Building a service-centric network with SCAFFOLD*

Michael J. Freedman
Princeton University

with Prem Gopalan, Steven Ko, Jen Rexford, and David Shue

* Service-Centric Architecture For Flexible Object Localization and Distribution
From a host-centric architecture

1960s

29 Oct 69 2100
LOADED OP. PROGRAM CSK
FOR BEN BARKER
BBY

22:30
Talked to SER
csl
Host to Host

Leftop inp. program (csl)
Running after sending
a host dead message
to inp.
From a host-centric architecture

1960s

1970s
From a host-centric architecture

1960s
1970s
1990s

NOW
To a service-centric architecture

1960s
1970s
1990s
2000s
To a service-centric architecture

• Users want services, agnostic of actual host

• Service operators need to support replica selection, failover, migration, ...

• Service-centric anycast as first-class primitive
Challenges

• Handling replicated services
  – Control over replica selection among groups
  – Control of network resources for shared between groups
  – Handling dynamics among group membership and deployments

• Handling churn
  – Flexibility: From sessions, to hosts, to datacenters
  – Robustness: Largely hide from applications
  – Scalability: Local changes shouldn’t need to update global info
  – Scalability: Churn shouldn’t require per-client state in network
  – Efficiency: Wide-area migration shouldn’t require tunneling
SCAFFOLD as ...

- Clean slate design
- Multi-datacenter architecture for single administrative domain
Target: Single administrative domain

- DC network management more unified, simple, centralized
- End-host OS net-imaged and can be fork-lift upgraded
- Already struggling to provide scalability and service-centrism
- Cloud computing trends lesson importance of fixed, physical hosts
Appearance of service-centrism today

**Layer 4/7:** DNS with small TTLs
- HTTP redirects
- Layer-7 switching

**Layer 3:** IP addresses and IP anycast
- Inter/intra routing updates

**Layer 2:** VIP/DIP load balancers
- VRRP, ARP spoofing
Outline of talk

• Principles for service-centric design

• Architecture and design of SCAFFOLD
  – Network support
    • New forwarding model
    • Support for migration and failover
    • Network and service management
  – End-host support: socket interface and network stack

• Implementation, especially OpenFlow/NOX details (and desiderata)
Principles of SCAFFOLD

• Service-centric naming
  – Service/object id’s as flexible naming, not hosts
    • Webservers providing front-tier web (calendar.google.com)
    • A particular region in a (distributed) Virtual World service
    • A particular file in a CDN
Principles of SCAFFOLD

• Service-centric naming
  – Service/object id’s as flexible naming, not hosts
  – Network-level addresses hidden from application

• Flows and anycast as basic network primitives
  – Names correspond to anycast groups, unicast as special case
  – Connection affinity for flows within anycasted endpoints

• Migration and failover through address remapping
  – Flows identified by each endpoint, not pairwise
  – Control through in-band signalling, stateless forwarders

• Minimize visibility of churn for scalability
  – Different addressing for different scopes (successive refinement)
  – Unity of functionality and management
Extent of changes

- Change socket layer + stack
- Change the packet format
- Change in-network support
Application’s network API

**Today (IP / BSD sockets)**

```c
fd = open();

Datagram:
sendto (IP:port, data)

Stream:
connect (fd, IP:port)
send (fd, data);
```

**SCAFFOLD**

```c
fd = open();

Unbound datagram:
sendto (objectId, data)

Bound datagram:
connect (fd, objectId)
send (fd, data);
```

**IP:** Application sees network, network doesn’t see app

**SCAFFOLD:** Network sees app, app doesn’t see network
SCAFFOLD in the network:

Unbound Datagrams and Network Support
Service-level naming and forwarding

Services should control instance selection
Successive refinement of datagrams

Data Center 1

Host ID: A

C: A
X: 2

Data Center 2

Host ID: B

C: 1
X: B
X: D
X: E

Hosts:

HID: D
HID: E
Successive refinement of datagrams

- Forwarding refers to successively-refined destinations
- Churn hidden from wider-area as much as possible
SCAFFOLD in the network:

Bound Flows
and Network Devices
Forwarding bound flows

Object Switch

Host ID: A

Host ID: B

SRC C
DST X

SRC X
DST B

SRC X
DST C

SRC C
DST X
Forwarding bound flows

Object Switch

Host ID: A

Host ID: B

Switch

C: A
X: B
X: D
X: E

SRC
C
A
p

DST
X
B

SRC
X
B
q

DST
C
A
p

SRC
X

DST

X

X

X

X

X

X

X

X

X
Forwarding bound flows

Host ID: A

Object Switch

Host ID: B

Flow ID

Header

Obj ID

Host ID

Sock ID

C

A

X

B

D

E

X

C

SRC

DST

C

A

p

X

DST

B

q

fd = 5 oid = C,X

fd = 9 oid = X,C

Host ID: B

SRC

DST

X

B

q

C

A

p

 SRC

DST

X

B

q

C

A

p
Services should control instance selection

Flow affinity, yet no per-flow network state

Flows identified by each endpoint
Forwarding bound flows

Data Center 1

Host ID: A

Data Center 2

Host ID: B
Forwarding bound flows

Data Center 1

Host ID: A

Data Center 2

Host ID: B

SRC

DST

C: A

X: 2

C 1 A p

X 2

C 1 A p

X 2 B

SRC

DST

1A : a_A

2 : a_2

2B : a_B

2D : a_D

2E : a_E

1 : a_1

SRC

DST

C 1 A p

X 2 B

2 : a_2

X

1

a

OS1

1A

2B

2D

2E

1 : a_1

OS2

2B

2D

2E

1 : a_1
Forwarding bound flows

Data Center 1

Data Center 2

Host ID: B

Applications name logical flows, not physical locations
Label management by end-host

User-Space Application

<table>
<thead>
<tr>
<th>Socket State</th>
<th>Local Object ID</th>
<th>Local Flow ID</th>
<th>Remote Object ID</th>
<th>Remote Flow ID</th>
<th>Accepted Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>C</td>
<td>1:A:p</td>
<td>X</td>
<td>2:B:q</td>
<td>No</td>
</tr>
<tr>
<td>bound</td>
<td>C</td>
<td>1:A:r</td>
<td>Y</td>
<td>--</td>
<td>No</td>
</tr>
<tr>
<td>unbound</td>
<td>E</td>
<td>--</td>
<td>Z</td>
<td>--</td>
<td>No</td>
</tr>
</tbody>
</table>

File Descriptor | Object IDs
--- | ---
5 | C → X
9 | C → X
47 | C → Y

IP: Application sees network, network doesn’t see app

SCAFFOLD: Network sees app, app doesn’t see network
### Label management by end-host

<table>
<thead>
<tr>
<th>SCAFFOLD</th>
<th>Socket State</th>
<th>Local Object ID</th>
<th>Local Flow ID</th>
<th>Remote Object ID</th>
<th>Remote Flow ID</th>
<th>Accepted Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>open</td>
<td>C</td>
<td>4:A:p</td>
<td>X</td>
<td>3:D:q</td>
<td>No</td>
</tr>
<tr>
<td>bound</td>
<td>bound</td>
<td>C</td>
<td>4:A:r</td>
<td>Y</td>
<td>--</td>
<td>No</td>
</tr>
<tr>
<td>unbound</td>
<td>unbound</td>
<td>E</td>
<td>--</td>
<td>Z</td>
<td>--</td>
<td>No</td>
</tr>
</tbody>
</table>

### User-Space Application

<table>
<thead>
<tr>
<th>File Descriptor</th>
<th>Object IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>C → X</td>
</tr>
<tr>
<td>9</td>
<td>C → X</td>
</tr>
<tr>
<td>47</td>
<td>C → Y</td>
</tr>
</tbody>
</table>

**IP:** Application sees network, network doesn’t see app

**SCAFFOLD:** Network sees app, app doesn’t see network
Migration and Failover

• Planned migration or physical mobility
  – In-band signaling: Destination replaces flowid_{old} with flowid_{new}

• Unplanned failover
  – Failure of destination causes removal from flow switch
  – Flow switch lookup fails, flow re-resolved at object switch
  – Sender again learns new location (flowid) via in-band signaling

• May require new 3-way handshake for renegotiation
Network Management APIs

- **Flow switch**
  - **Flow Table**: Map FlowID to network addr or out port

- **Object switch**
  - **Object Table**: Map ObjID to FlowID label
  - Typically collocates flow table

- **End-host**
  - Join/leave network
  - Register/unregister object IDs
  - Migrate/redirect flowids

- **Network Controller**
Network Management APIs

- **Flow switch**
  - Flow Table: Map FlowID to network addr or out port

- **Object switch**
  - Object Table: Map ObjID to FlowID label
  - Typically collocates flow table

- **End-host**
  - Join/leave network
  - Register/unregister object IDs
  - Migrate/redirect flowids

- **Network Controller**
Network Management APIs

- **Flow switch**
  - **Flow Table**: Map FlowID to network addr or out port

- **Object switch**
  - **Object Table**: Map ObjID to FlowID label
  - Typically collocates flow table

- **End-host**
  - Join/leave network
  - Register/unregister object IDs
  - Migrate/redirect flowids

- **Network Controller**
Incremental Deployment Model

DC 1

Backbone

DC 2

Internet

Legacy Clients
Incremental Deployment Model

Legacy Clients

Anycasted IP Prefix (BGP)

DC 1

DC 2

IP forwarding

MAC forwarding

Anycast Subprefix 1

Anycast Subprefix 2
Current implementation

- Backwards compatible with legacy IPv4 networks
  - SCAFFOLD packet format:
    - ObjectID in UDP port
    - Flowid in IPv4 addr

- Flow switch
  - OpenFlow software switch
  - Hit: LPM on flowID
  - Miss: EGRE tun to obj switch

- Object switch
  - OpenFlow software switch
  - Hit: Exact match on objID
  - Miss: Send packet to Controller
Current implementation

- **End-host**
  - New SCAFFOLD socket library
  - User-level Click process
    - Network and transport
    - Comm. with Controller
    - TUN/TAP driver and in-kernel Click for packet interception

- **Network Controller**
  - NOX
  - New host API (via packet_in)
  - Manage flow/object switches
Openflow Desiderata

• Match on one-of-N entries (e.g., hashing)
• Multicast (e.g., for planned redirect)
• Packet encapsulation (for forwarding to obj switch)
  – Unnecessary if SCAFFOLD-only network
• More flexibility/space for header encoding/rewriting
  – Currently $2^{16}$ objects and fixed $2^8$ DCs, $2^8$ hosts, $2^{16}$ sockets
  – IPv6 support would provide much greatly scalability
  – Ultimately prefer to define own header format
SCAFFOLD on end-hosts:

New socket API and network stack
Socket Architecture

```
typedef struct {
    uint16_t v;
} sf_obj_t;

struct sockaddr_sf {
    uint16_t family;
    sf_obj_t local_obj_id;
    sf_obj_t remote_obj_id;
};
```

```c
int socket_sf (int domain, int type, int protocol)

int bind_sf (int s, const sockaddr *, socklen_t)
    – Blocking call, returns after register call b/w scafd and Controller

int connect_sf (int s, const sockaddr *, socklen_t, sf_err_t &)
    – Both blocking and non-blocking versions (works with select)
    – Returns success after 3-way handshake with remote sockaddr
```
Socket Architecture

typedef struct {
    uint16_t v;
} sf_obj_t;

struct sockaddr_sf {
    uint16_t family;
    sf_obj_t local_obj_id;
    sf_obj_t remote_obj_id;
};

int listen_sf (int s, int backlog, sf_err_t &)
int listen_sf (int s, const sockaddr *, socklen_t, int backlog, sf_err_t &)

- Latter version allows single socket to listen on multiple objects
- Results in a register call b/w scafd and Controller

int accept_sf (int s, sockaddr *, socklen_t, sf_err_t &)

- Returns bound socket (sender/receiver flowids established)
Socket Architecture

```
typedef struct {
    uint16_t v;
} sf_obj_t;

struct sockaddr_sf {
    uint16_t family;
    sf_obj_t local_obj_id;
    sf_obj_t remote_obj_id;
};
```

ssize_t send_sf (int s, const void *, size_t, int flags, &)
ssize_t sendto_sf (int s, const void *, size_t, int flags,
                  const sockaddr *, socklen_t, sf_err_t &)
ssize_t recv_sf (int s, void *, size_t, int flags, &)
ssize_t recvfrom_sf (int s, void *, size_t, int flags,
                     struct sockaddr *, socklen_t *, sf_err_t &)

int close_sf (int s, sf_err_t &);

- Connected sockets execute 3-way handshake
- Bound/listening sockets unregister all objIDs with Controller
Applications

• Replicated web services
  – Fault-tolerant failover for unmodified services
• Key-value store w/o layer-7 switch (memcached, CRAQ)
• Layer-3 VM migration
• Wide-area content distribution network
• Substrate for Virtual Worlds (Meru)

• Current ports
  – Iperf
  – TFTP (FTP over UDP)
  – NFSv3 (in progress)
Unresolved for clean-slate design

• Discovery and ecosystem of authoritative object switches

• Security
  – Wide-area routing announcements
  – In-band signaling of flowid updates

• Flexibility and extensibility
  – Use for fine-grain, ephemeral obj ID’s (CCN)
  – Revisit stream-oriented apps as self-descriptive datagrams
  – Supplant all IP and host-to-host communication?
    “Host” as service ID with single location?
Related Work

- **Addressing**: Separating location from identity
  - SFR, LNA, DOA, LISP; ROFL, SEATTLE
  - Triad, DONA, CCN
  - Portland, VL2, SPAIN

- **Migration and Mobility**
  - Mobile IP, i3, LISP, TCP Migrate, SCTP; RTP, Trickles

- **Replication and IP anycast**
  - SFR, DOA; 4D-like control; PIAS, GIA

- **Routing on coarse grain identifiers**
  - AIP, NIRA
Building a service-centric network with SCAFFOLD*

Michael J. Freedman
Princeton University

with Prem Gopalan, Steven Ko, Jen Rexford, and David Shue

* Service-Centric Architecture For Flexible Object Localization and Distribution