associated with the linguistic variable. The threshold setting strategy is expressed in terms of a set of if-then rules that maps linguistic variables to linguistic variables. These rules are normally constructed by datacenter experts, incorporating their experiences on classification of elephant flow threshold as small, medium or large. As an illustration, we present in Table 1 concise threshold classification rules that essentially map the linguistic values of $curNetUtil$ and $curCtrload$ to those of $Threshold$.

TABLE 1. The rule base for threshold classification

$curNetUtil$	$curCtrload$	$Threshold$
Low	Low	Small
	Medium	Small
	High	Medium
Medium	Low	Medium
	Medium	Medium
	High	Large
High	Low	Medium
	Medium	Large
	High	Large

Intuitively, when the network utilization is already high, it is often hard to find a path that could accommodate elephant flows. Therefore, we would want to set a large threshold, so that less flows are identified as elephant. Similarly, if the center controller load is high, less flows should be directed to it for routing instructions, hence a large elephant flow threshold should be set. This intuition is expressed by the rule in the last row of Table 1, namely, if $curNetUtil$ is HIGH and $curCtrload$ is HIGH, $Threshold$ is LARGE. In the other extreme and by the same intuition, if the network utilization is low and the center controller load is low, a small threshold should be set, namely, if $curNetUtil$ is LOW and $curCtrload$ is LOW, $Threshold$ is SMALL, as expressed by the rule in the first row of Table 1.

Other rules in Table 1 can be interpreted similarly. For example, the rule in the third row of Table 1 states that if $curNetUtil$ is LOW and $curCtrload$ is HIGH, $Threshold$ is MEDIUM. This rule attempts to balance the two conflicting conditions of requiring a small threshold setting on the one hand, as prescribed by a low network utilization, and of requiring a large threshold setting on the other, as prescribed by a high controller load.

4.3. **Fuzzy inference.** The membership functions associated with linguistic values in a fuzzy inference system normally overlap. Therefore, several rules typically contribute to determining the linguistic value of an output linguistic variable. The result of executing each if-then rule is a fuzzy linguistic set, which is represented by a membership function derived from membership functions (for the linguistic values) of the linguistic variables specified in the if-then rule. Fuzzy linguistic sets need to be aggregated into a final linguistic set through a process called fuzzy inference. This is done by aggregating membership functions of individual linguistic sets to arrive at a final membership function representing the final linguistic set. This final linguistic set is then mapped into a specific numerical value of the output variable by applying an operator called "defuzzification" on its membership function (see Figure 4).

Key to the fuzzy inference process is how linguistic sets of fuzzy if-then rules are determined and aggregated. In fuzzy set theory [1], there are altogether eight important considerations for designing the output linguistic sets of fuzzy if-then rules and the aggregation operators. These considerations have led to the development of standard ways to implement the fuzzy inference process. A detailed discussion of this development is, however, outside the scope of this paper.

Let x, y, z be three linguistic variables and A, B, C be three linguistic values associated with them in order. In the following, we briefly describe how the fuzzy rule "if x is A and y is B then z is C" is normally realized in fuzzy logic applications.

Let μ_A, μ_B, μ_C be membership functions associated with the linguistic values A, B and C, and x_0 and y_0 be numerical values of x and y, respectively. The fuzzy statement "if x is A and y is B" is often realized by a minimum operator on the membership functions, namely, it is interpreted as the crisp value $Z^0 = \min(\mu_A(x_0), \mu_B(y_0))$. The linguistic set of the rule "if x is A and y is B then z is C" is then defined by the membership function $\mu_{C'}(z) = \min(Z^0, \mu_C(z))$.

If there are several rules, each rule is evaluated individually using the minimum operator as described above, and their results are aggregated using a maximum operator on the membership functions obtained for the individual rules. The overall result is a membership function representing the final linguistic set, aggregating individual linguistic sets.

The inference procedure described is called "min-max", referring to operators applied on membership functions. Other inference procedures can also be used. For example, the algebraic product operator can be used in place of the min operator and the algebraic sum operator can be used in place of the max operator, resulting in a "product-sum" inference procedure. Combinations of these operations, such as "min-sum" and "product-max", are also possible. In FuzzyDetec, we implement the fuzzy inference process using the "min-max" procedure.

4.4. **Defuzzification.** Fuzzy inference in FuzzyDetec results in a linguistic set of the linguistic variable *Threshold*. This linguistic set is represented by a membership function. To compute a numerical value of the elephant flow threshold for the end-hosts, the linguistic set obtained must be converted to a numerical value. This is done by applying a "defuzzification" operator on a linguistic set of *Threshold*. Defuzzification methods are often developed based on heuristic ideas, such as "taking the value that corresponds to the maximum membership" or "taking the value that is at the center of two peaks". They can also be derived through rigorous analysis processes, such as fuzzy linear programming or multi criteria analysis [1].

In the literature, the most common method used in defuzzification is the Center of Gravity method. This method computes a numerical value for the threshold from a linguistic set $\bar{T}$ as follows.

$$threshold = \frac{\int \tau \mu_{\bar{T}}(\tau) d\tau}{\int \mu_{\bar{T}}(\tau) \tau}$$

where $\mu_{\bar{T}}$ is the membership function representing the linguistic set $\bar{T}$.

Other methods for defuzzification include the "Max Criterion" method, the "Mean of Maxima" method and the "Bisector of Area" method [1]. In comparison with other methods, the Center of Gravity method has one important advantage of guaranteeing that every if-then rule contributes to determining the final output. In this paper, we use the Center of Gravity method for defuzzification in FuzzyDetec.

4.5. **Discussion.**

4.5.1. *Dynamic routing algorithms.* Once presented with a set of elephant flows, there are several ways for the center controller to compute and allocate paths to these flows. Correa and Goemans [32] propose an increasing first fit algorithm, which allocates the least congested path among possible paths to elephant flows, one at a time in a decreasing order of flow rates. Al-Fares et al. [21] propose two algorithms for computing paths for elephant flows: one greedily assigns a flow to the first path that can accommodate it and the other is based on Simulated Annealing [33], a population-based, probabilistic search algorithm. Either of the algorithms can be incorporated into FuzzyDetec for dynamically computing paths for detected elephant flows. In a concrete implementation of FuzzyDetec,

for the purpose of comparison with Mahout, we chose to implement the same algorithm as used in Mahout, namely, the increasing first fit algorithm proposed in [32].

4.5.2. *Threshold communication.* The elephant flow detection threshold is updated continually in response to changes in the values of the network utilization and center controller load. If every updated value is immediately communicated to the end-hosts, bandwidth resources might be consumed unnecessarily. To mitigate this problem, the center controller in FuzzyDetec only informs the end-hosts whenever there is a significant change in the threshold value, such as a change of 10% or more. An alternative is to use in-band signaling by piggybacking updated threshold values in OpenFlow protocol messages that the center controller frequently sends to individual switches. Further study is needed to investigate this alternative.

5. **Performance Study.** We now experimentally compare the performance of FuzzyDetec with that of Mahout [23] to show the effectiveness of FuzzyDetec in throughput improvement for datacenter traffic routing. In the following we describe the design of our simulator, the methodology used to generate simulated traffics and simulation results.

5.1. **Simulator.** For the same time consuming reason given in [21], it is inefficient to use existing packet level simulators, such as ns-2, ns-3 or OMNET, for our performance study. As described therein, for a moderate size fat-tree network of 8,192 hosts, each sending at 1Gbps, it would take around 71 hours to simulate just one test case of a 60-second datacenter run.

We designed and implemented an event-based flow level simulator. In our design, a datacenter network was modeled by a directed graph, with hosts and switches represented by nodes in the graph and links represented by capacitated edges. All experiments were run on directed graphs modeling a fat-tree network of 8192 hosts, with all links having capacity of 1Gbps.

For simplicity and without bias effect on the network throughput performance, we simulated ECMP by configuring every switch with multiple paths for every destination, and when a flow with multiple candidate paths arrives at a switch, we forwared the flow to a path selected randomly from the candidate paths.

Whenever a flow starts or completes, the simulator recomputes the rate of each flow. Similar to the idea described in [21], the rate is regulated by TCP.

Like [23], we model the OpenFlow protocol by only accounting for the time delay that a switch incurs in notifying the center controller of a new elephant flow and receiving a routing table entry setup from the center controller in turn. This is done by setting up a 10Mbps link (a typical bandwidth of OpenFlow links) between the center controller with every switch in the network and assuming that a notification message of 1500 bytes (the maximum transfer unit of Ethernet) is sent from a switch to the controller on this link whenever the switch needs to inform the center controller of the presence of a new elephant flow. A routing table entry setup message of 1500 byptes is then sent back from the center controller to the switch on the same link. Other times consumed in computing paths for elephant flows and setting up routing table entries are ignored. As a result, it is likely that an OpenFlow datacenter implementing our FuzzyDetec module will perform worse than predicted by our simulator in terms of throughput. However, this reduction in throughput performance is mostly attributed to the detailed implementation of the OpenFlow protocol and dynamic routing algorithms used by the center controller, and not the design of our fuzzy logic inference module.

5.2. **Traffic patterns.** In the absence of data on commercial datacenter network traces, the nature of datacenter traffics is not completely understood. As a result, generating simulated traffics to evaluate the performance of datacenters has always been challenging. In the literature, traffics are often generated from either synthetic communication patterns, or application-specific communication patterns such as those generated by MapReduce or scientific computing applications. However, both synthetic and simulated application-specific communication patterns are not guaranteed to be representative of datacenter traffics, which are often composed of traffics from a variety of distributed applications [34].

Recent work [22, 30] has attempted to empirically capture both macroscopic and microscopic measurements of datacenter traffics. The former specifies which hosts communicate with which other hosts and when, while the latter characterizes the distributions of datacenter flows' statistics such as durations, sizes, and inter-arrival times. At the macroscopic level, Kanula et al. [22] observe that within a rack, a host communicates either with all other hosts or less than 25% of the total number of hosts. Furthermore, a host either does not communicate with other hosts in different racks, or communicate with 1-10% of them. At the microscopic level, it is reported [22, 30] that 90% of flows carry less than 1MB of data and more than 90% of bytes transferred are in flows of sizes greater than 100MB.

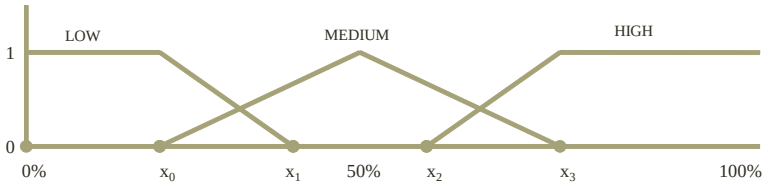
To incorporate recent research results on the nature of datacenter traffics into our performance study, we generated the simulated traffics as follows. The traffics were generated as mixtures of application-specific traffics, namely, those generated by the MapReduce application in its shuffle phase, and other background traffics. To generate MapReduce traffics, we applied the methodology presented in [23]. Therein, each host sends 128MB to every other host during MapReduce's shuffle phase. A host does so by simultaneously transferring five flows, each of size 128MB, to five other hosts; and once one of these flows is completed, it will start transferring another 128MB flow to one of the remaining hosts. The order in which a host is selected for receiving simulated MapReduce traffics is uniformly random. Other background traffics were generated following the macroscopic and microscopic measurements reported in [22, 30].

5.3. **Simulation methodology and performance metrics.** For each experiment, we generated traffics as described in the previous section, and performed traffic routing in Mahout and FuzzyDetec separately. In each experiment, MapReduce traffics generation was started only five minutes after background traffics generation, to allow the background traffics to reach a somewhat steady state. To track the performance of Mahout and FuzzyDetec, we aggregated the throughputs of all flows in each experiment, and recorded the average throughput of 50 experiments for each architecture.

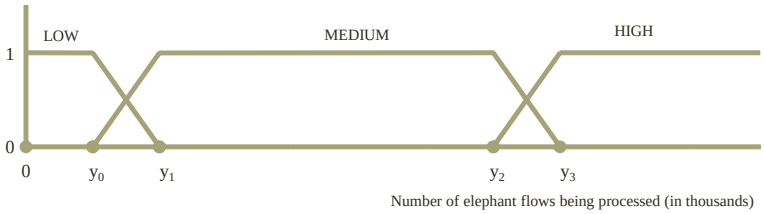
5.4. **Membership functions.** We used standard trapezoidal and triangular shapes for the membership functions of $curUtil$, $curCtrload$ and $Threshold$ in FuzzyDetec and put them in different configurations, as presented in Figure 6. The membership functions associated with $curUtil$ are characterized by four parameters x_0, x_1, x_2 and x_3. Similarly, those of $curCtrload$ and $Threshold$ were characterized by y_0, y_1, y_2, y_3 and z_0, z_1, z_2, z_3, respectively. For each set of specific values of x_i, y_i and z_i $(0 \leq i \leq 3)$, we have a specific configuration of membership functions for $curUtil$, $curCtrload$ and $Threshold$.

5.5. **Results.**

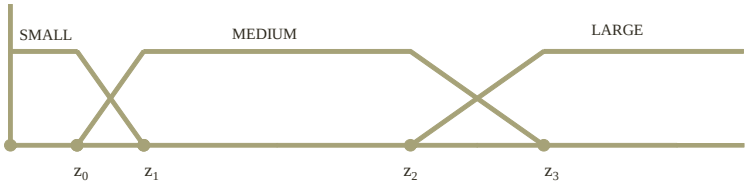
5.5.1. *Throughput performance.* We compared the throughput performances of using EC MP alone, using Mahout and using FuzzyDetec. We used five different thresholds for Mahout, namely, 128KB, 1MB, 20MB, 50MB and 100MB, and different configurations of



(a) Membership functions for linguistic values of $curNetUtil$



(b) Membership functions for linguistic values of $curCtrload$



(c) Membership functions for linguistic values of $Threshold$

FIGURE 6. Membership functions used for performance study

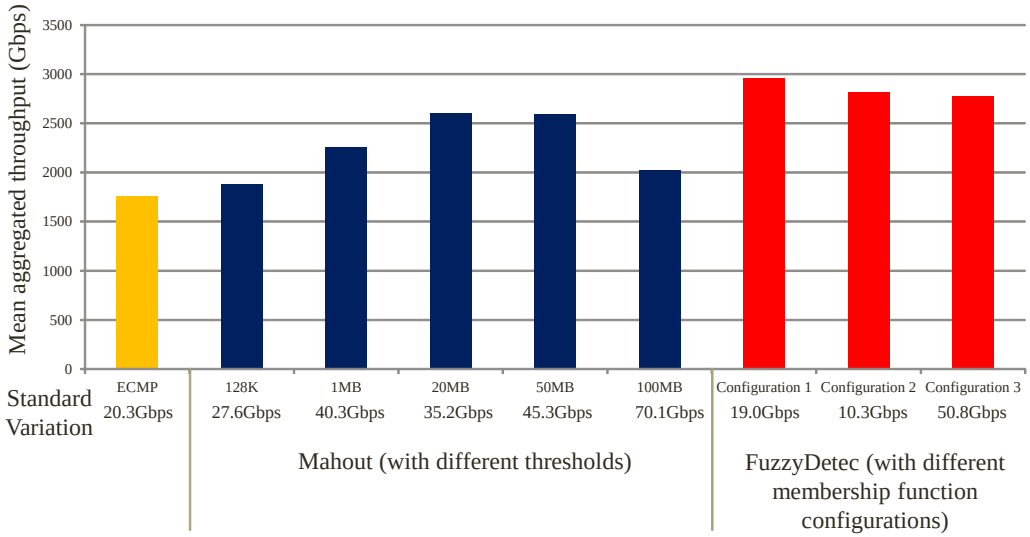


FIGURE 7. Mean and standard variation of throughput of 50 experiments for ECMP, Mahout and FuzzyDetec

membership functions for FuzzyDetec as described in the previous section. The center controller in FuzzyDetec only informs the end-hosts whenever there is a change of 10% or more in the threshold value. Figure 7 summarizes the representative results for three different configurations of membership functions for FuzzyDetec, as detailed in Table 2.

As can be seen from Figure 7, the throughput performance of Mahout is dependent on the chosen threshold value. Setting a threshold too low (i.e., 128KB in our experiment) or

TABLE 2. Parameter configurations for membership functions of *curUtil*, *curCtrload* and *Threshold*

	Utilization in %				No. of flows in thousands				Threshold in MB			
Configuration	x_0	x_1	x_2	x_3	y_0	y_1	y_2	y_3	z_0	z_1	z_2	z_3
1	20	40	60	80	100	200	700	800	0.5	1	20	50
2	30	45	75	90	200	400	800	900	1	20	50	80
3	40	49	90	99	200	600	600	900	20	50	80	99

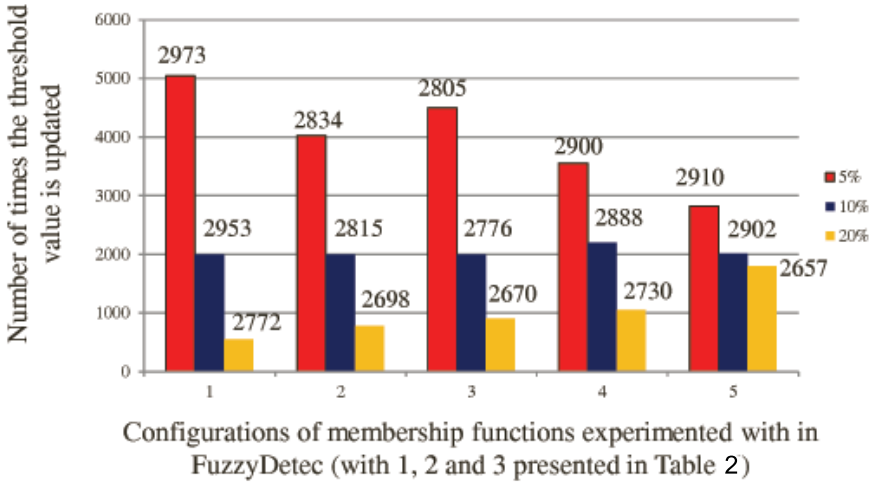


FIGURE 8. Number of times the threshold value is updated and the throughput (in Gbps) achieved (indicated on top of each bar) for each threshold change-setting of at least 5%, 10% and 20%

too high (i.e., 100MB in our experiment) will reduce throughput performance of Mahout significantly, to almost that of using ECMP alone.

As expected, FuzzyDetect performed better than Mahout in all cases. This increase in performance is attributed to both the intelligent fuzzy rules implemented in FuzzyDetec (see Table 1) and the shapes of the membership functions for *curUtil*, *curCtrload* and *Threshold*. To confirm this, we repeated our experiments with arbitrary, non-standard membership functions for *curUtil*, *curCtrload* and *Threshold* in FuzzyDetec, and found that in some extreme cases, the throughput performance was reduced significantly, and could be even worse than that of using ECMP alone. Using arbitrary non-intelligent rules also produced the same negative outcome.

5.5.2. *Control overhead.* We examined the communication overheads versus performance tradeoffs of FuzzyDetect, by counting the number of times the threshold value changed by at least 5%, 10% and 20% for different configurations of membership functions, and recording the throughput achieved for each such threshold-change setting. The number of times the threshold value changed as set indicates how many threshold update messages the center controller in FuzzyDetect needed to communicate to the end-hosts. The results are presented in Figure 8. Not unexpectedly, the higher the threshold-change setting, the lower the communication overheads. It can also be inferred from Figure 8 that the throughput performance can be improved, but at the expense of higher communication

overheads. Finally, we note that the throughput performance for all the configurations considered was better than that of Mahout depicted in Figure 7.

6. **Conclusion.** Timely detection of elephant flows is essential for selecting and deploying routing algorithms for datacenter networks with multipath topologies. In this paper, we have proposed FuzzyDetec, a novel close loop control architecture for the adaptive detection of elephant flows. By using fuzzy inference rules to intelligently incorporate human expertise and judgments in classifying elephant flow thresholds, FuzzyDetec is practically simple but crucial towards making feasible the autonomous operation of high performance end-host based elephant flow detection in datacenter networks. Our experimental results show that, over a well-known state-of-the-art architecture called Mahout [23], FuzzyDetec can achieve considerable improvement in throughput performance.

In conclusion, FuzzyDetec provides a new intelligent framework for datacenter network designers on the one hand, and serves as an important emerging application for fuzzy systems researchers on the other. The framework development is systematic, and is currently based on the most fundamental theory of fuzzy systems [1]. In future work, the framework would benefit from more advanced treatments in fuzzy systems, such as incorporating data mining techniques for efficient design of fuzzy inference rule sets [35], using probabilistic reasoning for automated selection of fuzzification schemes [36], and managing membership functions actively to enable robust on-line self-adaptation of fuzzy controllers [37]. We believe FuzzyDetec opens up rich opportunities for further theoretical and practical investigation.

REFERENCES

[1] H. H. J. Zimmermann, *Fuzzy Set Theory and Its Applications*, 2nd Edition, Kluwer Academic Publishers, Boston, Dordrect, London, 1995.

[2] M. Mas, M. Monserrat, J. Torrens and E. Trillas, A survey on fuzzy implication functions, *IEEE Transactions on Fuzzy Systems*, vol.15, no.6, pp.1107-1121, 2007.

[3] G. Feng, A survey on analysis and design of model-based fuzzy control systems, *IEEE Transactions on Fuzzy Systems*, vol.14, no.5, pp.676-697, 2006.

[4] C.-S. Lee, M.-H. Wang and H. Hagras, A type-2 fuzzy ontology and its application to personal diabetic-diet recommendation, *IEEE Transactions on Fuzzy Systems*, vol.18, no.2, pp.374-395, 2010.

[5] C.-S. Lee and M.-H. Wang, A fuzzy expert system for diabetes decision support application, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol.41, no.1, pp.139-153, 2011.

[6] H. Seker, M. O. Odetayo, D. Petrovic and R. N. G. Naguib, A fuzzy logic based-method for prognostic decision making in breast and prostate cancers, *IEEE Transactions on Information Technology in Biomedicine*, vol.7, no.2, pp.114-122, 2003.

[7] D. Xu, J. M. Keller, M. Popescu and R. Bondugula, *Applications of Fuzzy Logic in Bioinformatics*, Imperial College Press, London, 2008.

[8] K.-H. Huarng, T. H.-K. Yu and Y. W. Hsu, A multivariate heuristic model for fuzzy time-series forecasting, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol.37, no.4, pp.836-846, 2007.

[9] H. Huang, M. Pasquier and C. Quek, Financial market trading system with a hierarchical coevolutionary fuzzy predictive model, *IEEE Transactions on Evolutionary Computation*, vol.13, no.1, pp.56-70, 2009.

[10] K. Wang, C. K. Wang and C. Hu, Analytic hierarchy process with fuzzy scoring in evaluating multidisciplinary R&D projects in China, *IEEE Transactions on Information Technology in Biomedicine*, vol.52, no.1, pp.119-129, 2005.

[11] P. Maghouli, S. H. Hosseini, M. O. Buygi and M. Shahidehpour, A multi-objective framework for transmission expansion planning in deregulated environments, *IEEE Transactions on Power Systems*, vol.24, no.2, pp.1051-1061, 2009.

[12] L. R. Scott, T. Clark and B. Bagheri, *Scientific Parallel Computing*, Princeton University Press, 2005.

[13] J. Dean and S. Ghemawat, Mapreduce: Simplified data processing on large clusters, *Proc. of the USENIX Symposium on Operating Systems Design and Implementation*, San Francisco, CA, USA, pp.137-150, 2004.
[14] S. Brin and L. Page, The anatomy of a large-scale hypertextual web search engine, *Computer Networks and ISDN Systems*, vol.30, no.1-7, pp.107-117, 1998.
[15] *Cisco Data Center Infrastructure 2.5 Design Guide*, 2007.
[16] M. Al-Fares, A. Loukissas and A. Vahdat, A scalable, commodity data center network architecture, *Proc. of the ACM SIGCOMM Conference on Data Communication*, Seattle, WA, USA, pp.63-74, 2008.
[17] J. H. Ahn, N. Binkert, A. Davis, M. McLaren and R. S. Schreiber, Hyperx: Topology, routing, and packaging of efficient large-scale networks, *Proc. of the Conference on High Performance Computing Networking, Storage and Analysis*, Portland, OR, USA, 2009.
[18] J. Kim, W. J. Dally and D. Abts, Flattened butterfly: A cost-efficient topology for high-radix networks, *Proc. of the 34th Annual International Symposium on Computer Architecture*, San Diego, CA, USA, pp.126-137, 2007.
[19] T. Benson, A. Anand, A. Akella and M. Zhang, The case for fine-grained traffic engineering in data centers, *Proc. of the Internet Network Management Conference on Research on Enterprise Networking*, San Jose, CA, USA, pp.51-62, 2010.
[20] C. Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*, Internet Engineering Task Force, 2000.
[21] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang and A. Vahdat, Hedera: Dynamic flow scheduling for data center networks, *Proc. of the 7th USENIX Symposium on Networked Systems Design and Implementation*, San Jose, CA, USA, p.19, 2010.
[22] S. Kandula, S. Sengupta, A. Greenberg and P. Patel, The nature of data center traffic: Measurements and analysis, *Proc. of the Internet Measurement Conference*, Chicago, IL, USA, pp.202-208, 2009.
[23] A. R. Curtis, W. Kim and P. Yalagandula, Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection, *Proc. of the 30th IEEE International Conference on Computer Communications*, Shanghai, China, pp.1629-1637, 2011.
[24] J. Xu, M. Zhao, J. Fortes, R. Carpenter and M. Yousif, On the use of fuzzy modeling in virtualized data center management, *Proc. of the 4th International Conference on Autonomic Computing*, p.25, 2007.
[25] J. Xu et al., Autonomic resource management in virtualized data centers using fuzzy logic-based approaches, *Cluster Computing: The Journal of Networks, Software Tools and Applications*, vol.11, no.3, pp.213-227, 2008.
[26] J. M. Gil, J. H. Park and Y. S. Jeong, Data center selection based on neuro-fuzzy inference systems in cloud computing environments, *The Journal of Supercomputing*, 2011.
[27] *The OpenFlow Switch Consortium*, http://www.openflowswitch.org/.
[28] B. Braden, D. Clark and S. Shenker, Integrated service in the internet architecture: An overview, *Technical Report*, IETF, Networking Group, 1994.
[29] *sFlow*, http://www.sflow.org/.
[30] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel and S. Sengupta, Vl2: A scalable and flexible data center network, *Proc. of the ACM SIGCOMM Conference on Data Communication*, Barcelona, Spain, pp.51-62, 2009.
[31] J. Moy, *OSPF Version 2, RFC 2328*, Internet Engineering Task Force.
[32] J. R. Correa and M. X. Goemans, Improved bounds on nonblocking 3-stage clos networks, *SIAM Journal on Computing*, vol.37, no.3, pp.870-894, 2007.
[33] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by simulated annealing, *Science*, vol.220, no.4598, pp.671-680, 1983.
[34] T. Benson, A. Anand, A. Akella and M. Zhang, Understanding data center traffic characteristics, *ACM SIGCOMM Computer Communication Review*, vol.40, no.1, pp.92-99, 2010.
[35] J. Alcala-Fdez, R. Alcala and F. Herrera, A fuzzy association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning, *IEEE Transactions on Fuzzy Systems*, vol.19, no.5, pp.857-872, 2011.
[36] F. Diaz-Hermida, D. E. Losada, A. Bugarin and S. Barro, A probabilistic quantifier fuzzification mechanism: The model and its evaluation for information retrieval, *IEEE Transactions on Fuzzy Systems*, vol.13, no.5, pp.688-700, 2005.
[37] A. B. Cara, H. Pomares and I. Rojas, A new methodology for the online adaptation of fuzzy self-structuring controllers, *IEEE Transactions on Fuzzy Systems*, vol.19, no.3, pp.449-464, 2011.