Journal of Applied Computing Research, 1(2):76-83
July-December 2011
 - doi: 10.4013/jacr.2011.12.02

Flow Based Load Balancing: Optimizing Web Servers Resource Utilization

Daniel Stefani Marcon, Leonardo Richter Bays
Universidade Federal do Rio Grande do Sul. Av. Bento Gonçalves, 9500, 91501-970, Porto Alegre, RS, Brasil.
dsmarcon@inf.ufrgs.br, lrbays@inf.ufrgs.br

Abstract. The expansion of the Internet has caused a growth on the number of users requesting services through the network, as well as the number of servers and the amount of services they offer. In order to minimize this problem, web servers have started to use a distributed architecture implementation, however with only one external interface for receiving requests from users. In this paper, we propose an approach towards flow-oriented load balancing, using the OpenFlow technology. Thus, each data flow is directed to a server, according to the policy being employed. We evaluate three load balancing techniques: random choice, time slice based choice and weighted balancing, each of them with its advantages and disadvantages. Through our measurements, weighted balancing achieved the best results over the other policies. Moreover, random choice and time slice based choice are capable of distributing the load in an acceptable way among nodes, considering the average load of each server.

Keywords: Load balancing, OpenFlow, Flow based load balancing.

Introduction

The expansion of the Internet caused a growth on the number of users requesting services over the network, as well as an increase in the number of servers and in the amount of services they offer. Moreover, users expect to receive responses in a short period of time, regardless of the number of requests that web servers must process at a given time. This new scenario introduces several challenges to the client/server model used on the Internet, considering that the level of scalability of a single web server is relatively low (Kwan *et al*., 1995; Colajanni and Yu, 1997).

In order to minimize this problem, web servers have started to use a distributed architecture implementation, but with a single external interface for receiving requests from users. Thus, load balancing mechanisms are necessary to distribute requests among multiple nodes offering the same service (Aweya *et al*., 2002). Hence, the scalability of the service being offered is proportional to the efficiency of the load balancing algorithm used to guarantee an optimal utilization of available resources.

Recent advances in network management technologies enable dynamic reprogramming of devices by means of data flows. The OpenFlow protocol (McKeown *et al*., 2008) allows the reprogramming of switches in a network through an external controller, which centralizes management operations.

In this paper, we propose an approach towards flow based load balancing, using the OpenFlow technology. Therefore, each data flow is directed to a server, according to the policy being employed. To evaluate the effectiveness of this approach in the context of load balancing, we use three distinct policies: random choice, time slice based choice and weighted balancing.

Our measurements showed that weighted balancing achieved the best results over the other policies, despite the complexity of the code of this policy. The other two policies, random choice and time slice based choice, are capable of distributing the load in an acceptable way among nodes, considering the average load of each server. Despite proving the feasibility of our approach, which was our goal, more measurements are necessary to test the performance and the scalability of the OpenFlow in a real and complex scenario.

This paper is structured as follows. Section 2 describes the best techniques for load balancing in the state of the art, as well as the OpenFlow technology. Section 3 presents the proposed solution for load balancing and the experiments performed to evaluate our approach, and Section 4 exhibits the results obtained from the experiments. Last, Section 5 concludes the paper and outlines future work to be done.

Related Work

In this Section, we present a number of load balancing proposals found in the literature which are closely related to ours, as well as the OpenFlow technology, used by our approach to manipulate data flows in benefit of load balancing. Last, we present a summary of the load balancing proposals found in the state of the art, indicating how OpenFlow could assist in addressing their limitations.

Load Balancing

Load balancing is a topic with several academic works in the literature. In (Colajanni and Yu, 1997), the authors proposed the union of two techniques, namely round-robin policy and adaptive TTL (time-to-live) scheme. The combination of these techniques takes into account the unequal rates of client requests and the web server heterogeneity to map each request to a specific server.

In another paper, the authors propose an algorithm for load balancing based on real time server statistics (Real Time Statistics Server-based Load Balancing - RTSLB). This technique considers three main factors: CPU utilization of the web server, its response rate and the number of requests being handled by the server (Shadrach *et al*., 2009).

On the other hand, in (Xu *et al*., 2004), web traffic is modelled by stochastic processes and some load balancing algorithms based on DNS (Domain Name Server) are compared through simulations. The results showed that the algorithm RR2 (Two-Tier Round-Robin), which uses information from the user domain and from the load of the servers, had the best performance.

Another proposed solution is based on a DNS round-robin technique, with the addition of two new functionalities. Initially, a new algorithm for indexing the performance and the load of the nodes is described. Secondly, a scheduler is proposed for task distribution in the DNS server, which allocates a server based on the performance index and on the load index. To validate the proposal, the authors performed a simulation, comparing this approach with the standard round robin technique. The proposed solution got better results, completing its tasks in a shorter period of time (Chin *et al*., 2010).

In (Yang *et al*., 2009), the authors propose a technique called Random Early Detection (RED), which is used to detect web servers with high probability to be overloaded in the near future. According to RED, the probability of a server becoming overloaded is related to its current load. This technique is intended to mitigate the problem of DNS based techniques, which is related to the oscillation of the load among nodes offering the same service. In the simulations performed, this solution enabled greater stability, improving the quality of service.

Open Flow

The OpenFlow technology allows the testing of experimental protocols in real networks, concomitantly with production traffic. It is an abstraction and virtualization technology for networking, allowing control over network traffic through data flows (Kanaumi *et al*., 2010).

This technology enables the management of programmable switches, allowing the isolation of different types of traffic (e.g. experimental traffic and production traffic) by means of an external controller. This is possible due to the flow tables present in switches and routers, which are used to implement NAT (Network Address Translation), QoS (Quality of Service), and other features. Despite the variation of these tables among different manufacturers, there is a standard set of functions present on all devices.

An OpenFlow based network is composed of three main components: OpenFlow switches, the OpenFlow protocol and the OpenFlow controller (Kanaumi *et al*., 2010). The relationship among network components can be viewed in more detail in Figure 1.

An OpenFlow switch is composed by three main components:

- Flow tables: these contain the information used by the switch to process frames of a given flow. For flows under the control of OpenFlow, possible actions are defined

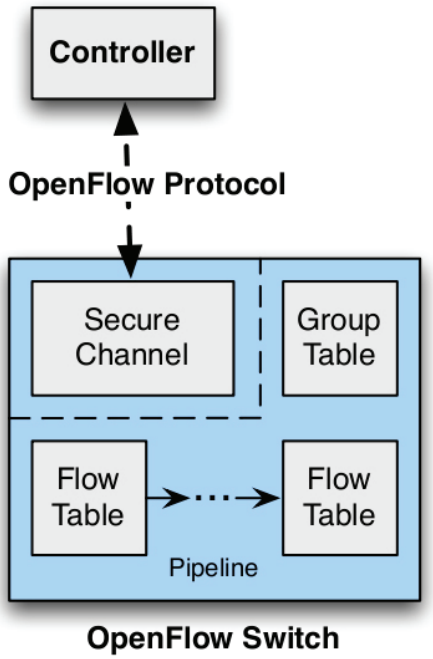

Figure 1. OpenFlow Switch (McKeown *et al*., 2011).

in the protocol specification (McKeown *et al*., 2011);
- Group table: it allows a flow to point to a group, increasing the forwarding options of frames;
- Secure communication channel: it is used for the communication between a switch and a remote controller, enabling control commands to be sent to the device responsible for managing network traffic.

The protocol defines the standard for communication between a switch and a controller, allowing the addition, removal and update of entries in flow tables. The third component, the controller, is responsible for managing the switch through the OpenFlow protocol.

The main function of the controller is to add and remove flows from the switch flow tables (McKeown *et al*., 2008). Upon receiving a frame, an OpenFlow-enabled switch searches its flow tables for any actions defined for the flow of the received frame. If an entry is found, the specified actions are performed and the frame is forwarded. Otherwise, the frame is sent to the controller, which will determine the action to be performed for the received frame, possibly adding an entry into the flow table for the next frames of the same flow. This flexibility of the controller allows the specification of a given flow and its actions with great level of detail.

Summary

Load balancing solutions proposed in the literature have some limitations, which can cause problems in certain scenarios. DNS based proposals have several advantages, especially when the problem is related to scalability. However, such solutions rely on the web server to send a message to the DNS server when it is overloaded. The overload of the web server may cause delays in sending such information to the DNS server, compromising service availability.

RTSLB algorithm takes into account several factors to distribute the load among servers; nevertheless, it depends on the premise that web servers send statistical information about their load. This transmission of information increases both network traffic and the complexity to choose a server, which may lead to delays in the selection of a server.

The use of a flow based platform, such as OpenFlow, is beneficial in the context of load balancing. Unlike the solutions presented above, it requires no action from the servers. The external controller has complete knowledge of the network, including the amount of active flows per server, which allows it to perform load balancing in an autonomous way.

Proposed Solution

This Section presents the proposed solution for the purpose of load balancing, through the utilization of the OpenFlow technology. Afterwards, the test environment and performed experiments are described.

Data Flow Based Balancing

In order to perform load balancing between several servers within a network, we propose a data flow oriented approach. A data flow is established the moment in which a client sends a request to a server. This flow remains active while there is communication between both entities, expiring after a certain period of inactivity.

Distinct data flows are divided between the existing servers in a way that is transparent for the client. Flows are managed by switches present in the network, which redirect packets to a server chosen through a load balancing policy. In a first step to evaluate solutions for the load balancing problem, we propose to combine three well known

policies with OpenFlow to distribute flows among network servers, with varying levels of complexity:

- **Random choice:** each time a flow is established, the controller selects a server in a random way. This is the simplest policy, where it is not necessary to store any data; however, the current load of each server is not considered.
- **Time slice based choice:** at each time interval, the controller selects a server which will be responsible for responding to any request made within this period. The server selection at each time interval is made sequentially (*round robin*). This way, no server will be responsible for more time slices than any other server. In this policy, the only information which needs to be stored is the server responsible for that time period; however, the current load of each server is also not considered.
- **Weighted balancing:** the controller has a general record regarding how many flows are being redirected to each of the servers. This way, in order to choose a server in the establishment of a new flow, it will always choose the server with the minimum load level; that is, the server which is currently communicating with the smallest number of clients. At each flow creation event, the flow counter for this server is incremented, and this same counter is decremented whenever a flow is removed.

Experiments

Initially, in order to perform the experiments, virtual networks with switches which support the OpenFlow protocol were created through the application Mininet (Lantz *et al*., 2010). The created network has a tree based topology, as presented in Figure 2. This topology holds a total of 16 hosts and five switches. Four or the 16 hosts present in the network were used as servers (hosts connected to switch **s18**, highlighted in the figure), and the remaining hosts operate as clients. All clients access the servers through the IP address of the main server (**h1**); however, all packets are modified before reaching their destination, so that load balancing can be performed between all four servers in the network (**h1**, **h2**, **h3**, and **h4**).

In order to perform load balancing between the servers present in the network, a script was created in the Python programming language using the API of the NOX controller (Gude *et al*., 2008). Each time a packet is received by a switch, this switch verifies if there is already a flow defined for this packet. If no flows exist, the switch alerts the controller, which will decide which action should be taken. The controller provides the default behaviour for every switch in the network. In other words, all switches possess the capability of learning MAC addresses and the respective ports in which the devices corresponding to these addresses are connected.

Additionally, the controller provides to switch **s17**, which interconnects the other switches present in the network, the capability of performing load balancing between the four servers in the network. Load balancing is performed through the dynamic creation of flows in this switch, which is located in the top level of the network. Upon receiving a packet addressed to the main server (**h1**), a flow is created in order to redirect packets to one of the four servers, according to the selected load balancing policy. This way, all subsequent packets received from the same origin in this connection will be redirected to this same server.

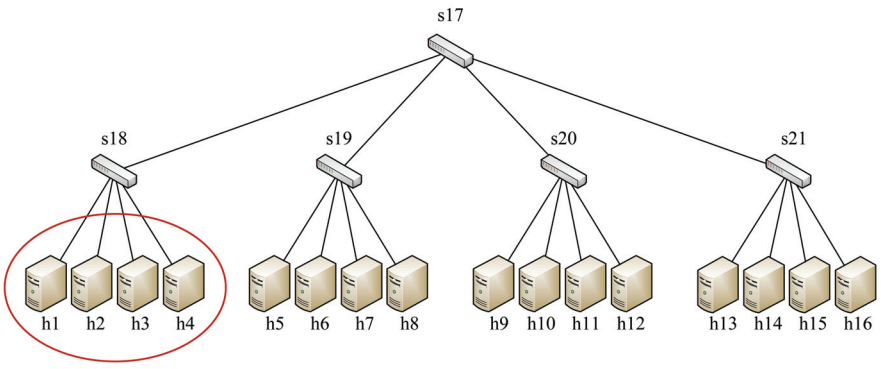

Figure 2. Network topology used for the experiments. The highlighted area shows the hosts used as servers.

The previously described scenario was instantiated for all three data flow based load balancing policies proposed in Section 3.1: random choice, time slice based choice (with five second time slices) and weighted balancing.

The created flows also have an associated timeout. In other words, flows are removed after a certain period of inactivity. This guarantees that inactive flows will not be used by other connections after the timeout is reached. This way, if there are variations in the activity levels between clients and servers over time, it is possible to adapt the load balancing.

In order to simulate client behavior, a script was created to run continuously in each client machine in the network. This script runs for 30 minutes (1800 seconds), and has the following behavior:

1. Sends *ICMP echo request* packets to the server continuously, for a period which may last between 60 and 300 seconds (1 to 5 minutes);
2. After this cycle, the client remains inactive for a period that lasts between 15 and 45 seconds. For testing purposes, flow timeout was set to a low value (10 seconds). This inactivity period present in the script guarantees that the flow will remain inactive for enough time to be removed from the switch;
3. Finally, after the inactivity period, the client returns to the initial state and resumes sending packets.

For each load balancing policy, the following steps were performed in the experiments:

1. The NOX controller is initialized with the controller script referring to the policy which is currently being tested;
2. Mininet is initialized with the network topology presented in Figure 2;
3. The testing script is started simultaneously in all network hosts which are not servers; that is, in 12 of the hosts.

Results

As described in the previous section, all tests were performed in a virtual network created through the Mininet tool. Data was collected through logs generated by the implemented controller itself. Log entries are created after every flow creation or removal event, and show the exact amount of active flows in the network during the testing period.

In tests performed using the random load balancing policy, it is possible to notice differences in load levels throughout the execution. The time series presented in Figure 3 shows that there was a distribution of load between the servers. However, it is possible to see periods in which certain servers had higher or lower charges than the others. Server **h4**, for example, was the server with highest load level in the period between approximately 6 and 11.5 minutes of execution. This represents a period of 5.5 consecutive minutes, or approximately 18% of the execution time for the experiment.

The time slice based policy shows a similar behavior during most of the execution. However, in the results obtained from the performed experiments, shown in Figure 4, it is possible to notice that there was a substantial peak in the number of connections to a single server in the initial testing period. This is due to the fact that all clients start sending packets at exactly the same time, and in this load balancing policy, a single server is chosen during a certain time slice. In the re-

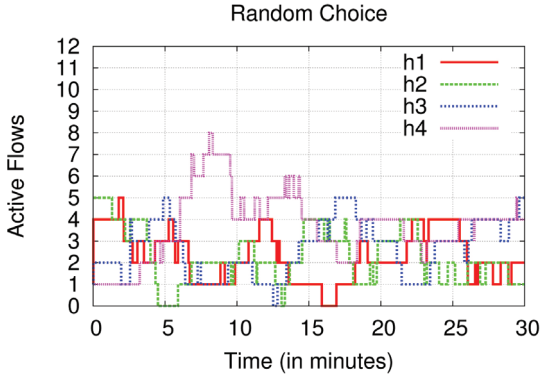

Figure 3. Server load levels during the experiment, using the random choice policy.

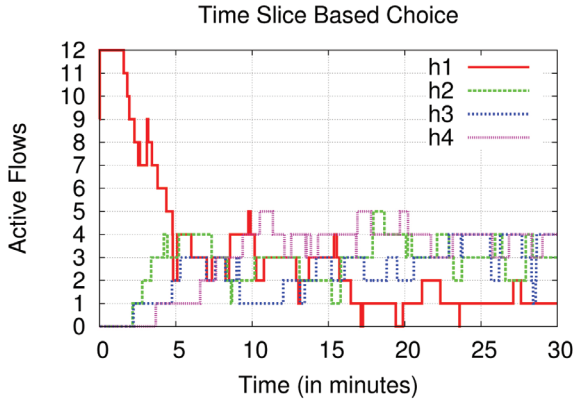

Figure 4. Server load levels during the experiment, using the time slice based policy.

mainder of the execution, load balancing results were similar to the ones obtained using the random policy. However, these results show that this policy is not adequate in flash crowd situations, where there is a substantial growth in the number of users in a network in a short period of time.

In contrast to the results from the first two policies, weighted balancing showed itself capable of providing nearly ideal load balancing during the entire execution period. As can be seen in Figure 5, load levels remain equally balanced between all servers, avoiding situations in which certain servers have too high or too low network loads.

Comparing average server loads during the execution time of the experiments, it is possible to see that there was no significant difference between all three policies. Using the random choice policy, the lowest average of active flows obtained was ≈2.3, while the highest was ≈3.6. Using the time slice based policy, the lowest and highest averages were, respectively, ≈2.3 and ≈3.1. Using weighted balancing, the averages remained between ≈2.5 and ≈2.8.

This comparison can be viewed in Figure 6, and it shows that weighted balancing was able to obtain greater equilibrium between the average server loads. However, we believe that such difference was not significant enough to be taken into consideration. It is also possible to observe that even though the time slice based policy produced an initial load level peak in server **h1**, average loads remained balanced between all four servers.

In the comparison between maximum load levels, presented in Figure 7, it is possible to see more significant differences. Using the random choice policy, peaks remained between 5 and 8 active flows per server. Using the time slice based policy; maximum load levels observed were between 4 and 12 active flows. The flash crowd event in the beginning of the experiment led one of the servers to be overloaded, responding to requests from all clients in the network simultaneously.

However, using the weighted balancing policy, none of the servers had a number of active flows greater than 3. This shows that this policy obtained a much greater success balancing the network load, since none of the 4 servers, in any moment, responded to a number of clients greater than ¼ of those present in the network.

Conclusions

Load balancing is a topic of great importance to ensure the efficient use of available resources, which is not limited to web servers. Nevertheless, it is a difficult task to implement an efficient policy, providing optimal resource utilization for nodes running the same web service. In other words, besides ensuring the optimization of available resources through a chosen policy, we should also take into account the complexity of the code that implements this policy.

In this sense, OpenFlow enables the creation of new techniques for load balancing, with greater flexibility and control over each server and also over each data flow, providing the ability to control the load of the servers at any time. Hence, we proposed three flow based load balancing policies. Through our measurements, weighted balancing achieved the best results over the other policies. Nonetheless, the complexity of the code of this policy is linear in relation to the number of servers (O(n)),

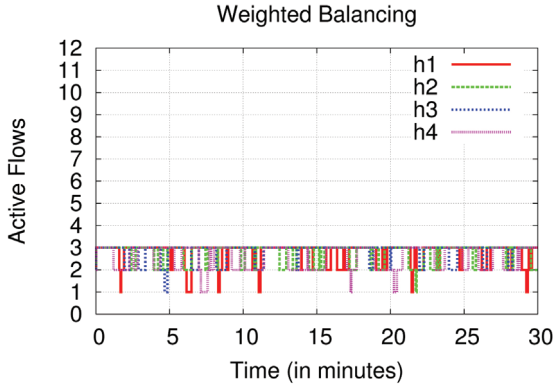

Figure 5. Server load levels during the experiment, using the weighted balancing policy.

Figure 6. Average server load during the experiments.

Figure 7. Maximum server load during the experiments.

which may be a disadvantage if there is a large number of nodes offering the same service in a distributed way. This is due to the fact that this policy must go through the whole list of servers for each new flow in order to find the least overloaded node at that moment.

According to the results, random choice and time slice based choice are capable of distributing the load in an acceptable way among nodes, considering the average load of each server. However, both policies have problems, which may result in servers being overloaded or underloaded. Such unequal distribution of flows does not occur in weighted balancing. In addition to this, we observed that the time slice based choice does not provide proper distribution of load in case of flash crowd events.

Future work involves a detailed analysis of the proposed load balancing policies in real networks, comparing them with state of the art proposals. In addition to this, we intend to propose new load balancing schemes, combining them with failover techniques in order to increase the availability of web servers.

References

AWEYA, J.; OUELLETTE, M.; MONTUNO, D.Y.; DORAY, B.; Felske, K. 2002. An adaptive load balancing scheme for web servers. *International Journal of Network Management*, **12**:3-39. http://dx.doi.org/10.1002/nem.421

CHIN, M.L.; TAN, C.E.; BANDA, M. 2010. Efficient load balancing for bursty demand in web based application services via domain name services. *In:* ASIA-PACIFIC SYMPOSIUM ON INFORMATION AND TELECOMMUNICATION TECHNOLOGIES, 8th, Kuching, 2010. *Proceedings…* Kuching, APSITT, p. 1-4.

COLAJANNI, M; YU, P.S. 1997. Adaptive TTL schemes for load balancing of distributed web servers. *SIGMETRICS Performance Evaluation Review*, **25**:36-42. http://dx.doi.org/10.1145/262391.262401

GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M.; MCKEOWN, N.; SHENKER, S. 2008. Nox: towards an operating system for networks. *SIGCOMM Computer Communication Review,* **38**:105-110. http://dx.doi.org/10.1145/1384609.1384625

KANAUMI, Y.; SAITO, S.; KAWAI, E. 2010. Toward large-scale programmable networks: Lessons learned through the operation and management of a wide-area openflow-based network. *In:* INTERNATIONAL CONFERENCE ON NETWORK AND SERVICE MANAGEMENT, 6th, Niagara Falls, 2010. *Proceedings…* Niagara Falls, CNSM, p. 330-333. http://dx.doi.org/10.1109/CNSM.2010.5691225

KWAN, T.; McGRATH, R.; REED, D. 1995. NCSA's World Wide Web server: design and performance. *Computer*, **28**(11):68-74. http://dx.doi.org/10.1109/2.471181

LANTZ, B.; HELLER, B.; McKEOWN, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. *In:* PROCEEDINGS OF THE NINTH ACM SIGCOMM WORKSHOP ON HOT TOPICS IN NETWORKS, 10, New York, 2010. *Proceedings…* New York, p. 1-6. http://dx.doi.org/10.1145/1868447.1868466

McKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. 2008. Openflow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, **38**:69-74. http://dx.doi.org/10.1145/1355734.1355746

McKEOWN, N.; PFA, B.; LANTZ, B.; HELLER, B.; BARKER, C.; COHN, D.; TALAYCO, D.; ERICKSON, D.; CRABBE, E.; GIBB, G.; APPENZELLER, G.; TOURRILHES, J.; PETTIT, J.; YAP, K.; POUTIEVSKI, L.; CASADO, M.; TAKAHASHI, M.; KOBAYASHI, M.; BALLAND, P.; RAMANATHAN, R.; PRICE, R.; SHERWOOD, R.; DAS, S.; YABE, T.; YIAKOUMIS, Y.; KIS, Z.L. 2011. Openflow switch specification version 1.1.0. Available at: http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf. Accessed on: 24/05/2011.

SHADRACH, D.; BALAGANI, K.; PHOHA, V. 2009. A weighted metric based adaptive algorithm for web server load balancing. *In:* INTERNATIONAL SYMPOSIUM ON INTELLIGENT INFORMATION TECHNOLOGY APPLICATION, 3, Nanchang, 2009. *Proceedings…* Nanchang, IITA, p. 449-452. http://dx.doi.org/10.1109/IITA.2009.84

XU, Z.; HUANG, R.; BHUYAN, L. 2004. Load balancing of dns-based distributed web server systems with page caching. *In:* INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED SYSTEMS, 10, Newport Beach, 2004. *Proceedings…* Newport Beach, ICPADS, p. 587-594. http://dx.doi.org/10.1109/ICPADS.2004.1316141

YANG, C.C.; CHEN, C.; CHEN, J.Y. 2009. Random early detection web servers for dynamic load balancing. *In:* INTERNATIONAL SYMPOSIUM ON PERVASIVE SYSTEMS, ALGORITHMS, AND NETWORKS, 10, kaohsiung, 2009. *Proceedings…* Kaohsiung, ISPAN, p. 364-368. http://dx.doi.org/10.1109/I-SPAN.2009.44

Submitted on October 10, 2011.
Accepted on December 12, 2011.