Multiple Service Load-Balancing with OpenFlow

Marc Koerner
Technische Universitaet Berlin
Department of Telecommunication Systems
Complex and Distributed IT Systems
Einsteinufer 17, 10587 Berlin, Germany
Email: marc.koerner@tu-berlin.de

Odej Kao
Technische Universitaet Berlin
Department of Telecommunication Systems
Complex and Distributed IT Systems
Einsteinufer 17, 10587 Berlin, Germany
Email: odej.kao@tu-berlin.de

***Abstract*—Load balancers have a decisive role in every enterprise network as they serve often as an entry point and have major impact on the performance and the availability of the network. While current load balancers are mostly implemented as specific hardware components, we developed a load balancer service based on the OpenFlow controllers to handle the load of multiple services without the necessity for a specific piece of hardware. This approach integrates the network and the load balancing functionality and reduces the maintenance effort. We increase the efficiency by providing dedicated and thus specifically adapted load balancing algorithms for the involved services. For example one controller handles the standard network traffic between the server nodes and other network components, while another controller handles the load-balancing of the web-servers and another one the load-balancing of e-mail servers. Furthermore experimental measurements in the local OpenFlow island using the FlowVisor and NOX controllers prove the performance capabilities of the developed prototype.**

I. Introduction

Load balancers have a decisive role in every enterprise network as they serve often as an entry point and have major impact on the performance and the availability of the network. They distribute the incoming server workload to an array of replicated servers in order to serve more clients with a minimum of latency and a maximum of throughput. This operation requires a rewrite of the destination NAT in the IP header. Incoming requests on one public IP are spread to a private subnet of servers and vice versa. Often applied load balancing strategies include policies such as round robin, random, load-based or connection-based balancing.

The efficiency increase has a price in terms of service availability, as the load balancers often represent a logical single point of failure. This problem is usually addressed by complex clustering solution, which on the other hand requires deep understanding of administration, often coupled with knowledge about the proprietary hardware solutions. Nevertheless load balancers are essential for state of the art data center and the services they provide.

In order to eliminate the complexity at the entry point in a data center and thus reduce the probability of failure, we analysed options to remove the separate hardware components and to integrate the load balancing functionality with the existing switching and routing elements. The basic for this research is the OpenFlow approach for the design of adaptable networks [1]. The load balancing algorithms are applied to the entries in the adaptable switches, so the two step operation – definition of the query target and the forwarding of the query to the determined server – are summarized into modification of the flow tables in the corresponding switches. This approach eliminates the necessity for a specific balancing component and allows an extensible and efficient solution, as the load balancing algorithms can be selected based on various parameters, e.g. the target service, the applied protocol, the current load situation, the source IP address etc. Finally, by dividing the functionality across multiple switches, we eliminate the single point of failure. In worst case, one service network might be offline, while the other service networks operate without interruption, different than a data center entirely depending on a (clustered) load balancer.

The feasibility of the proposed solution for the application in large data centers is supported by performance measurements in an existing OpenFlow island, which was created in scope of the EU-funded project OFELIA [2].

The remaining of the paper is organized as follows. The next section gives a short overview about the OpenFlow technology and the main building blocks for the implementation of the new load balancing concept. Section III describes the architecture of the proposed load balancing concept, while section IV refers to the implementation details of the prototype. Finally, Section V summarizes the results of the experimental measurements within the OpenFlow island.

II. Background and Related Work

The OpenFlow [1] approach aims to create a vendor independent, standardized interface for adding and removing flow entries on a switch. Thus, the networks flows can be defined dynamically based on the current state of the network and the expected load. The implementation is based on an API, which allows the modification of the FlowTables representing the forwarding decisions of a switch. The FlowTables consist of flow entries [3] that compile the mapping between a header information and the action to be executed as well as a counter for each entry. The header is the pattern for the matching process against the packet-header. Typical actions are for example switching or routing operation like a packet drop, forward or rewrite and forward to a port or ports. For every match of a packet and the flow entry, the counter is incremented and thus allows a statistical analysis of the usage

978-1-61284-0833-6/12/$26.00 ©2012 IEEE

of the switches, ports or flows. This information is used in the next adaptation step, so the network configuration follows the network conditions.

The OpenFlow protocol offers control of a switch on the ISO layers 1 to 3. A network administrator defines the flows and let them process with full line rate on the switching hardware. An OpenFlow controlled switch is represented by a packet forwarding hardware acting on the base of external controlled demands. The basic approach of this procedure is to have a centralized control instance where all switches are connected with. This allows to have a single entry point for the whole network administration where all forwarding decisions of the packet flows are set.

Currently, there are several OpenFlow controllers available e.g. Beacon [4], Trema, Maestro [5] and SNAC [6]. Some are Open-source and others are commercial and proprietary. In general it is possible to program an own controller using the OpenFlow specification [3]. NOX [7] is an Open-source controller platform which provides a plug-in interface. It was developed at the Stanford university. This allows researchers a fast implementation process of specialized controller-modules in C++ or Python which work on top of NOX using the NOX library. These individual controller modules can deal with traffic that uses new protocols or handle innovative layer two routing algorithms for example.

FlowVisor [8] is an application that acts as a policy based OpenFlow controller proxy. It creates the possibility to slice the network resources and use a different controller for every slice. The input and output interface of FlowVisor is a socket-connection speaking the OpenFlow protocol. This makes it transparent for the controllers and switches. The application itself is controlled via a dedicated XMLRPC or CLI interface. The slices or rather flow spaces can be defined and configured with these interfaces. The slicing mechanism creates the option for using different controllers on different parts of the network, see also figure 2. Every slice is assign to an own controller. Slices can be defined over the full spectrum of the OpenFlow API. This means from a physical port base layer (L1) up to the source or destination port (L3) of a TCP packet.

Thus a special controller for every kind of network traffic could be implemented to optimize or generally selectively affect the packet forwarding process in a slice. This also includes the option of different packet routes depending on the kind of traffic or purpose i.e. research or productive.

Deploying OpenFlow to datacenters provides benefits which enhance different requirements i.e. scalability [9]. Existing publications dealing with the thematic and advantages of load-balancing with OpenFlow using HTTP [10], [11] traffic. In a variety of scenarios they analyse the advantages and performance aspects using a single switch [12] or a network of switches to demonstrate the new processing technique and its opportunities.

All these concepts deal with a single type of traffic which is balanced with comparing the different algorithms to serve the servers. But in a data center the common use-case is to balance different services with different demands. So the primary objective for us is to focus on a new concept for balancing different kinds of traffic in a server network.

III. Architecture

The key idea of OpenFlow load balancing concept for multiple services is the replacement of the expensive and statically defined hardware components in data centers through OpenFlow controllers using the local network infrastructure for implementation of the load balancing strategies. The architecture is based on multiple controllers – one controller per service network – that forward the incoming messages to the target servers. Figure 1 shows the generalized physical topology approach of a local OpenFlow server network which is controlled with different NOX controllers using the FlowVisor slicing tool.

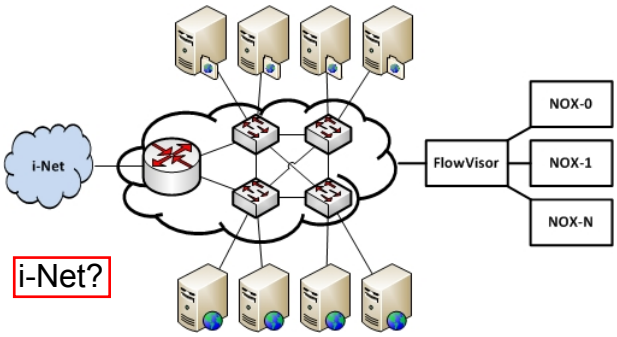

Fig. 1. Topology structure concept for multiple service load-balancing with OpenFlow

The distributed and dedicated nature of the controllers offers the chance to adapt the balancing policies to a-priori known processing pattern but also to the current load of the network. Moreover, a fail-over functionality can be implemented easily, as the controllers can take over the management for two or more service networks, if necessary.

Even more the controller has knowledge about the connection of the switches and servers and can provide features like OSPF and port-channels to improve the workload and network performance. In this concept it is feasible to distribute service requests to a single IP up to a subnet and forward them to the responsible server. One of the main advantages of this concept is that OpenFlow provides a native statistic environment for the controller implementation which is used as a load balancer. With the OpenFlow counters and a kind of server agent that collects load information of the services it is possible to implement an algorithm which have a deterministic knowledge of all network components reaching form all routing and switching nodes and their capacity up to the workload of the services in the server arrays. Another interesting thematic is the granularity of this concept. The default approach for granularity in a network is the separation into different subnets. If this is achieved every subnet could use an own FlowVisor with the associated load-balancing controllers. But in general FlowVisor could be used recursive which still provides a centralized OpenFlow control point with the option for extensions.

The figure 2 shows an example of a sliced network with four slices, each with two servers handling one service. In this case the servers of a slice are neighbors but generally the slicing technique allows any possible combination between them. This also allows the flexible extension of the system. The slices are defined with the FlowVisor application and each slice is controlled with an own controller. It is also possible to control every switch with an own controller but that creates a decentralize network as today. However, in this concept it is possible to use a flexible combination of distributed servers and switches or parts of switches building slices for every service and his controller or rather load balancer. This helps also distributing the forwarding workload from one device to several devices which improves numerous critical aspects as mentioned before the reliability.

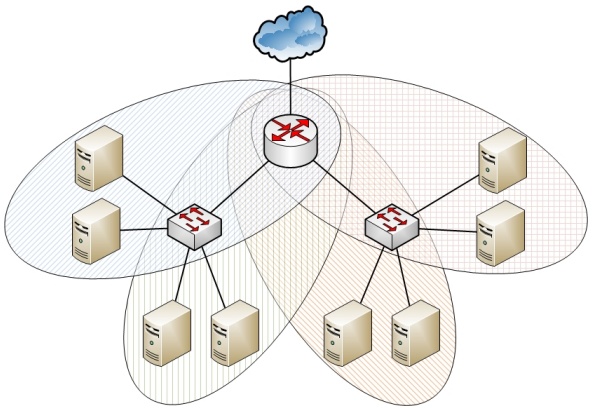

Fig. 2. Simple network example with four slices.

The architecture uses a NOX-based controller which implements a load-balancing algorithm. This controller works on a network slice defined by FlowVisor. If a new service connection request is detected by e.g. the ingress port of the top of rack switch or main router, depending on the granularity of the implemented model, it is passed to the FlowVisor application. The, the FlowVisor decides depending on the header information of the packet and his slice policies to which controller it has to be forwarded.

For example, an incoming packet with the destination port 80 is forwarded to the controller which handles the HTTP load balancing. Generally, new services request respectively flow request is delegated by FlowVisor to the corresponding controller. The controller sends the indication of the FlowTable entries back to FlowVisor where they are forwarded to the corresponding switches in the OpenFlow network. The indication contains the packet header rewrite information as well as the forwarding path. This process defines the rules for the switched forwarding of the flows through the local network. The path itself depends on the balancing algorithm as well as the switch where the header rewrite is processed due to the available resources and their distribution.

Table I and Table II are examples of the packet rewrite which is done in both directions. At the first time on an incoming packet and the second time on an outgoing packet.

Src. IP	Dst. IP	NewDst. IP	Ctl
1.2.3.4:12345	5.6.7.8:80	10.0.0.10:80	A
2.3.4.1:23451	5.6.7.8:80	10.0.0.11:80	A
3.4.1.2:34512	5.6.7.8:21	10.0.1.10:21	B
4.1.2.3:45123	5.6.7.8:80	10.0.0.12:80	A
1.2.4.3:51234	5.6.7.8:21	10.0.1.11:21	B
...	...	...	...

TABLE I
INCOMING PACKET HEADER REWRITE WITH CONTROLLER MAPPING PATTERN

Src. IP	Dst. IP	NewSrc. IP	Ctl
10.0.0.10:80	1.2.3.4:12345	5.6.7.8:80	A
10.0.0.11:80	2.3.4.1:23451	5.6.7.8:80	A
10.0.1.10:21	3.4.1.2:34512	5.6.7.8:21	B
...	...	...	...

TABLE II
OUTGOING PACKET HEADER REWRITE

This is done, for example, in the ingress/egress switching node of the server network and is processed with line rate on the hardware with the flow-table information that the controller has set at the first flow request. For clients form the internet it looks like all request are handled by one server independent from the requested service. Furthermore it is also possible to handle request to different IP's with the same method. Table I also shows the mapping of the requested service, represented by the port number, and the controller with the specialized balancing algorithm which delegates the request to the responsible server array.

IV. IMPLEMENTATION

For the first evaluation we implemented a round robin load-balancing algorithm as a plugin for the OpenFlow controller NOX. This plugin is implemented in c++ and adapts the NOX platform. To get a loadbalancer as proposed the plugin have to advice the switch to act as a device with different interfaces. This is done by answering ARP requests for the associated ip's of this device and delegate them to the next host out of the server pool by replacing the ip of the endhost with the device ip and mac. In the second step this is also done with the incoming IP connection as described before.

The implemented NOX-plugin is structured into three main parts. In the first part all packets where doped which should not be processed like LLDP packets. In the second part ARP request where handled for simulating the virtual loadbalancing device by directly sending packets out of the switch back to the requesting device. This operation is processed by the plugin itself. In the third and last part all traffic between the clients and server hosts where handled by rewriting MAC and IP address of the source and destination. Therefore a linked list is used where all processed client request are temporarily stored with their ip, port and server destination ip until the server answer rewrite is done. This process is established by setting up flow modification entry's for a forwarding between client and server with partial replacement of the Layer two

LLDP packet?
Link Layer Discovery Protocol

and three header. The whole forwarding and replacement operation is done with five OpenFlow actions, consisting of the particular layer two and three rewrite actions as well as an output action. This process is done in the switch by sending a flow modification entry from the plugin to the switch which ends up in the corresponding flow table entry. The plugin currently uses an idle flow timeout value of five seconds and a permanent hard timeout which means that a flow only times and is removed out of the flow table if the connection in not used for five seconds. For testing and scientific measurements this this is a good value but in production usage it should be adapted to the service for e.g. increasing performance or keep the session status alive.

V. Experimental evaluation

We tested the proposed software defined networking model using the local OpenFlow island created at the TU Berlin in the scope of the EU-funded project OFELIA. The available hardware includes three NEC IP8800/ S3640-48TW switches and one HP5400 switche as well as an Ixia T1600 testing system. The NEC switches are OpenFlow hardware processing capable and use a productive firmware image. On the other hand the HP work with a research firmware image which only processes L2 type 0x0800 (IPv4) forwarding in hardware but not the required header rewrite which is processed in software. Both switches work with OpenFlow version 1.0.

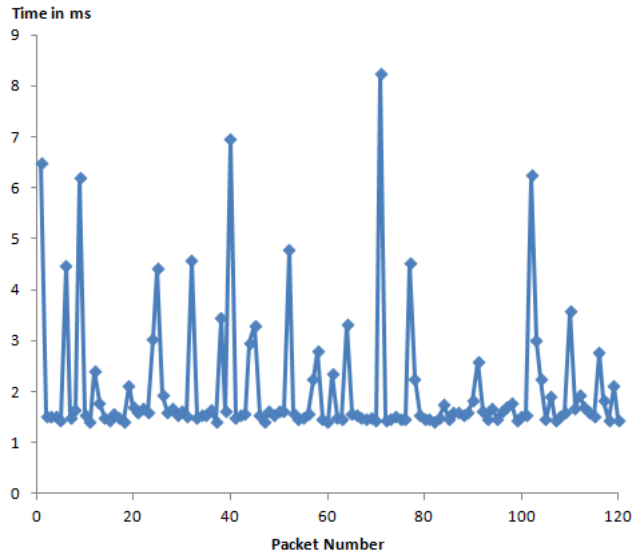

Fig. 3. Latency

The experimental evaluation is focussed on the OpenFlow performance of network elements more precisely an NEC OpenFlow switch and the feasibility of the proposed concept. The measured values are collected with the load-balancer controller implementation described in section IV on FlowVisor based slices.

The first measurement is collected as a reference in a port based slice with no other traffic. Figure 3 shows the latency between two virtual XEN machines in the slice which are

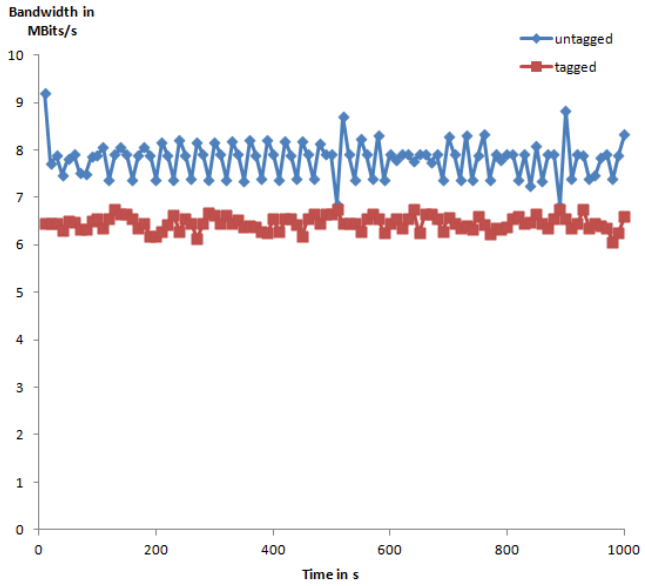

Fig. 4. Bandwidth of the refernce LB performance in one slice

physically connected to the NEC switch over Linux bridges. Figure 4 shows the bandwidth between these hosts. The latency values were measured with ICMP packets and the bandwidth with iperf tool and a TCP connection setup. The second values showen in figure 5 are collected to verify the functionality of the proposed model using two load-balancers in two vlan based slices sharing the same ports.

Latency:		2.066 ms
Bandwidth:	untagged	7.89 MBit/s
	tagged	6.46 MBit/s

TABLE III
Measurement of average values in a single slice

Bandwidth:	slice1	2.83 MBit/s
	slice2	3.53 MBit/s
	sum	6.36 MBit/s

TABLE IV
Measurement of average values with parallel LB in two slices

The measured performance of the reference is only round about one percent of the performance which is deliverd by the nox switching plugin. The conclusion of this is that currently neither NEC nor HP switches provide OpenFlow hardware processing support for layer three actions. In general it would be very interesting to test the implementation on an full hardware supported OpenFlow platform which is currently not available.

This implementation provides a complete NAT based load-balancing with destination and source NAT. An other option is the flat load-balancing which works only with destination NAT where the Load-balancer have to be the network gateway which is in conflict with the distributed approach of this paper.

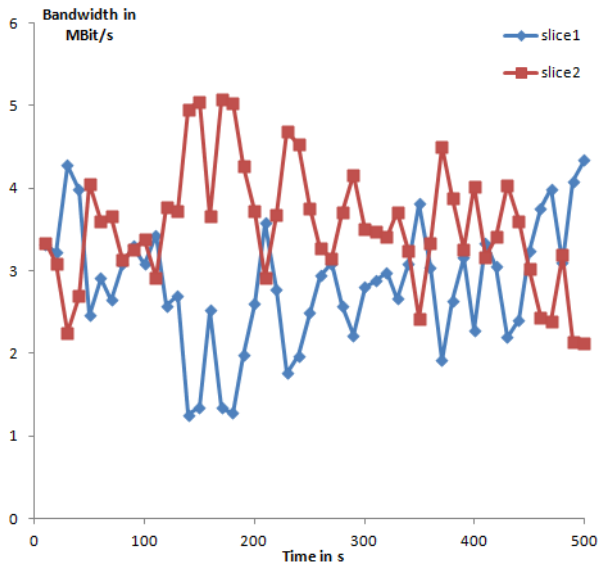

Fig. 5. Bandwidth with two slices using one load-balancer per slice

In that case the controller-switch combination can reduces the rewrite complexity by the four byte long client IP but have to handle the whole routing instead.

VI. Conclusion

This architecture combines different innovative solutions in one concept and transfer them to a new technology. From the opportunity of port filtering to the handling of different service depended load-balancers and everything works on the top of the normal switched network infrastructure. This has several advantages:

- no additional hardware
- dedicated and specialized balancing algorithm implementations depending on the requirements of services and workloads
- flexible extensions of server hardware
- flexible extensions of services
- complete modular controller (load-balancer) structure
- redundant network paths for reliability
- enhancing network scalability
- enhancing network granularity

Summarizing these issues it is an interesting concept for improving today's data centers and port them to an innovative future internet architecture.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, April 2008.

[2] "Openflow in europe: Linking infrastructure and applications," October 2011, http://www.fp7-ofelia.eu/.

[3] T. O. Consortium, "Openflow switch specification / version 1.1.0," February 2011, http://www.openflow.org/wp/documents/.

[4] "Beacon: A java-based openflow control platform." October 2011, http://www.beaconcontroller.net/.

[5] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A system for scalable openflow control."

[6] "Snac," October 2011, http://www.openflow.org/wp/snac/.

[7] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, July 2008.

[8] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *Technical Report Openflow-tr-2009-1, Stanford University*, July 2009.

[9] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying nox to the datacenter," *Eighth ACM Workshop on Hot Topics in Networks*, 2009.

[10] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: Load-balancing web traffic using openflow," *ACM SIGCOMM Computer Communication Review*, 2009.

[11] R. Wang, D. Butnariu, and J. Rexford, "Openflow-based server load balancing gonewild," *In Hot-ICE*, 2011.

[12] H. Uppal and D. Brandon, "Openflow based load balancing."