Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 34 (2014) 661 – 667

Procedia Computer Science

International Workshop on Software Defined Networks for a New Generation of Applications and Services (SDN-NGAS-2014)

Providing Infrastructure Functions for Virtual Networks by Applying Node Plug-in Architecture

Yasusi Kanada*

Central Research Laboratory, Hitachi, Ltd., Totsuka-ku Yoshida-cho 292, Yokohama 244-0817, Japan

Abstract

Although nodes in a network-virtualization infrastructure, which is called a virtualization node, usually contain a switch or a router with sophisticated and high-performance functions such as Ethernet switching, VLAN, and IP routing, most of such infrastructure functions cannot be reused as program components by slices. Accordingly, a method for providing such functions to slices on a virtualization node (VNode) infrastructure, by applying the previously proposed plug-in architecture, is proposed. This architecture defines two types of plug-ins, i.e., control plug-ins and data plug-ins, and interfaces for them. As for the proposed method, the switch or router in the VNode is regarded as a data plug-in, and a control plug-in that allocates and isolates the switch/router resources was developed. The data plug-in interface was customized to handle a data plug-in, i.e., a layer-3 switch in a VNode, and a control plug-in and the interfaces for providing layer-3/VLAN switch functions to slices were designed, implemented, and evaluated. The evaluation result shows that instead of specifying a routing/switching program or method, specifying only an additional 8 to 25 lines in a slice definition enables slice developers to use routing and switching functions.

Selection and peer-review under responsibility of Conference Program Chairs

Keywords: Network-node plug-in architecture; Network virtualization; Virtualization node; VNode; In-slice switching; In-slice routing; Deep programmability

* Corresponding author. Tel.: +81-50-3135-3485.
E-mail address: Yasusi.Kanada.yq@hitachi.com .

Selection and peer-review under responsibility of Conference Program Chairs
doi:10.1016/j.procs.2014.07.094

1. Introduction

Various infrastructures for network virtualization, such as PlanetLab [Tur 07], GENI [Ber 14], and VNode (virtualization node) [Nak 10][Nak 12], have been developed. By using such a network-virtualization infrastructure (i.e., substrate), slices (i.e., virtual networks) are created and managed. These infrastructures enable flexible node functions by providing *deep programmability*. They provide programmability of both slow paths (i.e., general-purpose CPU-based hardware) and fast paths (i.e., specialized hardware such as network processors). However, although nodes in the infrastructure usually contain a switch or router with sophisticated and high-performance functions such as Ethernet switching, VLAN, and routing, most of these functions cannot be reused by slices.

For the VNode infrastructure, an Ethernet switching function is provided to slices by "network accommodation equipment" (NACE or NC) [Kan 12b]; however, the functions of NACE are restricted. NACE enables high-speed and large-capacity Ethernet switching in slices, but it is very specialized and does not provide any other functions.

This paper proposes a method for providing various infrastructure functions to slices on the VNode infrastructure by applying the previously-proposed plug-in architecture [Kan 13][Kan 14a]. This architecture defines two types of plug-ins, i.e., *control plug-ins* and *data plug-ins*, and plug-in interfaces for them. As for the proposed method, the switch or router in the VNode is regarded as a data plug-in, which is called an *internal plug-in*, and a control plug-in that allocates and isolates the switch/router resources is developed. A method for providing layer-3 (L3) or VLAN switch functions (i.e., routing/switching functions) to slices was developed, implemented, and evaluated. Using the control plug-in, operators and vendors can provide infrastructure functions easily, and slice developers can specify functions implemented by the plug-ins very simply in slice definitions.

2. Related Work

Two categories of related work on generic network-node programmability and programmability of node functions, such as routing or switching, are summarized here. The first category is generic network-node programmability. A network-node function may be extended by using plug-ins and a protocol such as the ForCES [Dor 10] or OpenFlow. The protocol separates centralized control-plane components from the data plane and defines interfaces between these planes. In contrast, the proposed plug-in architecture defines distributed and corresponding control-plane and data-plane components, i.e., plug-ins, and interfaces between plug-ins and a network node. In addition, OpenFlow is not suited to handling non-standard headers, but the proposed method can be used for them.

The second category is node-function programmability. Providing network-node functions, such as switching or routing, for slices by reusing infrastructure functions is focused on. As described by Casado [Cas 08], methods for providing network-node functions have been categorized as the following three types. First, integrated-hardware-based methods are conventionally used for commercial routers and switches. Second, software-centered methods have been widely studied (e.g., [Egi 08]) because they make it easy to implement packet forwarding and routing. Network-processor (NP)-based methods can be categorized as the second type, but it may achieve higher performance. The third type is general-purpose hardware-based methods. An example of this type is the TCAM-based method proposed by Casado. This method was succeeded by an OpenFlow-based method (e.g., [Boz 13]).

As for the developed method, an L3/VLAN switch and software are used in combination. This method is software-centered, but it can be regarded as a method that extends integrated hardware, i.e., an L3/VLAN switch.

3. Outline of Plug-in Architecture

The basic VNode architecture, which enables computational and networking components of VNodes to be independently developed, is reviewed in the following. A VNode [Nak 12][Kan 12a] is a type of physical node for building a virtualization network infrastructure, which is called a VNode infrastructure. Virtual-network functions in a VNode are categorized as *computational functions* or *networking functions*, which are represented by built-in types of "node slivers" (i.e., virtual nodes) and "link slivers" (i.e., virtual links). A slice developer can define a slice by writing a *slice definition* (a similar concept to resource specifications (RSpec) [Ber 14]) described in XML and sent to the manager of the VNode infrastructure (i.e., the domain controller [Kan 12a] or VNet manager [Kat 13]).

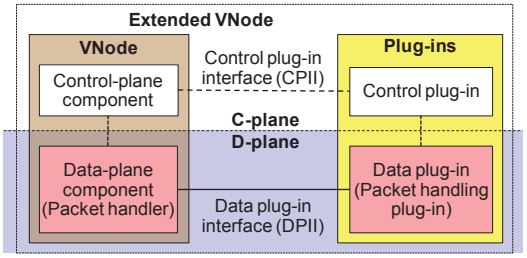
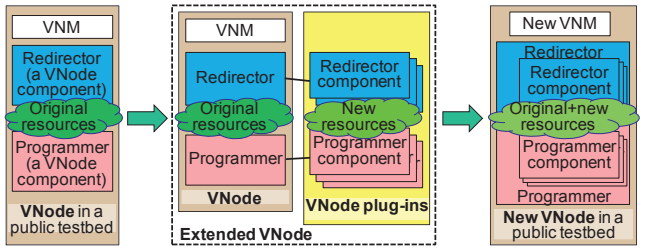


Fig. 1. Open VNode plug-in architecture



(a) Original stage (b) Experimental stage (c) Operational stage

Fig. 2. Proposed VNode evolution stages

The VNode plug-in architecture and interfaces were previously proposed [Kan 13][Kan 14a]. Plug-ins can extend both node and link functions of a VNode. There are two types of plug-ins: *data plug-ins* (which perform data-plane functions such as packet forwarding) and *control plug-ins* (which manage the resources and configurations of a data plug-in). A plug-in consists of hardware and software. A data plug-in may contain specialized hardware required for guaranteeing high performance, isolation, and QoS of slices. Plug-ins are connected to a VNode by predefined control/data plug-in interfaces (CPII/DPII), which should be built into the VNode (**Fig. 1**).

The operator or vendor of a VNode infrastructure can develop new functions by using a set of plug-ins, and slice developers can then easily use them by specifying plug-ins or new types of nodes or links. Each development scenario should have two stages, i.e., an *experimental stage* and an *operational stage* (**Fig. 2**). In the experimental stage, the operator does not need to update VNodes; instead, slice developers must specify plug-ins and plug-in parameters. However, in the operational stage, the operator authorizes and installs a set of plug-ins, and the slice developers can define a new type of virtual node or link by using the same method as that used for the built-in types.

4. Application of Plug-in Architecture

The method for applying the VNode plug-in architecture to implement infrastructure functions is described in this section. Especially, the interfaces for connecting an internal plug-in (i.e., part of the switch/router in the VNode) to a VNode and the method for specifying virtual nodes or links that the plug-in implements are described.

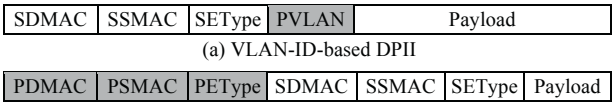
4.1. Requirements for plug-ins and plug-in interfaces

A set of internal/external plug-ins that implement a virtual node or link function must satisfy the following three conditions. First, control plug-ins must implement the CPII. Second, data plug-ins must implement the DPII. Third, the most-important condition, is that control plug-ins must manage the resources of virtual nodes or links and isolate resources of multiple instances of virtual node or link type. For example, because an L3 switch is not usually designed for network virtualization, the control plug-in must map VLAN/IP resources to slices and manage per-slice resources in order to isolate resources and slices. However, because the switch or router to be used as an internal plug-in is usually fixed, that is, the components of a VNode cannot be replaced, the DPII must be flexible to connect internal data plug-ins. In the case of L3 switches, the required interface is VLAN-based. If the DPII is properly defined, the functions of the switch/router, such as switching and routing, can be provided to slices.

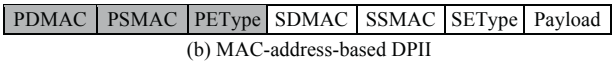
4.2. Plug-in interfaces for internal plug-ins

Unlike the cases described in previous papers [Kan 13][Kan 14a], to use an L3 switch as a data plug-in, all the ports of virtual nodes must be identified by VLAN IDs instead of MAC addresses. VLAN IDs are used for port identification (and for logical isolation of slices) because, if two ports have the same VLAN ID, packets do not pass through the virtual node but are bypassed through the L3 switch.

The packet format used for this VLAN-ID-based



(a) VLAN-ID-based DPII



(b) MAC-address-based DPII

Fig. 3. Comparison of two types of data plug-in interface (DPII)

DPII is compared with that used for the normal MAC-address-based DPII in **Fig. 3**. Each field in the figure is explained as follows. SDMAC and SSMAC are the MAC addresses on the slice, and SEType is the Ethernet type of the slice. PVLAN is the value for identifying the VLAN path. The field with PVLAN exists between fields of the in-slice Ethernet; however, it belongs to the virtualization infrastructure. The whole packet format is conformant to the VLAN (Ethernet) standards. PDMAC and PSMAC are the MAC addresses on the infrastructure, and PEType is the Ethernet type of the infrastructure. As described above, the PVLAN field is squeezed into the packet on the slice. If the DPII is MAC-address-based, the format of packets on slice is not restricted, i.e., it can be non-IP and non-Ethernet. However, if the DPII is VLAN-ID-based, it can be non-IP but must be an Ethernet format.

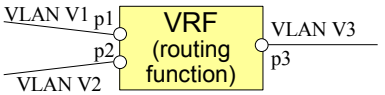


Fig. 4. Plug-in specification of routing function

5. L3 Switch Functions to be Provided to Slices

Methods for providing two important functions of L3/VLAN switches, i.e., routing and switching, to slices by using internal plug-ins are described in the following.

5.1. Routing function

When IP routing and forwarding is required on a slice, it is advantageous to implement it by using an internal plug-in that uses the L3 switch functions. The control-plane function, i.e., building a routing table, is suited to software, but the data-plane function, i.e., packet forwarding by using the routing table, may require high-performance processing, so specialized hardware is often required. By using the virtual routing and forwarding (VRF) function, which is a per-VLAN IP-routing function that many L3 switches have, the plug-ins can provide a high-performance routing function to each slice. An in-slice routing function can be visualized as shown in **Fig. 4**.

```
<nodeSliver name="vrf1" ...>
 <instance type="extension">
  <params>
   <param key="PlugInName" value="intSw" />
     <!-- Plug-in name -->
   <param key="DataPort" value="vlan" />
     <!-- Specification of data plug-in interface (DPII) -->
   <param key="ControlPort" value="192.168.110.61" />
     <!-- Specification of control plug-in interface (CPII) -->
   <param key="Command-runNodeSliver" value="run vrf" />
     <!-- Specification of CPI command for configuration -->
   <param key="Command-stopNodeSliver" value="stop vrf" />
     <!-- Specification of CPI command for de-configuration -->
   <!-- Other paramaters: -->
   <param key="P1" value="V1" /> <!-- A routing parameter -->
   ...
   <param key="Pn" value="Vn" /> <!-- A routing parameter -->
  </params>
 </instance>
 <vports>...</vports>
</nodeSliver>
```
Fig. 5. Slice definition for in-slice IP routing in the experimental stage

A slice definition for a virtual router is explained. **Fig. 5** shows a simple definition in the experimental stage. In the figure, the routing parameters are specified as *P1=V1*, …, *Pn=Vn*. Routing protocols such as RIP or BGP can be specified, but OSPF is specified here. The following routing parameters can be given.

```
<param key="routing_protocol" value="ospf" /><param key="ospf_subnet" value="192.168.0.0" />
<param key="ospf_mask" value="0.0.15.255" /><param key="ospf_area" value="110" />
<param key="ospf_domain" value="1" /><param key="router_ip" value="192.168.101.1" />
```

These parameters specify the routing protocol, i.e., OSPF, the network that OSPF runs, the area, the domain, and the IP address of the virtual router. VLAN IDs are assigned to ports p1, p2, and p3 by the same method as that used by the switch plug-in. The IP addresses of these ports can be selected from the link subnet if a method for negotiation between control plug-ins of two VNodes is implemented [Kan 14b]. However, currently, the addresses must be specified explicitly by the following parameters.

```
<param key="ip_p1" value="192.168.1.0/255.255.255.0" />
<param key="ip_p2" value="192.168.2.0/255.255.255.0" />
<param key="ip_p3" value="192.168.3.0/255.255.255.0" />
```

In specifying this definition, the slice developer must avoid inconsistent configuration of the slice, which can be avoided when the addresses are automatically negotiated.

In the operational stage, the implementation-dependent name and parameters are not specified in a slice definition. However, the specification of the routing protocol, i.e., OSPF, and OSPF parameters, must be specified. The slice definition corresponding to Fig. 5 is given as follows.

```
<nodeSliver name="vrf1" ...>
  <instance type="virtual router">
    <params><param key="P1" value="V1" /> ... <param key="Pn" value="Vn" /></params>
  </instance>
  <vports>...</vports>
</nodeSliver>
```

When introducing a three-port virtual router into a slice, a slice developer must specify 25 lines in the experimental stage and 19 lines in the operational stage except comment lines to specify a virtual switch. Note that comments in the slice definition are not counted in the lines.

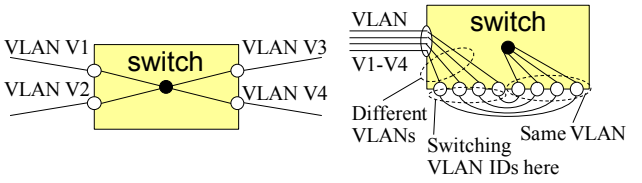
5.2. Switching function

Packet switching, which is often required on slices, can be implemented by using an internal plug-in. Although software switching can be used for packet switching, a VLAN switch function is used because it is superior in terms of capacity and latency. Because a VLAN switch has a per-VLAN switching function, per-slice switching can be implemented by mapping slices to VLANs. A virtual switch can be visualized as shown in **Fig. 6**(a).

Although a virtual switch is conceptually simple, the actual method for implementing it is complicated because of restrictions on VLAN switch functions. Because an internal virtual-switch plug-in is connected to the data-plane component of the VNode, i.e., the VLAN switch, it would be better to connect them internally in the switch, if possible. However, although packets that belong to the same VLAN are switched among different physical ports in a VLAN switch, packets must be switched among different VLANs on the same physical port. The packets of the VLANs to be connected to the virtual switch must thus be extracted from the VLAN switch and input into it again without changing the VLAN ID (see Fig. 6(b)). This method is the same as the one used for NACE [Kan 12b].

The slice definitions for the virtual switch are similar to those for the virtual router, so detailed explanation is omitted. However, the command names (**`run_sw`** and **`stop_sw`**) or the name of the virtual-switch type (**`virtual_switch`**) are used instead of **`run_vrf`** / **`stop_vrf`** or **`virtual_router`**. To specify a virtual switch, a slice developer must specify 16 lines in the experimental stage and 8 lines in the operational stage except comment lines. When the specified DPII (DataPort) is a VLAN, the management components of the VNode assign VLAN IDs to all the ports specified by **`<vports>`**…**`</vports>`**. When ports p1, p2, …, pn are specified and VLAN ID v1, v2, …, vn are assigned to the ports, pairs of the ports and VLAN IDs are passed as a command parameter to the plug-in as command arguments of **`run_sw`** and **`stop_sw`**.

6. Prototyping and Preliminary Evaluation



(a) Data plug-in interface (b) Implementation in VNode

Fig. 6. Plug-in specification and implementation of virtual switching function

To prototype and evaluate a virtual routing function by using an internal plug-in, the CPII and a control plug-in using this interface were designed, implemented, connected to the L3 switch, and preliminarily evaluated. The plug-in interfaces for the internal plug-ins are being implemented in a NACE, and a control plug-in (written in Perl) for both switching and routing is being developed. The control plug-in receives parameters from the NACE through a command-line interface (CLI, i.e., CPII) and configures the switch by using another CLI (telnet). Because the L3 switch is configured by both the management component of the NACE and the control plug-in, it must be configured very carefully; that is, resources in the L3 switch must be isolated by careful programming of both the control plug-in and the NACE because they are not isolated automatically by the plug-in architecture. However, slice developers do not have to be concerned with this isolation.

The development of in-slice routing by using plug-in is explained as follows. The syntax of the command for slice configuration is as follows.

```
run_vrf SliverId=<VirtNodeID> PlugInName=intSw ports=<PID>:<PVID>{+<PID>:<PVID>} \
   {ip_<PID>=<PIPaddress>/<PSubnet>} routing_protocol=ospf ospf_subnet=<RSubnet> \
   ospf_mask=<RSubnetMask> ospf_area=<OSPFArea> ospf_domain=<OSPFDomain> router_ip=<RouterIP>
```

Commands "**`run_vrf`**" and "**`stop_vrf`**", which are specified in **Fig. 5**, are processed by the same program; that is, in this plug-in implementation, the configuration and de-configuration are handled by the same program. The data

plug-in specified by "**intSw**" implements both virtual switching and routing. Parameter **ports** specifies pairs of ports of the virtual node and VLAN IDs assigned to the ports. The IP addresses and subnet masks are specified by parameters **ip_**<PID> (i.e., **ip_1**, **ip_2**, and **ip_3**). Parameters prefixed by "**ospf_**" are effective only when OSPF is used for routing. The control plug-in converts the above parameters to a command and configures the L3 switch. For example, the following command generates a VRF (virtual router) that corresponds to virtual node **vrf1**, generates and configures its three ports, and configures OSPF.

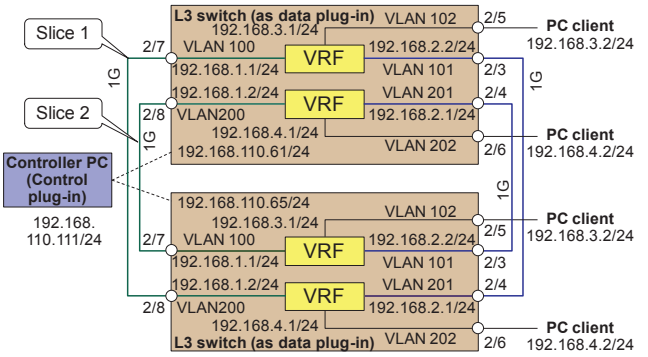


Fig. 7. Network structure for in-slice OSPF routing experiment using two L3 switches

```
run_vrf SliverId=vrf1 PlugInName=intSw ports=port1:100+port2:101+port3:102 \
  ip_port1=192.168.1.1/255.255.255.0 ... ip_port3=192.168.3.1/255.255.255.0 \
  routing_protocol=ospf ospf_subnet=192.168.0.0 ospf_mask=0.0.15.255 ospf_domain=1 \
  ospf_area=110 router_ip=192.168.101.1
```

The in-slice routing was evaluated as follows. By using the control plug-in described above, OSPF routing on a slice on a network with two L3 switches (two VNodes), which is shown in **Fig. 7**, was tested. In this test, two L3 switches (as data plug-ins) were configured (as shown in this figure) by the control plug-in. OSPF routing with this configuration was confirmed to work correctly by the following two test results. First, ping packets were routed correctly between two PC clients either when they were connected to the first or second slice. Second, when one of the four physical links between two L3 switches was cut, the communication between the PCs was disabled, but it was recovered after 40 to 50 seconds.

7. Concluding Remarks

A method for providing sophisticated and high-performance functions, such as Ethernet switching and IP routing to slices on a VNode infrastructure, by using internal plug-ins is proposed. A previously developed data plug-in interface was customized to handle an L3 switch in a VNode, and a control plug-in, and the interfaces for providing L3 switch functions to slices, which are distributed to each node, were designed, implemented, and evaluated. The evaluation result shows that instead of specifying a routing or switching program, specifying only an additional 8 to 25 lines in a slice definition enables slice developers to use routing and switching functions. Because the implementation of plug-ins and an extended plug-in interface are in the draft stage, it must be finalized, the plug-in installation, authentication, and authorization mechanisms should be built into VNodes, and the plug-in implementation should be evaluated in future.

Acknowledgements

Part of the research results is an outcome of "Advanced Network Virtualization Platform Project A" funded by the National Institute of Information and Communications Technology (NICT). The author thanks Akihiro Nakao from The University of Tokyo, Satoshi Kamiya from NEC, and other members of the VNode Project for their discussions on virtual-switch interfaces. He also thanks Yasushi Kasugai, Kei Shiraishi, Takanori Ariyoshi, and Takeshi Ishikura from Hitachi for implementing the plug-in interfaces.

References

[Ber 14] Berman, M., Chase, J. S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., Ricci, R., and Seskar, I., "GENI: A Federated Testbed for Innovative Network Experiments", *Computer Networks*, Vol. 58, January 2014.

[Boz 13] Bozakov, Z. and Papadimitriou, P., "OpenVRoute: An Open Architecture for High-performance Programmable Virtual Routers", *IEEE HPSR'13*, pp. 191–196, 2013.

[Cas 08] Casado, M., Koponen, T., Moon, D., and Shenker, S., "Rethinking Packet Forwarding Hardware", *7th ACM SIGCOMM HotNets Workshop*, p. 11, October 2008.
[Dor 10] Doria, A., Hadi Salim, J., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and Halpern, J., "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, IETF, March 2010.
[Egi 08] Egi, N., Greenhalgh, A., Handley, M., Hoerdt, M., Huici, F., and Mathy, L., "Towards High Performance Virtual Routers on Commodity Hardware", *2008 ACM CoNEXT Conference*, p. 20, December 2008.
[Kan 12a] Kanada, Y., Shiraishi, K., and Nakao, A., "Network-Virtualization Nodes that Support Mutually Independent Development and Evolution of Components", *IEEE ICCS 2012*, November 2012.
[Kan 12b] Kanada, Y., Shiraishi, K., and Nakao, A., "High-performance Network Accommodation into Slices and In-slice Switching Using A Type of Virtualization Node", *IARIA Infocomp 2012*, October 2012.
[Kan 13] Kanada, Y., "A Node Plug-in Architecture for Evolving Network Virtualization Nodes", *SDN4FNS 2013*, November 2013.
[Kan 14a] Kanada, Y., "A Method for Evolving Networks by Introducing New Virtual Node/link Types using Node Plug-ins", *IEEE/IFIP SDNMO 2014*, May 2014.
[Kan 14b] Kanada, Y., "Controlling Network Processors by using Packet-processing Cores", *NetMM 2014*, May 2014.
[Kat 13] Katayama, Y., Yamada, K., Shimano, K., and Nakao, A., "Hierarchical Resource Management System on Network Virtualization Platform for Reduction of Virtual Network Embedding Calculation", *APNOMS 2013*, September 2013.
[Nak 10] Nakao, A., "Virtual Node Project", *NICT News*, No. 393, pp. 1–6, Jun 2010.
[Nak 12] Nakao, A., "VNode: A Deeply Programmable Network Testbed Through Network Virtualization", *3rd IEICE Technical Committee on Network Virtualization*, March 2012, http://www.ieice.org/~nv/05-nv20120302-nakao.pdf
[Tur 07] Turner, J., et al., "Supercharging PlanetLab — High Performance, Multi-Application, Overlay Network Platform", *ACM SIGCOMM Computer Communication Review*, Vol. 37, No. 4, pp. 85–96, October 2007.