ECCP
A Formally-Verified Migration Protocol
For Mobile, Multi-Homed Hosts

Matvey Arye

Joint work with:
Erik Nordström, Robert Kiefer
Jennifer Rexford, Michael J. Freedman
Original Internet Architecture

Hosts did not move and had a single connection to the Internet
Fast Forward

• Mobile devices have new capabilities
  – Devices move
  – Multiple points-of-attachment

• Servers have changed
  – VM migration
  – Multiple network attachments (NICs)
  – Data-center multihoming
Extending Network Capabilities

• Host mobility, VM Migration
  – Connection shouldn’t break when hosts move

• Switching seamlessly between WiFi and 4G
  – Ability to switch between network interfaces

• Load balancing between network paths across interfaces
  – Ability to move individual flows between interfaces

• Having backup routes on alternative interfaces
  – Maintaining a list of alt. interfaces for connections
Problems Arise From Current Abstractions

- No network changes
- Independent of data delivery semantics
The Flow Abstraction

Application

Data Delivery

Connection Control

Network

PID Connection PID

FlowID1 Flow1 FlowID1

FlowID2 Flow2 FlowID2

IP1 Flow1 IP2

IP3 Flow2 IP4

Application

Data Delivery

Connection Control

Network
Our Approach For Handling Device Mobility

Connection Control

Network

FlowID1  Flow1  FlowID1

IP1  Flow1  IP5

Connection Control

Network

My Address has changed

Alice

Bob: IP2

Bob: IP5
Contribution 1: ECCP

End-to-End Connection Control Protocol

- Host mobility through end-to-end signaling
- Transport-layer independence
- Multipath through new flow abstraction
Contribution 2: Formal Verification

• Connection control protocols hard to get right
  – We show that TCP-Migrate and HIP are incorrect

• Non-Determinism makes it hard to verify
  – Unreliable network, changing network identifiers
  – Non-determinism leads to state-space explosion

• We show new techniques to enable verification
  – Verified ECCP in SPIN
## Other End-to-End Protocols

<table>
<thead>
<tr>
<th></th>
<th>ECCP</th>
<th>TCP-Migrate</th>
<th>MPTCP</th>
<th>HIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formally Verified</td>
<td>✓</td>
<td></td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Transport Independent</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Rapid Migration</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Multipath Capable</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Per-Flow Migration</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Protocols

• Establishing connections

• Moving flows to new addresses

• Adding flows to connection

• Handling NATs

• Simultaneous migrations
Connection Establishment

• Three-way handshakes to establish states

• Each peer communicates flowID to other peer
  – Unlike IP addr., doesn’t change during migration
  – Packets demultiplexed on local flowID

• Optionally sends alternative addresses to peer for fail-over and additional flows
The Protocol – Initial Flow

Client

- SYN (Service S)
- Flow ID-C
- Demux on: Flowd ID-C

Server

- SYN-ACK
- Flow ID-C
- Flow ID-S
- Demux on: S
- Demux on: Flowd ID-S

ACK
- Flow ID-C
- Flow ID-S
The Protocol – Changing Addresses

Mobile
New Address IP5
- RSYN
  Version #-M
  Flow ID-M
  Flow ID-S
  SRC=IP5

Demux on: Flow ID-M
- ACK
  Version #-M
  Flow ID-M
  Flow ID-S

Stationary
- Demux on: Flow ID-S
  Record addresses IP5
- RSYN-ACK
  Version #-M
  Flow ID-M
  Flow ID-S

- Demux on: Flow ID-S
  Change address to IP5
Version #s

• Need to use versioning on migration messages
• HIP, TCP-Migrate use TCP-like sequence #s
  – Ties connection control to data delivery
  – Creates problems -- need different semantics

Semantics when getting packet N:

“Sequence”
Received 0 to N-1
Cannot skip ahead

“Version”
All previous #s < N
Can skip ahead
Sequence # Semantics are Dangerous

**Mobile**

New Address IP5

RSYN
Sequence #n

**Stationary**

RSYN-ACK
Sequence #n

Can’t process sequence #n+1 because didn’t finish #n

New Address IP6

RSYN
Sequence #n+1
Formal Verification
Formal Verification - Overview

• Modeled in SPIN

• Checks for deadlocks
  – Neither party can send or receive messages

• Checks for livelocks
  – Neither party can do anything useful
  – Each host can ping the other host
Goals of Connection Control

• Robust connectivity across mobility events
  – Maintain up-to-date mapping between flows & IPs
  – Correct if each host can ping its peer

• What connection control is NOT
  – Reliable delivery
  – Bit-correctness of data (i.e. checksums)
  – Ordering of data
Model Checking 101: Explore All Interleavings
Model Checking 101: Explore All Interleavings

Process 1

Process 2
Model Checking 101: Build Global State-Space

State 1

State 2
State 3
State 4
State 5
Verification Challenges

• Most protocols verified in SPIN sit on top of a reliable data-delivery layer
  – But for ECCP, the network is unreliable: loss, duplication, and reordering of packets are possible but can cause state-space explosion

• State-space explosion due to random FlowIDs

• No notion of time in SPIN – timeouts are tricky
  – But are needed to recover from packet loss
Modeling an Unreliable Network

Network simulator can drop, reorder or duplicate packets
Creates Unnecessary States

Process 1  Network Sim  Process 2

1

2
Creates Unnecessary States

Process 1  Network Sim  Process 2

1

2
Creates Unnecessary States

Relative order does not matter For protocol execution
More Efficient Implementation

Network simulator runs as part of the sending process
Formal Verification - Completeness

• Each version # creates new state-space tree

• So, verification does not reach a fixed point
  – But, verifies up to 6 migrations for base protocol
  – 4 migrations for full protocol
Implementation

• ECCP part of larger Serval project
  – Next-generation service-oriented network stack
  – http://www.serval-arch.org/

• Loadable kernel module
  – Runs on Linux, Android,…

• Adapts “ESTABLISHED” state of TCP
Evaluation – Client Interface Changes

Saves > 2GB cellular data per month

One of the authors walks through campus, playing music through Google Play Music. No loss in playback quality.
Conclusion

• New abstractions
  – Decoupling data delivery and connection control
  – Flows as path-dependent parts of connections

• Design of demultiplexing keys is important
  – Independent of network identifiers

• Ordering semantics are tricky to get right

• Formal verification is important and possible
Formal Models

http://www.serval-arch.org/eccp/

Implementation

http://www.serval-arch.org/