Serval: An End-Host Stack for Service-Centric Networking

Erik Nordström
David Shue, Prem Gopalan, Rob Kiefer,
Mat Arye, Steven Ko, Jen Rexford, Mike Freedman

Princeton University
The Internet of the 1970s

Killer Apps: telnet, ftp

Network designed for accessing hosts
The Internet of the 2000s

Users agnostic of actual service location and host
What does Service Access Involve?

1. Locate a nearby service datacenter
   – Map *service* name to location

2. Connect to service
   – Establish data *flow* to instance
   – Load balance between pool of replicas

3. Maintain connectivity to service
   – Migrate between interfaces and networks
Today’s (Overloaded) Abstractions

• Service is IP + port
  – Exposes location
  – Specifies app. protocol
  – One service per IP

• Flow is “five tuple”
  – Binds flow to interface and location
  – Cannot migrate between interfaces or networks
Service Access Today

Enterprise Network

Transit Provider

Cellular Provider

Datacenter

Datacenter

4G
Finding a Service Location

- DNS binds service to location at client (early binding)
  - Caching and ignoring TTL exacerbates the problem
  - Slow failover when instance or load balancer fail
Connecting to Service

- Datacenter LB maps single IP to multiple servers
  - Must do this for every packet on path -> fate sharing
  - Increases complexity and cost
Maintaining Connectivity to Service

- Migrate VMs to balance load in the cloud
  - Requires flat addressing or tunneling within datacenter
Maintaining Connectivity to Service

- Flows break when switching networks or interfaces
Contributions

• Naming abstractions
  – Services, flows
  – Clean role separation in the network stack

• Software architecture for services (Serval)
  – Service-level control/data plane split
  – Service-level events
Naming Abstractions
Today’s (Overloaded) Abstractions

TCP/IP
- connect (IP + port)
- demux (IP + port)
- forward (IP)

Application
Transport
Network
Serval Abstractions

• **Serval cleans the slate**
  – (But not completely)

• **Network layer unmodified!**

• **Service Access Layer (SAL)**
  – Connects to services
  – Maintains connectivity

- Serval
- Application
- Transport
- Service Access
- Network
- forward (IP)
Serval Abstractions

• **Service = ServiceID**
  - *Group* of processes with identical functionality

• **Flow = FlowID**
  - Invariant demux key
  - Host-local, ephemeral

• **Location = IP address**
  - Location, interface
  - Can change dynamically
A Clean Role Separation in the Stack

- What you access (**serviceID**), over which flows (**flowIDs**), and at which service instance (**IP address**)

<table>
<thead>
<tr>
<th>Layer</th>
<th>TCP/IP</th>
<th>Serval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td><strong>connect (IP + port)</strong></td>
<td><strong>connect (serviceID)</strong></td>
</tr>
<tr>
<td>Transport</td>
<td><strong>demux (IP + port)</strong></td>
<td></td>
</tr>
<tr>
<td>Service Access</td>
<td></td>
<td><strong>demux (serviceID, flowID)</strong></td>
</tr>
<tr>
<td>Network</td>
<td><strong>forward (IP)</strong></td>
<td><strong>forward (IP)</strong></td>
</tr>
</tbody>
</table>
Service Names (ServiceIDs)

- **ServiceIDs allocated in blocks**
  - Prefix ensures global uniqueness
  - Prefix-based aggregation and LPM

- **A ServiceID late binds to service instance**
  - ServiceID in *first* packet of connection
  - Service-level routing and forwarding

| Provider prefix | Provider-specific | Self-certifying |
A Service-Aware Network Stack

connect(sock, serviceID)

bind(sock, serviceID)
listen(sock)

Network stack must resolve service to instance for client

Network stack must advertise service for server
Software Architecture
Serval End-host Architecture

Application

Service Controller

Service Control API

Flow Table

FlowID  Socket

Service Table

ServiceID  Action  Sock/Addr

IP Forwarding Table

Dest Address  Next Hop
## Data Plane: The Service Table

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Rule State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix A</td>
<td>FORWARD</td>
<td>Send to addr A1</td>
</tr>
<tr>
<td>Prefix B</td>
<td>FORWARD</td>
<td>Send to [A2, A3, A4]</td>
</tr>
<tr>
<td>Prefix C</td>
<td>DEMUX</td>
<td>Send to listening sock s</td>
</tr>
<tr>
<td>Prefix D</td>
<td>DELAY</td>
<td>Queue and notify service controller</td>
</tr>
<tr>
<td>Prefix E</td>
<td>DROP</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>FORWARD</td>
<td>Send to A5</td>
</tr>
</tbody>
</table>
### Data Plane: The Service Table

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Rule State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix A</td>
<td>FORWARD</td>
<td>Send to addr A1</td>
</tr>
<tr>
<td>Prefix B</td>
<td>FORWARD</td>
<td>Send to [A2, A3, A4]</td>
</tr>
<tr>
<td>Prefix C</td>
<td>DEMUX</td>
<td>Send to listening sock s</td>
</tr>
<tr>
<td>Prefix D</td>
<td>DELAY</td>
<td>Queue and notify service controller</td>
</tr>
<tr>
<td>Prefix E</td>
<td>DROP</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>FORWARD</td>
<td>Send to A5</td>
</tr>
<tr>
<td>ServiceID</td>
<td>Action</td>
<td>Rule State</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>Prefix A</td>
<td>FORWARD</td>
<td>Send to addr A1</td>
</tr>
<tr>
<td>Prefix B</td>
<td>FORWARD</td>
<td>Send to [A2, A3, A4]</td>
</tr>
<tr>
<td>Prefix C</td>
<td>DEMUX</td>
<td>Send to listening sock s</td>
</tr>
<tr>
<td>Prefix D</td>
<td>DELAY</td>
<td>Queue and notify service controller</td>
</tr>
<tr>
<td>Prefix E</td>
<td>DROP</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>FORWARD</td>
<td>Send to A5</td>
</tr>
</tbody>
</table>
## Data Plane: The Service Table

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Rule State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix A</td>
<td>FORWARD</td>
<td>Send to addr A1</td>
</tr>
<tr>
<td>Prefix B</td>
<td>FORWARD</td>
<td>Send to [A2, A3, A4]</td>
</tr>
<tr>
<td>Prefix C</td>
<td>DEMUX</td>
<td>Send to listening sock s</td>
</tr>
<tr>
<td>Prefix D</td>
<td>DELAY</td>
<td>Queue and notify service controller</td>
</tr>
<tr>
<td>Prefix E</td>
<td>DROP</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>FORWARD</td>
<td>Send to A5</td>
</tr>
</tbody>
</table>
# Data Plane: The Service Table

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Rule State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix A</td>
<td>FORWARD</td>
<td>Send to addr A1</td>
</tr>
<tr>
<td>Prefix B</td>
<td>FORWARD</td>
<td>Send to [A2, A3, A4]</td>
</tr>
<tr>
<td>Prefix C</td>
<td>DEMUX</td>
<td>Send to listening sock s</td>
</tr>
<tr>
<td>Prefix D</td>
<td>DELAY</td>
<td>Queue and notify service controller</td>
</tr>
<tr>
<td>Prefix E</td>
<td>DROP</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>FORWARD</td>
<td>Send to A5</td>
</tr>
</tbody>
</table>
Service Access with Serval
Adding a Service Instance

Application

Service Controller

Register Service X

bind(X)
listen()

<table>
<thead>
<tr>
<th>FlowID</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Sock/Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>DMX</td>
<td>S</td>
</tr>
</tbody>
</table>

Add DEMUX rule
Removing a Service Instance

- **Application**
- **Service Controller**
- Unregister Service X

**close()**

<table>
<thead>
<tr>
<th>FlowID</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Sock/Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>DMX</td>
<td>s</td>
</tr>
</tbody>
</table>

Remove DEMUX rule
Control Plane: The Service Controller

Service X @ address a
Control Plane: The Service Controller

<table>
<thead>
<tr>
<th>FlowID</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Sock/Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>FWD</td>
<td>d</td>
</tr>
</tbody>
</table>

Add FORWARD rule

Service Controller

Service X @ address d
Service Access with Serval
Connecting to Service X

Application

socket()

Service Controller

<table>
<thead>
<tr>
<th>FlowID</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Sock/Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>FWD</td>
<td>c</td>
</tr>
</tbody>
</table>

Allocates local flowID
Connecting to Service X

Application

connect(X)

<table>
<thead>
<tr>
<th>FlowID</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Sock/Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>FWD</td>
<td>c</td>
</tr>
</tbody>
</table>

To c
Load Balancing in Service Router

<table>
<thead>
<tr>
<th>From a</th>
<th>c</th>
<th>2</th>
<th>X</th>
<th>SYN</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td>2</td>
<td>X</td>
<td>SYN</td>
</tr>
<tr>
<td>To e</td>
<td>f</td>
<td>2</td>
<td>X</td>
<td>SYN</td>
</tr>
</tbody>
</table>

Table:

<table>
<thead>
<tr>
<th>ServiceID</th>
<th>Action</th>
<th>Sock/Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>FWD</td>
<td>d,e</td>
</tr>
</tbody>
</table>
Service Instance Providing Service X

Application

accept()

FlowID | Socket
---|---
3 | \(s_c\)

ServiceID | Action | Sock/Addr
---|---|---
X | DMX | \(s\)

To a

\(e\)

e a 3 2 SYN-ACK
Service Access with Serval

Internet Access with Serval

datacenter SYN

Service Router

d,e

data

Router

d,e

data
Ad hoc Service Discovery

Accessing service X

connect(X)

ServiceID | Action    | Rule State
----------|-----------|-------------
default   | FORWARD   | “broadcast”
What does Service Access Involve?

1. Locating a nearby service datacenter
   - Map service name to location

2. Connecting to service
   - Establish data flow to instance
   - Load balance between pool of replicas

3. Maintaining connectivity to service
   - Migrate between interfaces and networks
Migration of Flows

Migrate flow $a_1 \rightarrow a_2$
Add flow \( a2 \leftrightarrow a4 \)
Prototype

• End-host network stack (28,000 LOC)
  – Linux kernel module
  – BSD sockets with AF_SERVAL protocol family
  – AF_INET sockets can be accessed simultaneously

• Legacy middleboxes / NATs handled via encap.

• Translator for incremental deployment
  – Unmodified apps and end-hosts
  – Serval apps with unmodified services
Incremental Deployment

App

Translator

TCP/IP

Serval
Incremental Deployment

Translator

App

TCP/IP

Serval

TCP/IP

Translator
Use of Migration on Clients

WiFi

Cellular

Saves > 900 MB cellular data per month

Single Serval TCP connection that never breaks
Uses of Migration on Servers

Load balancing across NICs

Throughput (Mbps)

Time (s)

Flow 1 moved to eth1

Both flows use eth0

Flow 1

Flow 2

Flow 1

Flow 2
Uses of Migration on Servers

Migrating VMs across subnets

Throughput (Mbps) vs. Time (s)

Flow 1

VM changes subnet, acquiring new address

VM migrates flow to new address
## Competitive Performance

### TCP Throughput

<table>
<thead>
<tr>
<th></th>
<th>Mean (Mbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP</td>
<td>934.5</td>
</tr>
<tr>
<td>Serval</td>
<td>933.8</td>
</tr>
<tr>
<td>Translator</td>
<td>932.1</td>
</tr>
</tbody>
</table>

### Service Table Throughput

<table>
<thead>
<tr>
<th></th>
<th>Mbit/s</th>
<th>Kpkt/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP forwarding</td>
<td>987</td>
<td>388.4</td>
</tr>
<tr>
<td>Serval</td>
<td>872</td>
<td>142.8</td>
</tr>
</tbody>
</table>
### Applications are Easy to Port

<table>
<thead>
<tr>
<th>Application</th>
<th>Codebase</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iperf</td>
<td>5,934</td>
<td>240</td>
</tr>
<tr>
<td>TFTP</td>
<td>3,452</td>
<td>90</td>
</tr>
<tr>
<td>wget</td>
<td>87,164</td>
<td>207</td>
</tr>
<tr>
<td>Elinks browser</td>
<td>115,224</td>
<td>234</td>
</tr>
<tr>
<td><strong>Firefox browser</strong></td>
<td><strong>4,615,324</strong></td>
<td><strong>70</strong></td>
</tr>
<tr>
<td>Mongoose webserver</td>
<td>8,831</td>
<td>425</td>
</tr>
<tr>
<td>Memcached server</td>
<td>8,329</td>
<td>159</td>
</tr>
<tr>
<td>Memcached client</td>
<td>12,503</td>
<td>184</td>
</tr>
</tbody>
</table>
SDN to the Edges!

• SDN about network-wide visibility and control
  – Today’s “SDN” (OpenFlow) primarily focuses on layer-2 / layer-3 abstractions

• Serval extends SDN model to the network edge
  – New programming abstractions for services, flows, hosts, and interfaces
  – Service-level control/data plane split

• Joint service and network control
Summary of Contributions

• New naming abstractions
  – Clean role separation in the stack
  – Makes it easier to build and manage services

• Software architecture for services
  – Flexible service resolution and discovery
  – Maintains robust connectivity
  – Joint service and network management
Papers, demos, source code (GPL) online
Related Work

• Locator/identifier separation
  – HIP, i3, HAIR, DOA, LISP, LNA

• Data-oriented networking
  – DONA, CCNx

• Support for mobility and migration
  – TCP Migrate, Mobile IP, ROAM

• Multipath and multi-homing
  – MPTCP, SCTP, Shim6