Scientific Journal of Impact Factor (SJIF): 5.71

e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 5, Issue 04, April -2018

REINFORCEMENT OF ROUTING PERFORMANCE IN WSN THROUGH DYNAMIC PROGRAMMING

[1]T.ANITHA, [2]G.GAYATHRI

[1]*ASSISTANT PROFESSOR CSE DEPARTMENT ANITS*
[2]*ASSISTANT PROFESSOR CSE DEPARTMENT ANITS*

Abstract—*The Internet of Things has become a spotlight for a long period of time and generates massive amounts of sensor data. Thus, data centers play more and more crucial roles in processing and analyzing the explosively increasing data. To remedy the shortcomings of traditional tree-based structure, many novel server-centric network structures have been proposed in recent years. Their original routing mechanisms based on divide and conquer (DC) are not able to work out the shortest paths. So, there is still promotion room for communication delay reduction. Since dynamic programming (DP) is a classical strategy to obtain optimal solution, this paper proposes a routing mechanism based on DP and applies it to data center for better solving the weakness occurred by DC. Experiments firmly support the conclusion that adopting DP in routing calculation achieves appealing performance of short latency, great fault-tolerance and reasonable resource consumption. Theoretical analysis also proves that it is applicable to most popular structures.*

Index Terms—*Data center, dynamic programming (DP), Internet of Things (IoT), routing.*

I. INTRODUCTION

IOT refers to the networked interconnection of everyday objects [1] it is generally viewed as a self configuring wireless network of sensors whose purpose would be to interconnect all things[1]. The trend of the market is the information to be available independently of the place or the geographic location for this reason ,currently the internet is used to bring a real time interaction among devices that will not be possible with other mediums [2]. All the data gathered from the sensors must be available at the data centers to be managed and controlled. Thus a server-centric(central point of management) structure exists where the data from sensors remotely distributed is stored. Further, the industrial processes make necessary to implement wireless communication system (because of hostile environment and the difficult accesses to the places) to transmit the signals generated by the sensors making up the control [3] – [9]

IOT heralds a vision of the future internet where connecting physical things from bank notes to bicycles, through a network will let them take an active part in the internet exchanging information about themselves & their surroundings. This will give immediate access to information about the physical world & the objects in it which leads to innovative services & increase in efficiency productivity. According to Gartner, there will be nearly 26 billion devices on the IOT by 2020. ABI Research estimates that more than 30 billion devices will be wirelessly connected to the IOT by 2020 [10]. In the intervening period, the huge data generated by devices need to be organized in real time, which bring many challenges and workloads for IOT data centers.

Large scale data center networks in wireless sensor networks with adequate performance are crucial to organize and examine the huge amounts of data. In purpose of promoting efficiency of the IOT , optimizing the performance of data center network is the necessary point.

Any preferable data center networking should meet three requirements.

1. Network architecture must be adapted to increased demands (scalable) and allow future development.

2. Must be fault tolerant.

3. High network capacity/volume/Potential & (short latency) less intermission to better supporting real-time services.

In addition many scholars have proposed huge feasible schemes with the structures defined recursively, server centric ,which put linking and routing intelligence on servers instead of switches such as Dcell [10], Ficoun [11] , Totoro [12] , & so forth.

These structures help for fault tolerant capacity as the routing algorithms of most existing data center structure take advantage of the structural properties of symmetry & homogeneity and exploit divide and conquer (DC) approach to calculate forwarding path.

- DC approach is simple but not effective as they cannot assure to get optimal solutions to reduce overhead and communication delay (jitter).
- Dcell and Totoro both followed DC approaches on routing path.
- All link nodes information needs to be altered/revised at short intervals in case of any modifications or failure.

The basic principle of ARM is transferring the responsibility of path calculation on source server, the intermediate nodes only take charge of transmitting data packets. The validation and capacity of a path are confirmed by path probing scheme, thus broadcast can be omitted for saving bandwidth. Besides, our routing algorithm is based on DP, hence the shortest paths can be worked out primarily in path calculation process. The selection of paths and load balancing are achieved by path probing procedure. In the following parts, we will present the fundamental theory and implementation of our ARM. We will also prove the high efficiency and superior fault-tolerant capability of ARM at the support of extensive simulations. Moreover, our ARM is a propagable option which can be generalized to most other server-centric data center structures mentioned before. In this paper, we choose Totoro as the physical network architecture to test the performance of ARM.
Our work has the following strong points:

1) lower jitter;

2) minimization of fault-tolerant capability;

3) less resources consumption;
This paper is organized as follows. Section II elaborates on ARM. Section III describes the implementations and Section IV use simulations to evaluate ARM. Lastly, Section V concludes this paper.

II. **ARM**

In this paper ,our ARM consists of routing algorithm and a path probing policy. As we know, the general routing algorithm is based on broadcasting link status in a broadcast domain, and the source and intermediate servers do repetitive calculations to find a next hop continually with the link status. This policy causes considerable waste of computation and network bandwidth automatically. Hence, we propose another routing mechanism called ARM to address this problem. Since the physical structure is estimable, the path to any destination host can be worked out by the source host exclusively. Intermediate nodes only take charge of forwarding data packets. Before moving to the detailed implementation of ARM, we first focus on presenting the basic algorithm, Athena routing algorithm (ARA), which is conducted by the source host.

A. **DP**

DP is a method for solving a complicated problem by breaking it down into several simple sub problems, solving them and then combine the solutions of them to find an overall solution for the whole problem. DP is usually used for optimization [13]–[15]. It has two properties, which are:

1) optimal substructure and 2) subproblem overlap. Optimal substructure requires that the solved substructures at each step is always optimal. Subproblem overlap means that the solutions of subproblems overlap and thus the space of subproblems become small. Subproblem overlap is the main difference between “DC” and “DP.”

Algorithm 1. Athena Routing Algorithm

1 **Function** *ARoute (src*, *dst*, *count)*
2 if *src* == *dst* **then**

3 return *NULL*

4 *u* = getLCL (*src*, *dst*)
5 if *u* == 0 **then**

6 return *P* (*src*, *dst*)

7 Set *topLinkSet* = getTopLinks (*src*, *dst*, *u*)
8 Set *result*
9 for *each link L* ∈ *topLinkSet* **do**

10 *leftPathSet* = ARoute (*src*, *L.left*, *count*)
11 *rightP athSet* = ARoute (*L.right*, *dst*, *count*)

12 *result*.add (*leftPathSet* + *L* + *rightP athSet*)
13 SortByLength (*result*)

14 *result* = *result*.sublist (0, *count*)
15 return *result*

Fig.1 Overlap path in path set.

To achieve DP, there are two ways, which are: 1) top–down approach and 2) bottom–up approach. Only the neede subproblems are caluculated and solutions are maintained in the table so that when ever we try to solve the new problem we take the solutions, if it exists from the table to solve the problem .later we solve every low level subproblem use their solutions to obtain optimal solutions to the higher level subproblems.

Taking advantage of optimal substructure and subproblem overlap, DP presents an optimal solution for the given problem with far less time. Some classical algorithm problems are solved by DP, including Fibonacci sequence, sequence alignment, tower of Hanoi puzzle, egg dropping puzzle, knapsack problem, and so forth.

In all aforementioned recursively defined structures, a highlevel structure can be constructed by several low-level structures. Consequently, the path from a source to a destination can be divided into certain low-level paths. In addition, the shortest paths between two nodes are always existing, which means the requirement of optimal substructure is also met. Therefore, we can draw a conclusion that DP is able to applied to the routing computation process for data center networks.

B. **ARA**

ARA is based on DP to obtain simplicity and efficiency. We present how we recursively work out constant number of shortest paths in Algorithm 1. The function **getLCL** returns the lowest common level *u* of two nodes (Line 1). Then, the function **getTopLinks** (Line 1) figures out all level-*u* links starting from the whole level-*u* substructure which the source is located. Afterward, for each independent top link, we can recursively find a set of completed paths from the source server to the destination server. After each round of recursion completed, a sort function will be performed, so that we can limit the needed number of shortest paths. What noteworthy is the total independent paths might share same nodes, even though we choose absolutely different links in every round of recursion. This is because different high-level paths may share the same low-level paths. So when one path fails, we cannot assure that all the other paths are unaffected. This is a tradeoff to facilitate the routing algorithm and we can alleviate this problem by detecting multiple paths simultaneously. As Fig. 1 indicates, a set of paths connecting the same source and destination contain the same overlap path, which is already found out; thus repeated computation can be omitted.

C. **Path Probing of ARM**

The basic idea of ARM is figuring out all completed paths by ARA before sending probing packet (actually, we need not find all paths in practice; a threshold value can be utilized to limit the number of paths). If the source host src maintains a path cache to the destination host dst, this step can be omitted. Then, a set of probing packets will be dispatched to detect the capacities of selected paths. The intermediate servers record the link capacities in the probing packets and forward them to the destinations by the precalculated paths. After the probing packets arrive, the destination servers reply these probing requests by sending the probing packets back to the source hosts according to the original paths. Among all the valid paths,

we tend to choose the one with higher bandwidth and/or shorter length, so that we can get sufficient resources every time and utilize links evenly.

Fig. 2. Dividing level-*k* problem into level-*k* − 1 problems in Totoro*k*.

D. **Properties of ARA**

Our ARM is not constrained to some certain structure, but can be generalized to most server-centric recursive structures mentioned before. Take Totoro as an exemplification, our ARM performs well according to various experiments. Primarily, our routing algorithm works out all paths directly connecting two substructures, so no circuitous path is involved. With the feature of DP, we can ensure the shortest path length. In addition, our ARA takes consideration of all top-level switches connecting the substructures, which src and dst are located, respectively. Thus, we can get sufficient feasible paths by ARA to achieve a desirable fault-tolerant capacity. Moreover, the path probing policy of ARM is able to save a great number of bandwidth for Totoro, since the size of probing packet is far less than broadcast data packet among all servers. The intermediate servers are also released from heavy routing computation loads because they only carry on forwarding data packets.

Theorem 1: k is the structural level. *N* represents the total number of servers in Totoro*k*. *Tk* is denoted as the time complexity of calculating a level-*k* path. The time complexity of ARA is

TK=O() ….(1)

Proof: Suppose the source host and the destination host locate in different substructures, denoted as Totoro*k*−1[src] and Totoro*k*−1[dst], respectively. These two substructures are connected directly with level-*k* switches. Hence, shortest paths from Totoro*k*−1[src] to Totoro*k*−1[dst] must go through these switches directly without circuitously traversing other Totoro*k*−1s. The number of level-*k* switches is (*n*/2)*k*, i.e., the number of level- *k* links is also (*n*/2)*k*. For each level-*k* link (*m*, *n*), suppose *m* and *n* are two end nodes of this link (see Fig. 2), ARA has to calculate two subpaths: one is from *src* to *m* and the other is from *n* to dst. Calculation of subpath can be conducted by following the similar steps recursively. Therefore, we can get $Tk = 2\times Tk-1 \times (n/2)^k$. According to mathematical induction, we can finally figure out the following equation

TK= …… (2)

Omitting the denominator (since it is lager than 1), Theorem 1 is rigidly proved. Note that *k* is a small integer, a low-level Totoro (e.g., $n = 32$, $k = 3$) still can support more than one million servers (323+1 = 1 048 576). Thus the time complexity of ARA is relatively low when *k* is small. Similarly, we can also get the time complexity of ARA in DCell and Ficonn, respectively, in Theorems 2 and 3.

Theorem 2: k is the structural level. *N* represents the total number of servers in DCell*k*. *Tk* is denoted as the time complexity of calculating a level-*k* path. The time complexity of ARA is

$Tk = O(N\log^2 N)$. ……(3)

Theorem 3: k is the structural level. *N* represents the total number of servers in Ficonn*k*. *Tk* is denoted as the time complexity of calculating a level-*k* path. The time complexity of
ARA is

$Tk = O(N\log^2 N)$. ..... (4)

Comparing with shortest path algorithm (SPA) [based on Floyd–Warshall algorithm and the time complexity is $O(N3)$], our ARM is able to work out the shortest paths with more acceptable complexity. Hence, all above structures will get a huge promotion of average path length. In addition, according to the physical properties of the architecture, the paths between arbitrary pair of nodes can be totally figured out by the source server solely. Especially, for DCell, which adopts broadcasting link status among servers, our path probing mechanism will efficiently save network bandwidth as well as relieve

computation load for intermediate servers. All in all, our proposal can promote the whole efficiency and fault-tolerant capability for many structures.

III. ATHENA PROTOCOL IMPLEMENTATION

A. ARM Address

Since most applications are based on TCP/IP, we then design ARM protocol as a 2.5-layer protocol. In adaptation of Totoro structure, we represent a specific server by a 32-bits tuple named ARM Address. Since it has the same length with IP address, we then utilize this tuple in place of IP address in the IP header, i.e., we set the IP address to the same value of ARM address. Thus, we can use the source and destination address from IP header directly, rather than adding two additional fields.

There are three fields in and ARM address: *Li*, dir and vmid. *Li* denotes the server position in the network. In this paper, we suppose *i* is no more than three so that *Li* consists of tuples from *L*0 to *L*3 with 6-bits length each. Actually, *i* indicates the level of Totoro structure. Note that

a 4-level Totoro structure (k = 3, n = 48) can support as many as five millions servers. We can simply complete the high-order position or adjusting length of each *Li* field to apply to a smaller structure with less servers. The dir takes up 1 bit to indicate this port connects to an intra-switch or inter-switch. vmid means the index of virtual machine in a physical server. It occupies seven bits, so

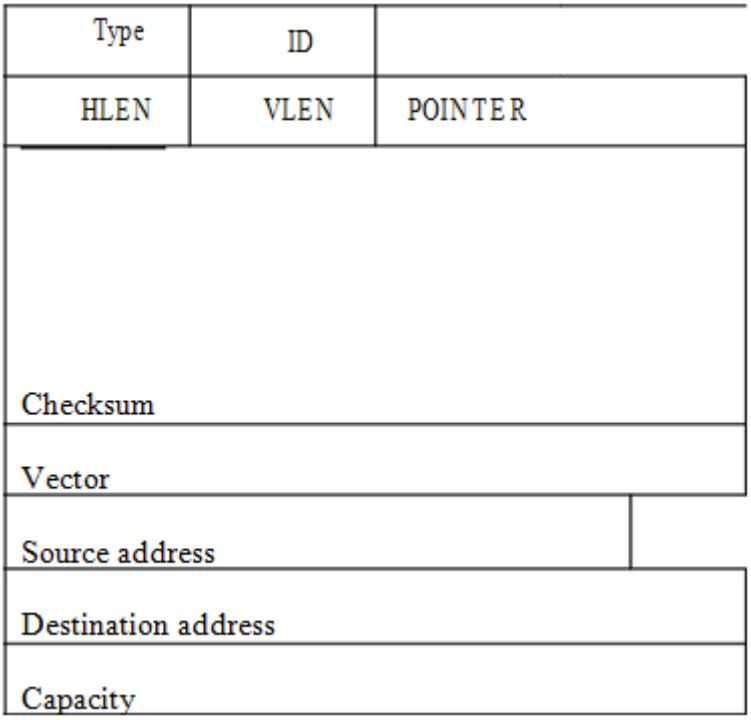


Fig.3. Header Format.

Checksum

Vector

Source address

Destination address

Capacity

Fig. 4. Vector calculation.

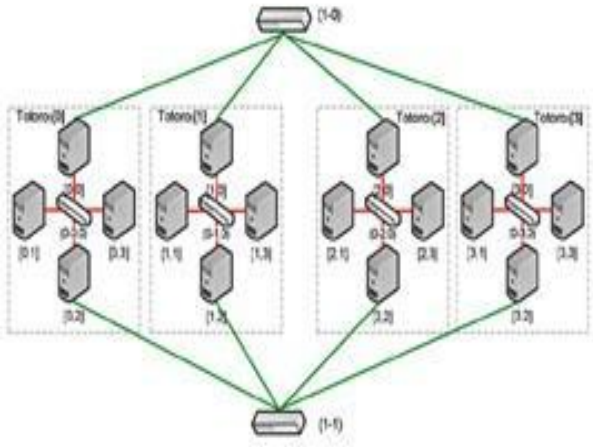
that we can support 127 virtual machines at most (vmid = 0, represents the physical server itself). We set this field only for adapting the trend of cloud computing, and it will not be used in the routing computing since only physical addresses are involved in our ARM and when data packet arrives at the target physical server, it will be transferred to the specific virtual machine by operation system.

B. **Packet Format**

There are two types of packet, which are: 1) path-probing packet and 2) data packet. Before dispatching a data packet, a set of path-probing packets will be delivered first to confirm the capacities of selected paths. It involves the source address and destination address, as well as the capacity of one path and so forth. However, the fields of source address, destination address and capacity are not required in data packets. Fig. 3 shows the format of packet header of these two types of packets.

C. **ARM Protocol**

As the fact that two adjacent servers in a path only differ at one "bit" (i.e., one item in Totoro tuple), we adopt the path transformation vector to preserve path information in Fig. 4. Vector is included in a data packet, and each item of this vector represents "one-bit" change. An intermediate server determines the next hop according to the value of item which current pointer is pointing at. This approach of preserving the path information in a vector has been adopted in former research [16]. If a server receives a packet from the upper layer, it first checks whether the destination address is a loop address or not. If so, then it returns the packet to the upper layer. Otherwise, the server checks if it maintains a cache of path information to the destination. If not, it then employs ARA to figure out a set of paths and update path cache. Then, the probing packet of request is constructed, in which the path transformation vector is also initialized. Afterward, the probing packet is dispatched, intermediate servers forward it to the next hop according to the vector. When it arrives at the destination host, the



Fi g. 5. Totoro1 structure with *n* = 4.

host will send back a reply message along the previous path. Then, a data packets will be dispatched to the destination along the selected path.

IV. EXPERIMENTS AND RESULTS

A. **Totoro Structure**

In order to evaluate the performance of our ARM, we simulate it on the physical framework of Totoro and compare with the original Totoro fault-tolerant algorithm (TFR) and SPA (based on Floyd–Warshall [17]). TFR is the original TFR, which broadcasts link status in a domain and calculates the routing path hop-by-hop. This section briefly presents the physical structure of Totoro. Totoro structure consists of a series of commodity servers with dual ports and low-end commodity switches. The basic partition of Totoro is denoted as Totoro0, constructed by *n* servers connecting to an *n*-port switch. As mentioned before, Totoro is a server-centric structure with recursive definition. A Totoro*i* ($i > 0$) is constructed from *n* Totoro*i*−1*s*. Each round of construction consumes half of the total available ports, and the rest half are remained for expansion. As Fig. 5, a Totoro1 structure with $N = 4$, $n = 4$ is composed of 4 Totoro 0*s*. Each Totoro0 has four servers and an intra-switch with four ports. Four Totoro0*s* connect through two inter-switches. Unlike DCell and FiConn, there are duple

direct links between two equivalent substructures, thus the redundant links can be fully used for distributing data flows. Please refer to [11] for details.

B. **Evaluating Failure**

In Fig. 6, we evaluate the path failure ratio of Totoro using ARM under four types of failure, including link failure, server failure, switch failure, and rack failure. We run ARA, TFR, and SPA on a Totoro2 (n = 16, k = 2, tk 4096) under those four types of failures. Meanwhile, the results are compared with TFR and SPA. A rack consists of a whole Totoro0. Failures are generated randomly ranging from the ratio of 2% to 20%. Servers deliver packet to other nodes 20 times, Which means every final result is the average of 20 running results.

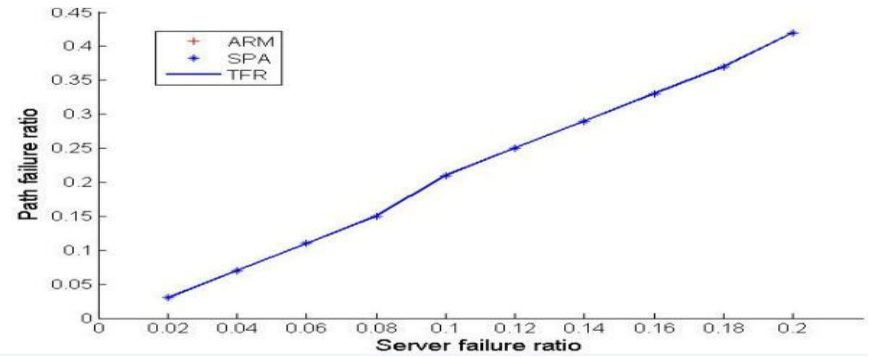
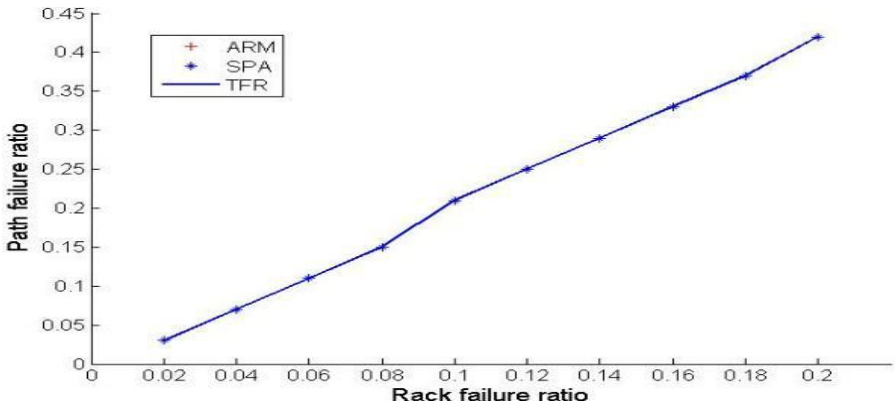
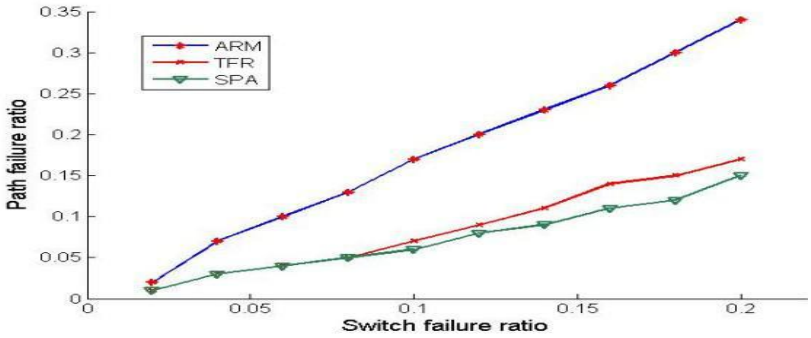


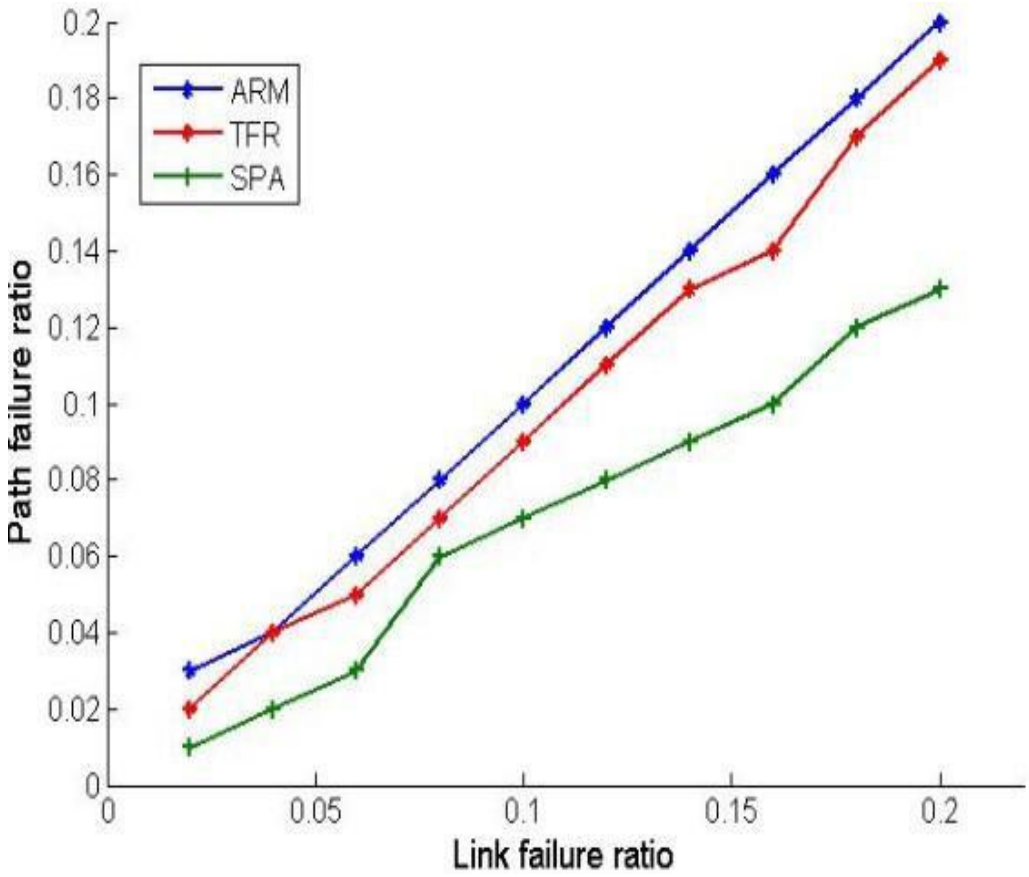
Fig. 6. Evaluating path failure ratios. **(a) Path versus server.**



(b) Path versus rack



(c) Path versus switch.



(d) Path versus link.

Before presenting our experiment results, we will introduce TFR and SPA briefly. By TFR, servers randomly choose a nearest level-*u* (*u* denotes the lowest common level with destination) link to the next hop, proceeding this process recursively until finding the destination. If failure occurs, another level-*u* or even higher links will be adopted. In addition, Totoro breaks the whole network into broadcast domains (TBD). In a TBD, link status are exchanged by broadcast among all servers. Thus the Dijkstra algorithm can be performed in a TBD to shorten path length. SPA is based on Floyd–Warshall algorithm, which requires global link states information to find out the shortest path. Consequently, SPA is globally optimal whereas the time complexity of it is as high as O(*N*3), where *N* denotes the total number of servers in a Totoro structure.

As Fig. 6(a) and (b) indicates, the path failure ratios of the all three algorithms are equivalent under server and rack failure, respectively. That is, the fault-tolerant capacity of ARM is almost optimal. It also proves that ARM makes full use of redundant links and switches between two substructures. Rack failure means all servers in a rack are invalid, so it is analogous to server failure. In Fig. 6(c), under switch failure scenario, ARM performs much better than the original TFR. From Fig. 6(d), we can see when a high link failure occurs, ARM achieves slightly better fault-tolerant capacity than TFR. But compared with SPA, ARM is still a bit inferior. This makes sense since all top-level links calculated in Totoro are direct links between two substructures. On the contrary, SPA will traverse all feasible links in the whole structure until finding a valid path. Hence this is a tradeoff that ARM makes to facilitate algorithmic complexity and save computation resources.

C. **Evaluating Path Length**

The efficiency of routing algorithm can be directly evaluated by path length. A short average path length contributes to a short latency. Table I lists the values of average path length calculated by ARM, TFR and SPA, respectively, for a Totoro2 (n = 16, k = 2, N = 4096). We take SPA as the benchmark of routing performance because SPA is globally optimal. Comparing the results of ARM and SPA, it is easy to draw a conclusion that the differences are negligible.

Failure Ratio		0.040.08	0.12	0.16	0.20
Server Failure	TFR	8.448.48	8.52	8.57	8.63
	ARM	7.347.41	7.48	7.56	7.64
	SPA	7.347.41	7.48	7.56	7.64
Link Failure	TFR	8.989.68	10.64	11.97	13.69
	ARM	7.477.68	7.90	8.16	8.47
	SPA	7.477.69	7.94	8.22	8.54
Switch Failure	TFR	8.438.46	8.50	8.53	8.58
	ARM	7.397.53	7.67	7.82	7.97
	SPA	7.407.55	7.70	7.86	8.03
Rack Failure	TFR	8.548.69	8.85	9.02	9.23
	ARM	7.337.40	7.48	7.57	7.66
	SPA	7.337.40	7.48	7.55	7.65

Table 1. Average Path Lengths In T16,2

In some cases, such as server failure and rack failure, the average path lengths of both are equivalent. Whereas in link and switch failure, our ARM achieves shorter path lengths than SPA does, this is because the path failure ratio of ARM is a bit higher than that of SPA, thus our total path length is shorter. Besides, ARM is much simpler than SPA, for the latter's computation complexity is as high as $O(N3)$, and it requires frequent exchanges of link status, which may cause heavy loads for data center. In conclusion, the failure experiments prove the great fault-tolerant capacity as well as high efficiency of our ARM.

Evaluating CPU Usage

The CPU usage of committing routing algorithm in source server to evaluate the computing efficiency. The experiment environment is under a Lenovo T350 G7 server with quad-core processors and 8 GB memory. We simulate a Totoro2 (n = 16, k = 2,N = 4096), and run ARM on the experimental server to work out 10 completed paths with any node in the same or neighbor 3 Totoro1 as the destination. Because of data locality, a server usually communicates with servers which are located in the same row or adjacent several rows. We set the initial nodes amount as 10 and increase by 10 per second until reaching the threshold value 500, and the total computing time is 3 min. As Fig. 7 indicates, the CPU usage continuously increases until achieving the peak value of 28% at around the 20th second. Afterward, it dramatically drops to 0% and remains to the end. Thus, we can get a conclusion that our ARM has great performance with rather low CPU usage. Besides, the dash line represents the number of nodes which the experimental

Fig**. 2. Dividing level-*k* problem into level-*k* − 1 problems in complete graph based structure.**

server calculates per second. From the graph, we can see the server figures out 10 paths of 500 nodes/s in the cost of 0% CPU usage, this is benefited from path cache constructed in the source server. Suppose a server maintains a cache of 10 completed paths to all of the rest nodes in the network. The maximal length of a path consists of five hops, and the vector of a path takes up 6 bits × 5 hops = 30 bits memory, e.g., is 4B approximately. Then, we can get the largest size of cache to preserve all path information on each host is 4B ×10 × 4096 ≈ 164 KB at most. This also further proves our ARM is resource saving.

V. CONCLUSION

In this paper, we applied DP to routing mechanisms of novelly proposed server-centric data center structures. Since communication latency is significant for IoT, the conventional routing algorithms cannot assure to get the optimal solutions. This motivates us to propose a universally applicable routing mechanism named ARM comprising a routing algorithm called ARA and a path probing scheme. On basis of DP, ARA is able to find out the shortest paths with lower time complexity than SPA. ARM adopts source routing and path probing scheme, which means the source server performs computing a set of completed paths and dispatches probing packets to detect the connectivity and capacities of paths. ARM is implemented by a 2.5-layer protocol and a 32-bits ARM address. In comparison with other conventional routing policies, our ARM simplifies the functionalities of intermediate servers as well as eliminates the extra bandwidth consumption caused by broadcasting link states among servers. Besides, our failure experiments on Totoro structure prove the satisfactory fault tolerant capacity of ARM comparing with TFR and SPA under different types of failures. We also demonstrate the relatively low CPU usage ratio of source server during the path computing process. Therefore, our ARM is a reliable and efficient mechanism which can be generalized to most server-centric structures to promote the overall performance of data center network. In the future work, we will focus on the implementation of ARM in DCell, FiConn, and other structures to further verify the performance of our ARM.

REFERENCES

[1] A special – purpose peer to peer file sharing system for mobile adhoc networks. A . Klemm; C.Lindemann; o.p. Waldhorst.

[2] S.cesar San Martin , F. Torres, R.Barrientos, & M,Sandoval, " Monitoneo Y control de temperetura de un estanque da agwa entre chile Y Espana usando redes de alta velocidad, " Revista FACULTAD DE INGENIERIA, VTA (CHILE), vol. 11,no .1, pp. 41 – 46, 2003.

[3] L . Hov and N. Bergmann, " Novel Industrial Wireless sensor networks for machine condition monitoring and fault diagnosis ," Instrumentation & measurement, IEEE Transactions on,Vol.61;no.10, pp. 2782 – 2798 , 2012.

[4] Z.Ke, L .Yang, X.Wang-hui , and S.HEEJONG, " The application of a wireless sensor network design based on Zigbee is petro chemical industry field ," in intelligent Networks and intelligent systems 2008 , pp . 284 – 287.

[5] G.Cena, A . Valenzano, and S . Vitluri , " Wireless exetensions of wired industrial communications networks, " in Industrial informaties , 2007 5th IEEE International conference on vol . 1 , 2007, pp . 273 – 278.

[6] K .Koumpis, L.Hanna, M. Andersson , and M. Johansson, " Wireless Industrial control and monitoring beyond cable replacement ", in proc . 2nd PROFIBUS int . conf ,combe Abbey, Warwickshine, UK, 2005.

[7] S.Trikaliotis & A . Gnad, " mapping wireless into profinet & profibus fieldbusses " , in emuging Technologies Factory Automation , 2009. ETFA A 2009. IEEE conference on 2009, pp. 1-4.

[8] Siemens, " profinet the industrial Ethernet standard for automation 8th IEEE international workshop on Factory communication systems COMMUNICATIONS IN AUTOMATION , 2007.

[9] D . Miorandi, E . Uhlemann S . Vitluri , and A . Willig, " Guest editional : special section on wireless technologies in factory & industrial automation , part i , " Industrial Informaties , IEEE Transactions on , vol. 3 , no. 2, pp . 95-98, 2007.

[10] C . Goo et al. , " Dcell : A scalable and fault tolerant network structure for data centers ", in proc . ACM SIGCOMM, 2008 , vol . 38, no. 4, pp. 75 – 86.

[11] D . Li et al . , " Ficonn; using backup port for server interconnection in data centers , " in proc. IEEE INFOCOM , 2009 , PP . 2276 – 2285.

[12] J . Xie , Y . Deng , and K . Zhou, " Totoro : A suxlable and fault tolerant data center network by using backup port ", in proc , Netw, parallel comput. , 2013, pp . 94 – 105.

[13] P. F. Felzenszwalb and R. Zabih, "Dynamic programming and graph algorithms in computer vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 4, pp. 721–740, Feb. 2011.

[14] S. Tang *et al.*, "Easypdp: An efficient parallel dynamic programming runtime system for computational biology," *IEEE Trans. Parallel Distrib Syst.*, vol. 23, no. 5, pp. 862– 872, Apr. 2012.

[15] J. Li, G. Tan, and M. Chen, "Automatically tuned dynamic programming with an algorithm-by-blocks," in *Proc. IEEE 16th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, 2010, pp. 452–459.

[16] G. F. Riley, M. H. Ammar, and E. W. Zegura, "Efficient routing using nix-vectors," in *Proc. IEEE Workshop High Perform. Switching Routing*, 2001, pp. 390–395.

[17] R.W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, 1962.

