A Method for Evolving Networks by Introducing New Virtual Node/link Types using Node Plug-ins

Yasusi Kanada
Hitachi, Ltd., Japan

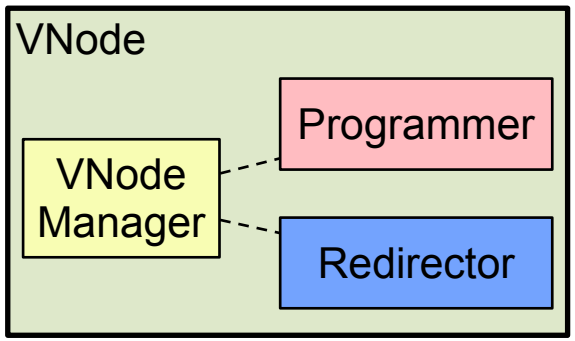
Introduction

- **Virtual networks (or slices) are suited for development of new services.**
  - Service providers develop new services as *slice developers*.
- **New services may be supported by evolution of nodes with network-virtualization functions, i.e., “VNodes”.**
  - VNodes mean physical network nodes (not virtual nodes).
  - New services requires new virtual-network functions, which may be supported by evolution of node software or hardware.
- **Two-stage evolution method for VNodes is proposed.**
  - **[1st stage]** VNodes are evolved by adding and updating plug-ins without modifying original components (previous paper).
  - **[2nd stage] Plug-ins are integrated into VNode (focus of this paper).**
  - This method was developed for *our VNodes*, but it is a generic method.

Our VNode and its Components

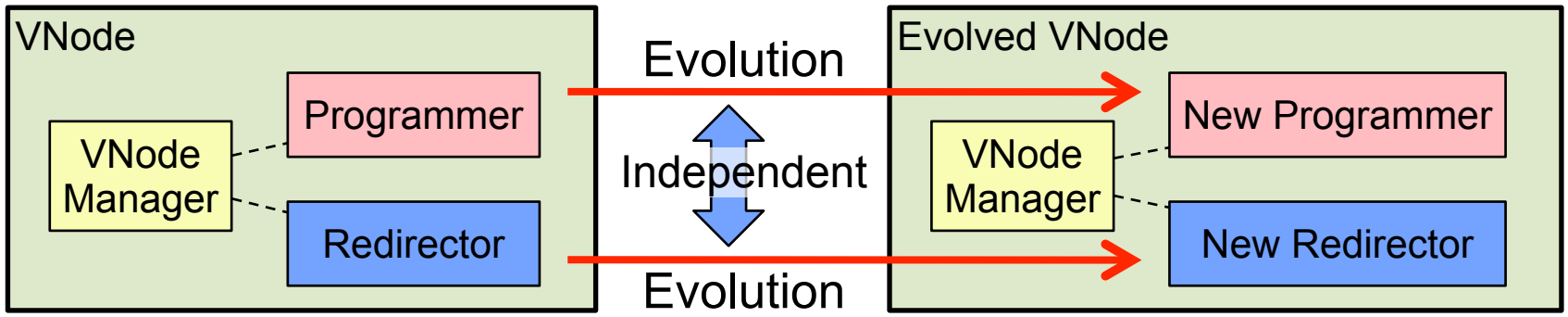
- **A type of VNode was developed by a collaborative project conducted by Professor Aki Nakao (U. Tokyo).**

- **This VNode consists of modular components:**
  - **Programmer**: a *deeply-programmable* computational component.
  - **Redirector**: a networking component.



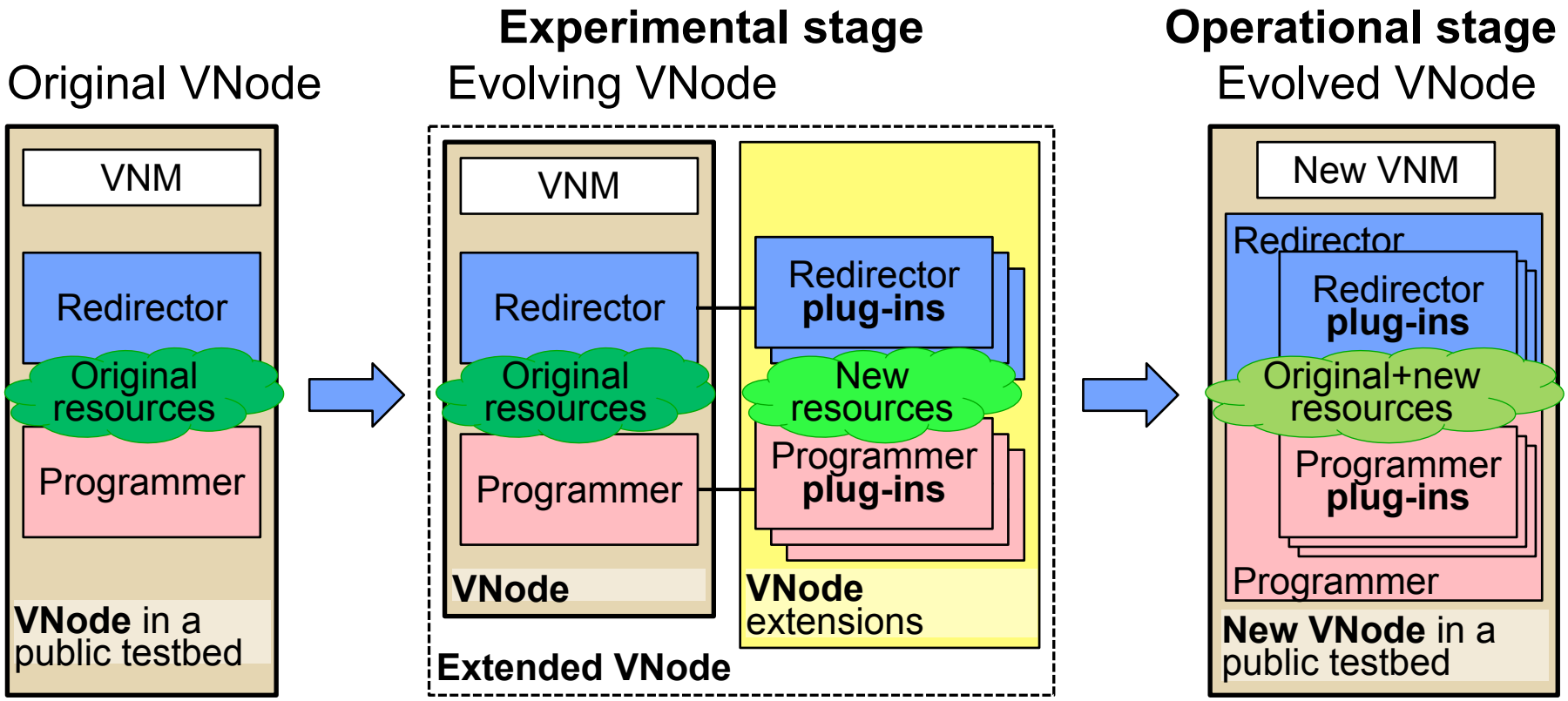
Evolution of VNode

- **VNode components may evolve by using *new node hardware* or *software*, or *new functions*.**
  - **New hardware/software**: New network processors, GPGPUs, New types of VMs, etc.
  - **New functions**: New protocol stacks, etc.
- **VNode components can evolve independently because they are *modular*.**



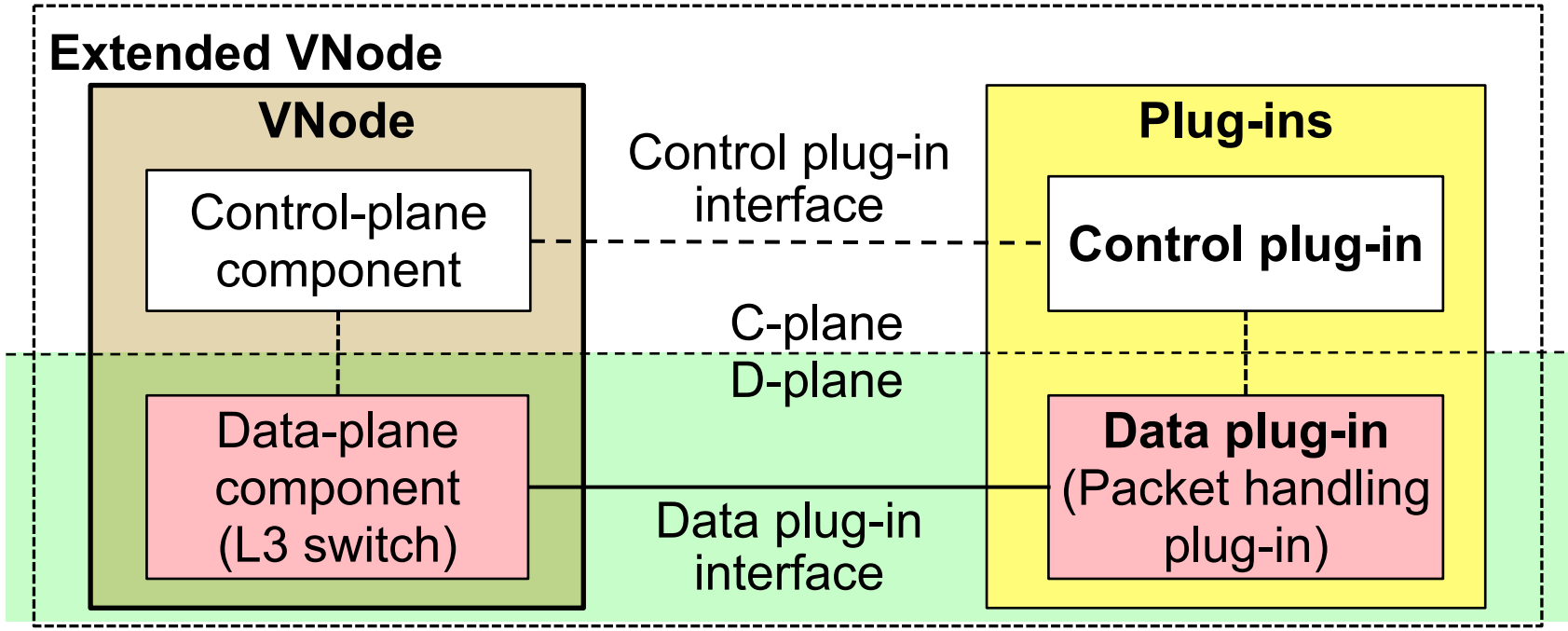
Proposed Method: Two-stage Evolution of VNode

- **Experimental stage: Operators/vendors develop new subcomponents as plug-ins w/o updating VNode.**
- **Operational stage: Operators/vendors merge the plug-ins into VNode and developers use the new functions.**



Experimental Stage: Plug-in Architecture

- **A data plug-in handles data packets.**
  - It consists of hardware and software.
- **The data plug-in is controlled by a control plug-in.**
  - New control functions required for the data plug-in is implemented by the control plug-in (which is assumed to be distributed).
- **Plug-ins are connected to a VNode.**



Experimental Stage: Virtual-node/link Spec

- **A developer specifies a new type of virtual-node/link with parameters that specify plug-in parameters.**

A virtual link specification (RSpec):

```
<linkSliver type="link" …>
  <vports>
    <vport name="vport1">
      <params>
        <param key="ControlPort" value="CPIl1-addr"/>
        <param key="DataPort" value="DPIl1-port"/>
      </params>
    </vport>
    <vport name="vport2">
      <params>
        <param key="ControlPort" value="CPIl2-addr"/>
        <param key="DataPort" value="DPIl2-port"/>
      </params>
    </vport>
  </vports>
  <params>
    <param key="PlugInName" value="virtual-li
    <param key="Command-reserveLinkSliver1"
           value="ls_setup_1"/>
    <param key="Command-reserveLinkSli
           value="ls_setup_2"/>
    <param key="Command-reserveLink
           value="ls_setup_3"/>
    <param key="Command-runSliver"
           value="ls_run"/>
    <param key="Command-shutdownSli
           value="ls_stop"/>
  </params>
</linkSliver>
```

So many implementation-dependent parameters must be specified!

- **VNode control components *tunnels* these parameters.**
  - They do not check everything in specifications.

Operational Stage: Requirements

- **Plug-in functions must be merged into the VNode.**
- **Requirements for operators and vendors**
  - Newly added modules (types) must be authenticated and authorized.
    - Plug-ins may contain “viruses”, which must be avoided in operational stage.
  - The implementation cost must be minimized.
    - To merge new functions to the original modules (i.e., tightly-coupled modules) may be too costly.
- **Requirement for developers**
  - New virtual-node/link types must be available in the same method (by the same simple syntax) as built-in types.

Operational Stage: Loosely-coupled Architecture

- **To reduce the cost, plug-ins are not modified.**
  - The data plug-in is still managed by the control plug-in.
- **Plug-in information is entered into the VNode and the network manager** (in plug-in parameter mapping tables).

Keys	Control plug-in	Data plug-in
node-type-1	*Control plug-in identifier, Authorization information, Commands*	*Data plug-in identifier, Authentication information*
VNode0	*CPIn0-addr*	*DPIn0-port*
VNode1	*CPIn1-addr*	*DPIn1-port*
VNode2	*CPIn2-addr*	*DPIn2-port*
link-type-1	*Control plug-in identifier , Authorization information, Commands*	*Data plug-in Authentica*
VNode0	*CPIl0-addr*	*DPIl0-p*
VNode1	*CPIl1-addr*	*DPIl1-p*
VNode2	*CPIl2-addr*	*DPIl2-p*

There are parameters to be distributed and centralized

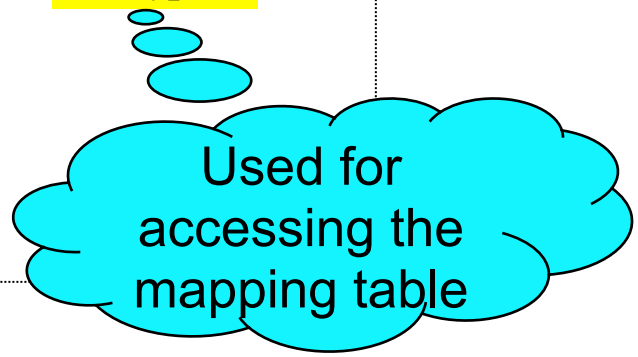
- **Plug-ins are authorized by the operator.**
  - The mapping table contains authorized plug-ins only.
- **Plug-ins are authenticated by the VNode.**
  - The operator registers the authentication information.

Operational Stage: Virtual-node/link Spec

- **Slice developers can specify a new virtual-node/link type using the same *simple syntax* as built-in types.**
  - Plug-in parameters are supplied by VNode control components by using the mapping table.

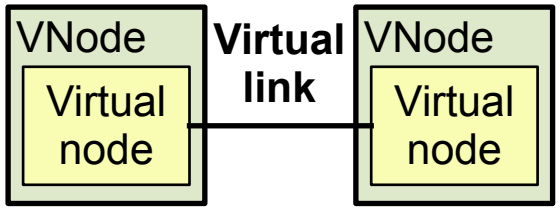
A virtual link specification (RSpec):

```
<linkSliver type="link" subtype="link-type-1">
  <vports>
    <vport name="vport1"/>
    <vport name="vport2"/>
  </vports>
  // No plug-in parameters
</linkSliver>
```



Prototyping

- **Set of plug-ins and new virtual-link type were developed.**
  - Experimental stage was evaluated in the previous paper.
  - For (simulated) operational stage, a preprocessor was developed.
- **The preprocessor**
  - contains a plug-in parameter mapping table.
  - converts virtual-link specifications.




```
<linkSliver type="link" subtype="link-type-1">
  <vports>
    <vport name="vport1"/>
    <vport name="vport2"/>
  </vports>
  // No plug-in parameters
</linkSliver>
```

preprocessing

Keys		Control plug-in	Data plug-in
node-type-1		*Control plug-in identifier , Authorization information, Commands*	*Data plug-in identifier, Authentication information*
	VNode0	*CPIn0-addr*	*DPIn0-port*
	VNode1	*CPIn1-addr*	*DPIn1-port*
	VNode2	*CPIn2-addr*	*DPIn2-port*
link-type-1		*Control plug-in identifier , Authorization information, Commands*	*Data plug-in identifier, Authentication information*
	VNode0	*CPIl0-addr*	*DPIl0-port*
	VNode1	*CPIl1-addr*	*DPIl1-port*
	VNode2	*CPIl2-addr*	*DPIl2-port*

```
<linkSliver type="link" …>
  <vports>
    <vport name="vport1">
      <params>
        <param key="ControlPort" value="CPIl1-addr"/>
        <param key="DataPort" value="DPIl1-port"/>
      </params>
    </vport>
    <vport name="vport2">
      <params>
        <param key="ControlPort" val…l2-addr"/>
        <param key="DataPort" …port"/>
      </params>
    </vport>
  </vports>
  <params>
    <param ke…tual-link-1"/>
    <param ke…LinkSliver1"
           va…up_1"/>
    <param key…Command-reserveLinkSliver2"
           value="ls_setup_2"/>
    <param key="Command-reserveLinkSliver3"
           value="ls_setup_3"/>
    <param key="Command-runSliver"
           value="ls_run"/>
    <param key="Command-shutdownSliver"
           value="ls_stop"/>
  </params>
</linkSliver>
```

The syntax is different from the final version.

- **VNodes can process the specifications w/o modification.**

Evaluation of Prototype

- **Comparison of virtual-link specifications**

Virtual-link specification	Definition length (lines)	Implementation-dependent parameters
Original (before preprocessing)	7	0
Translated (after preprocessing)	14	4

- **Verification: The generated virtual-link was tested by “ping”.**

Conclusion

- **A method for adding new virtual-nodes/link types by evolving VNodes is proposed.**
  - The second stage, i.e., operational stage, of this evolution was explained.
- **This method**
  - enables abstract and simple specifications of slices by developers.
  - enables authentication and authorization of plug-ins.
  - remains modular plug-in architecture used in the experimental stage.
- **This extended VNode architecture supports**
  - a combination of programmable data-plane and control-plane components.
  - a combination of a decentralized and centralized control.
  - new functions (i.e., new virtual-node/link types) created by combinations of software and hardware.