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.
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.

Original VNode  \[\xrightarrow{\text{Experimental stage}}\]  Evolving VNode  \[\xrightarrow{\text{Operational stage}}\]  Evolved VNode
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.
A developer specifies a new type of virtual-node/link with parameters that specify plug-in parameters.

A virtual link specification (RSpec):

```xml
<linkSliver type="link" ...>
  <vports>
    <vport name="vport1">
      <params>
        <param key="ControlPort" value="CPI11-addr"/>
        <param key="DataPort" value="DPI11-port"/>
      </params>
    </vport>
    <vport name="vport2">
      <params>
        <param key="ControlPort" value="CPI12-addr"/>
        <param key="DataPort" value="DPI12-port"/>
      </params>
    </vport>
  </vports>
  <params>
    <param key="PlugInName" value="virtual-link-1"/>
    <param key="Command-reserveLinkSliver1" value="ls_setup_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>
```

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).

<table>
<thead>
<tr>
<th>Keys</th>
<th>Control plug-in</th>
<th>Data plug-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-type-1</td>
<td>Control plug-in identifier, Authorization information, Commands</td>
<td>Data plug-in identifier, Authentication information</td>
</tr>
<tr>
<td>VNode0</td>
<td>CPIIn0-addr</td>
<td>DPIIn0-port</td>
</tr>
<tr>
<td>VNode1</td>
<td>CPIIn1-addr</td>
<td>DPIIn1-port</td>
</tr>
<tr>
<td>VNode2</td>
<td>CPIIn2-addr</td>
<td>DPIIn2-port</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>link-type-1</th>
<th>Control plug-in identifier, Authorization information, Commands</th>
<th>Data plug-in identifier, Authentication information</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNode0</td>
<td>CPIIl0-addr</td>
<td>DPIIl0-port</td>
</tr>
<tr>
<td>VNode1</td>
<td>CPIIl1-addr</td>
<td>DPIIl1-port</td>
</tr>
<tr>
<td>VNode2</td>
<td>CPIIl2-addr</td>
<td>DPIIl2-port</td>
</tr>
</tbody>
</table>

► 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):

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

Used for accessing the mapping table
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.

- VNodes can process the specifications w/o modification.
### Evaluation of Prototype

**Comparison of virtual-link specifications**

<table>
<thead>
<tr>
<th>Virtual-link specification</th>
<th>Definition length (lines)</th>
<th>Implementation-dependent parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original (before preprocessing)</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Translated (after preprocessing)</td>
<td>14</td>
<td>4</td>
</tr>
</tbody>
</table>

**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.