Control as LCD for future networking

Artur Hecker and Zoran Despotovic

European Research Center, Munich
Huawei Technologies
Programmable networks: change in paradigm

• Legacy: design a network to provide a specific (set of) service(s)
  • Protocols/ exchanges (management, control, data) + stacks/ logic
  • Deployment topology, configuration to bind the pieces
  • Operational traffic steering/traffic distribution

• Programmable networks:
  • Infrastructure with basic capabilities and open interfaces
  • Services: several logics programmed on the latter

• Change: design the network now, program the service later

• Software brittleness (*): cyclomatic complexity

(*) Steve Bellovin
Example: OF SDN

• **Chicken-Egg Problem** in SDN
  • Current SDN *promises* a “software-defined networking”, yet it actually *requires* an existing, well-configured and well-working TCP/IP network
    • Note that this is independent of in-band / out-of-band discussions
  • A pre-set, fixed CP in SDN cannot suit all use cases that SDN promises
    • Non-functional requirements: QoS, scalability, reliability, resilience

• **Self-inflicted errors** in SDN
  • Insufficient protection: the programming model is comparable to DOS
    • You can write a control app to disconnect the controller from switches
    • Hard to protect against this w/o limiting programmability
General Problem Statement

• Context
  • Many components (HW/SW; remote/local; short-/long-lived)
  • Need to be able to bind them to working services operationally

• What is the common minimal requirement on all those components?
  • How to make them programmable?
    • Without making the components too complex
    • Without having to manually deploy things
  • How to make such programmability simple / usable?
    • What do you need to know to start? Does trial-and-error model work?
Our Proposal: Unified control

• Resource-to-resource protocol suite, **dedicated to establishing and maintaining control**
  • Akin to BGP establishing and maintaining IP routing
  • ... But without presuming a specific usage

• Two dimensions of unification
  • Horizontal: span different types of components
  • Vertical: span both executing and executed modules
Our model

New function: Provide a stable view on a distributed resource pool

Resource control agent, running the resource-to-resource protocol
Two faces of the RCA: the RCA acts both as an interface to the local resource that it controls, and as a building block of the control plane spanning all resources within the control domain (unified control peer).

A. Hecker, Z. Despotovic
Control as LCD for programmable networks
Phases

• All resource elements (RE) have an RCA
• Phases, repeated (on event / periodically)
  • All REs bootstrap (find all visible friends)
    • Friends: RCAs from the same control domain
  • All RCAs choose from the friend list some neighbors
  • RCAs run routing over neighbors only
  • The controller capacity and placement is decided autonomously
    • E.g. the topologically most important RE with compute capacity becomes Controller
  • Using distributed storage, RCAs eventually discover a new abstract function “controller”
Conclusions

• We propose a new resource to resource protocol suite

• Capable of producing self-* control planes
  • Need to produce a resilient common functionality to be able to control the resources and the modules
    • Network OS “Kernel”

• Could be a possible extension to ANIMA
  • Extend to other resource types and modules
  • Extend from control channel to control plane
  • Fundamental: infrastructure control through the controlled infrastructure
    • Conflict modeling
# Appendix: Cmp. Unified Control to ANIMA

<table>
<thead>
<tr>
<th>Criterion</th>
<th>ANIMA</th>
<th>Unified Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero preconfiguration ready</td>
<td>Yes (for networking resources)</td>
<td>Yes (for resources)</td>
</tr>
<tr>
<td>Discovery</td>
<td>“All nodes”</td>
<td>Only friends</td>
</tr>
<tr>
<td>Autonomic Control Plane</td>
<td>Yes, interconnect nodes</td>
<td>Yes, establishes control</td>
</tr>
<tr>
<td>Routing</td>
<td>On all nodes</td>
<td>On neighbors only</td>
</tr>
<tr>
<td>Compute Nodes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Overlay structure</td>
<td>As it emerges</td>
<td>Use neighbor selection criteria</td>
</tr>
<tr>
<td>Distributed storage</td>
<td>No?</td>
<td>Yes</td>
</tr>
<tr>
<td>Secure bootstrap</td>
<td>Yes</td>
<td>Not considered so far</td>
</tr>
<tr>
<td>Support for topology dynamics</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Religion/ paradigm</td>
<td>Autonomic networking</td>
<td>Controlled networking</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only autonomic in its own</td>
</tr>
<tr>
<td></td>
<td></td>
<td>implementation</td>
</tr>
</tbody>
</table>
Appendix: what’s wrong with orchestration?

• Orchestration is a management function 😊
  • Requires signaling channels and control
  • Is too far away
  • Cannot efficiently react to faults, local events, etc

State of the art: