Algorithmic Opportunities and Challenges of NFV and SDN
A Guided Tour

Stefan Schmid
Aalborg University, Denmark & TU Berlin, Germany
NFV+SDN: It’s a great time to be a researcher!

Innovation Rhone and Arve Rivers, Switzerland
Credits: George Varghese.
Why Flexibilities? Changing Requirements!

- Microservices deployed using containers introduce rapid changes in traffic workloads.
- Augmented reality requires real-time responsiveness.
- IoT significantly increases the # connected devices.
- Datacenter traffic is growing (but has structure and is sparse):

Jupiter rising @ SIGCOMM 2015

Heatmap of rack-to-rack traffic ProjecToR @ SIGCOMM 2016

Credits: Why (and How) Networks Should Run Themselves. Nick Feamster and Jennifer Rexford
We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network change was [...] executed incorrectly [...] more "stuck" volumes and added more requests to the re-mirroring storm

Service outage was due to a series of internal network events that corrupted router data tables

Experienced a network connectivity issue [...] interrupted the airline's flight departures, airport processing and reservations systems

Credits: Nate Foster
Outage of a data center of a Wall Street investment bank: lost revenue measured in USD $10^6 / \text{min}$!

Quickly, assembled emergency team:

The compute team: quickly came armed with reams of logs, showing how and when the applications failed, and had already written experiments to reproduce and isolate the error, along with candidate prototype programs to workaround the failure.

The storage team: similarly equipped, showing which file system logs were affected, and already progressing with workaround programs.

The networking team: All the networking team had were two tools invented over twenty years ago [*ping* and *traceroute*] to merely test end-to-end connectivity. Neither tool could reveal problems with the switches, the congestion experienced by individual packets, or provide any means to create experiments to identify, quarantine and resolve the problem.

Source: «The world’s fastest and most programmable networks» White Paper Barefoot Networks

- Internet-of-Things, e.g., DDoS Fall 2016
  - “Baby-phone”, hacked cameras, etc.
  - Biggest Internet attack ever: >500 Gbps

- Untrusted hardware
  - Attackers repeatedly compromised routers
  - Compromised routers are traded underground
  - Network vendors left backdoors open
  - National security agencies can bug network equipment (e.g., hardware backdoors, Snowden leaks)

- Hacked wireless/cellular equipment
  - Insecure femto cells
  - Rogue access points

- Internet-of-Things, e.g., DDoS Fall 2016
  - “Baby-phone”, hacked cameras, etc.
  - Biggest Internet attack ever: >500 Gbps

- Untrusted hardware
  - Attackers repeatedly compromised routers
  - Compromised routers are traded underground
  - Network vendors left backdoors open
  - National security agencies can bug network equipment (e.g., hardware backdoors, Snowden leaks)

- Hacked wireless/cellular equipment
  - Insecure femto cells
  - Rogue access points

How to build a secure network over insecure hardware?!
Big Challenge: Efficient Resource Utilization

- Wireless infrastructure not used very efficiently today
- E.g., WiFi: huge demand-supply mismatch (e.g., home networks):

  Millions of access points

  but:

  A device can access only through a very small percentage

Solution: virtualization, multi-tenancy, etc.?
Big Challenge: Efficient Resource Utilization

Wireless infrastructure not used very efficiently today:

Further reading:

**OpenSDWN: Programmatic Control over Home and Enterprise WiFi**
ACM Sigcomm Symposium on SDN Research (SOSR), Santa Clara, California, USA, June 2015.

**SecuSpot: Toward Cloud-Assisted Secure Multi-Tenant WiFi HotSpot Infrastructures**
ACM CoNEXT Workshop on Cloud-Assisted Networking (CAN), Irvine, California, USA, December 2016.

Solution: virtualization, multi-tenancy, etc.?
Big Challenges: Sharing and *Predictable* Performance

Tenant 1

Embedding

Tenant 2

Flexible Resource Sharing and (Performance) Isolation
Big Challenges: Sharing and *Predictable* Performance

Tenant 1

Tenant 2

Flexible Resource Sharing and (Performance) Isolation

Solution: explicit reservations of all resources?
And many more...

- **Slow innovation**: Innovation speed depends on **hardware life-cycles**, impossible to tailor to specific needs.

- **Traffic Engineering (TE)**: efficient use of WAN infrastructure through more **direct and fine-grained control** of traffic (e.g., beyond shortest paths, destination-based routing).

- **Failover**: failover via control plane is slow, especially if control plane is **decentralized** (reconvergence time).

- **Cost**: special purpose hardware expensive.
SDN/NFV Opportunities: Programmability, (logical) centralization and virtualization (multi-tenancy).

Some (often read) claims:

- Simpler
- More flexible
- Automatically verifiable
- And hence more secure?
SDN/NFV Opportunities: Programmability, (logical) centralization and virtualization (multi-tenancy).

Some (often read) claims:

- Simpler
- More flexible
- Automatically verifiable
- And hence more secure?
SDN/NFV Opportunities: Programmability, (logical) centralization and virtualization (multi-tenancy).

Some (often read) claims:

- Simpler
- More flexible
- Automatically verifiable
- And hence more secure?

Really?

 Algorithms? Avoid instabilities!

 Complexity of this?

 New threats?
A Mental Model for This Talk
A Mental Model for This Talk

Possibly virtualized (on commodity hw)…
Let’s talk about opportunities!

Opportunity: centralization!
Example: Adversarial Trajectory Sampling

Trajectory Sampling
- Method to infer packet routes
- Low overhead, direct and passive measurement

**Principle:** Sample subset of packets consistently (e.g., hash over immutable fields)

Packets sampled either at all or no location!
Example: Adversarial Trajectory Sampling

Trajectory Sampling

- Method to infer packet routes
- Low overhead, direct and passive measurement

**Principle:** Sample subset of packets consistently (e.g., hash over immutable fields)

Packets sampled either at all or no location!

But: Fails when switches are malicious! E.g., switch knows which headers are currently not sampled: no risk of detection!
A Malicious Switch Could Do Many Things...

Mirror!

Exfiltration
A Malicious Switch Could Do Many Things…

Exfiltration

Also: drop packets (that are currently not sampled), inject packets, change VLAN tag, …
A Malicious Switch Could Do Many Things…

Mirror!

„Could SDN be used to render trajectory sampling more robust to such behavior?“

Exfiltration

Also: drop packets (that are currently not sampled), inject packets, change VLAN tag, ...
A Malicious Switch Could Do Many Things...

Mirror!

Idea: Introduce risk of detection! Good nodes $G_1$, $G_2$, $G_3$, could help detect if bad node $B$ does not know their sampling range!
Adversarial Trajectory Sampling: A Case of SDN?

Idea: design SDN application that makes sampling **unpredictable**!

Controller distributes hash ranges **redundantly**...

... but securely over (**secure**) communication channels.
Adversarial Trajectory Sampling: A Case of SDN?

Idea: design SDN application that makes sampling unpredictable!

Controller distributes hash ranges redundantly...

... but securely over (secure) communication channels.

How to minimize sampling overhead and maximize detection probability? An algorithmic question.
Adversarial Trajectory Sampling: A Case of SDN?

Idea: design SDN application that makes sampling unpredictable!

Further reading:
Software-Defined Adversarial Trajectory Sampling

... but securely over (secure) communication channels.

How to minimize sampling overhead and maximize detection probability? An algorithmic question.
A Mental Model for This Talk

Challenge: centralization!
Central Controller Can Increase Attack Surface: E.g., May Be Exploited For Covert Communication

- Controllers react to switch events (packet-ins, link failures, etc.) for MAC learning, support mobility, VM migration, failover, etc.

- Reaction: send flowmods, packet-outs, performing path-paving...

- Triggering such events may be exploited for (covert) communication or even port scans, etc. even in presence of firewall/IDS/...
Teleportation

- May be used to **bypass firewall**

- Not easy to detect:
  - Traffic follows **normal pattern of control communication**, indirectly via controller
  - Teleportation channel is inside (encrypted) OpenFlow channel

- Need e.g., to **correlate** packet-ins, packet-outs, flow-mods, etc.
Controller has communication channels to all network elements: could be exploited to spread a virus and compromise entire datacenter.

Case study in OvS.
Pave-Path Technique
Controller performs **MAC learning** and updates paths to support **mobility**, **VM migration**, etc.

Example: If host X appears on new switch, controller installs **new rules** on new switch and removes on old switch

① **S₁** announces address X
Controller performs MAC learning and updates paths to support mobility, VM migration, etc.

Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

1. $S_1$ announces address X

E.g., 0xBADDAD
Controller performs MAC learning and updates paths to support mobility, VM migration, etc.

Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch.

1. $S_1$ announces address X
Controller performs MAC learning and updates paths to support mobility, VM migration, etc.

Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

1. $S_1$ announces address X
2. Packet-out to $h_1$
3. Add rule for X
Controller performs MAC learning and updates paths to support mobility, VM migration, etc.

Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

1. $S_1$ announces address X
2. Packet-out to $h_1$
3. Add rule for X
4. $S_2$ announces X
Controller performs **MAC learning** and updates paths to support mobility, VM migration, etc.

Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch.

1. **S₁** announces address X
2. Packet-out to **h₁**
3. **S₂** announces X

Learns: X on **S₂**

1. **S₁** announces address X
2. Packet-out to **h₁**
3. Add rule for X
4. **S₂** announces X

Diagram showing the process of teleportation: Path Update.
Controller performs MAC learning and updates paths to support mobility, VM migration, etc.

Example: If host X appears on new switch, controller installs new rules on new switch and removes on old switch

1. \(S_1\) announces address X
2. Packet-out to \(h_1\)
3. Add rule for X
4. \(S_2\) announces X
5. Flow delete

Learns: X on \(S_2\)
Controller performs MAC learning and updates paths to support mobility, VM migration, etc.

Learns: X on $S_2$

Admittedly very implicit: need to modulate information e.g., using timing or order of MAC addresses

Similar implicit teleportation based on mutual-exclusion. E.g., switches try same Datapath-ID (DPID) field in the Features-reply message.
Teleportation: Out-of-Band Forwarding

- E.g., exploiting ONOS Intent Reactive Forwarding (ifwd)
- By default, ifwd installs host-to-host connectivity when receiving a packet-in for which no flows exist (using path-pave technique)

Packet-in

1. Packet-in (X→h₂)

Knows: h₂ on S₂

DENY: h₁ ↔ h₂
Teleportation: Out-of-Band Forwarding

- E.g., exploiting ONOS Intent Reactive Forwarding (ifwd)
- By default, ifwd installs host-to-host connectivity when receiving a packet-in for which no flows exist (using path-pave technique)

1. Packet-in
2. Packet-out

Knows: $h_2$ on $S_2$

Packet-in ($X \rightarrow h_2$)

DENY: $h_1 \leftrightarrow h_2$
Teleportation: Out-of-Band Forwarding

- E.g., exploiting ONOS Intent Reactive Forwarding (ifwd)
- By default, ifwd installs host-to-host connectivity when receiving a packet-in for which no flows exist (using path-pave technique)

1. Packet-in
2. Packet-out
3. Flow-mod

Establish path through firewall: no more packet-ins, blocked. (But could use another MAC address next time.)
Teleportation: Out-of-Band Forwarding

- E.g., exploiting ONOS Intent Reactive Forwarding (ifwd)

- By default, ifwd installs host-to-host connectivity when receiving a packet-in for which no flows exist (using path-pave technique)

Further reading:
Outsmarting Network Security with SDN Teleportation
Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid.

1. Packet-in
2. Packet-out
3. Flow-mod

Establish path through firewall: no more packet-ins, blocked. (But could use another MAC address next time.)
A Mental Model for This Talk

Challenge: centralization!

Despite centralization: SDN stays a distributed system!
Challenge: Controller may miss events

- Basic task: MAC learning

- Principle: for packet \((src,dst)\) arriving at port \(p\)
  - If \(dst\) unknown: broadcast packets to all ports
  - Otherwise forward directly to known port
  - Also: if \(src\) unknown, switch learns: \(src\) is behind \(p\)
Challenge: Controller may miss events

- Basic task: MAC learning

- Principle: for packet \((src, dst)\) arriving at port \(p\)
  - If \(dst\) unknown: broadcast packets to all ports
  - Otherwise forward directly to known port
  - Also: if \(src\) unknown, switch learns: \(src\) is behind \(p\)

- Example
  - h1 sends to h2:
Challenge: Controller may miss events

- Basic task: MAC learning

- Principle: for packet $(src, dst)$ arriving at port $p$
  - If $dst$ unknown: broadcast packets to all ports
  - Otherwise forward directly to known port
  - Also: if $src$ unknown, switch learns: $src$ is behind $p$

- Example
  - h1 sends to h2: flood
Challenge: Controller may miss events

- Basic task: MAC learning

- Principle: for packet \((\text{src}, \text{dst})\) arriving at port \(p\)
  - If \(\text{dst}\) unknown: broadcast packets to all ports
  - Otherwise forward directly to known port
  - Also: if \(\text{src}\) unknown, switch learns: \(\text{src}\) is behind \(p\)

- Example
  - h1 sends to h2: flood, learn \((h1,p1)\)
Challenge: Controller may miss events

- Basic task: MAC learning

- Principle: for packet \((\text{src}, \text{dst})\) arriving at port \(p\)
  - If \(\text{dst}\) unknown: broadcast packets to all ports
  - Otherwise forward directly to known port
  - Also: if \(\text{src}\) unknown, switch learns: \(\text{src}\) is behind \(p\)

- Example
  - h1 sends to h2: flood, learn \((\text{h1}, \text{p1})\)
  - h3 sends to h1: forward to \(\text{p1}\)

\[
\text{dstmac}=\text{h1}, \text{fwd}(1)
\]
Challenge: Controller may miss events

- Basic task: MAC learning

- Principle: for packet \((src, dst)\) arriving at port \(p\)
  - If \(dst\) unknown: broadcast packets to all ports
  - Otherwise forward directly to known port
  - Also: if \(src\) unknown, switch learns: \(src\) is behind \(p\)

- Example
  - \(h1\) sends to \(h2\): flood, learn \((h1, p1)\)
  - \(h3\) sends to \(h1\): forward to \(p1\), learn \((h3, p3)\)
  
  \[\text{dstmac}=h1, \text{fwd}(1)\]
  \[\text{dstmac}=h3, \text{fwd}(3)\]
Challenge: Controller may miss events

- Basic task: MAC learning

- Principle: for packet \((\text{src}, \text{dst})\) arriving at port \(p\)
  - If \(\text{dst}\) unknown: broadcast packets to all ports
  - Otherwise forward directly to known port
  - Also: if \(\text{src}\) unknown, switch learns: \(\text{src}\) is behind \(p\)

- Example
  - h1 sends to h2: \textit{flood, learn (h1,p1)}
  - h3 sends to h1: \textit{forward to p1, learn (h3,p3)}
  - h1 sends to h3: \textit{forward to p3}
Challenge: Controller may miss events

How to implement this behavior in SDN?
Example: SDN MAC Learning
Done Wrong

- Initially table: Send everything to controller

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>
Example: SDN MAC Learning
Done Wrong

- Initially table: Send everything to controller

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>

- When \texttt{h1} sends to \texttt{h2}:
Example: SDN MAC Learning
Done Wrong

- Principle: only send to ctrl if destination unknown

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>

- When h1 sends to h2:
  - Controller learns that h1@p1, updates table, and floods
Example: SDN MAC Learning

- Principle: only send to ctrl if destination unknown

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstmac=h1</td>
<td>Forward(1)</td>
</tr>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>

- Now assume **h2 sends to h1**: 
Example: SDN MAC Learning

- Principle: only send to ctrl if destination unknown

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstmac=h1</td>
<td>Forward(1)</td>
</tr>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>

- Now assume **h2 sends to h1:**
  - *Switch knows destination*: message forwarded to h1
  - **BUT**: No controller interaction, does **not learn about h2**: no new rule for h2
Example: SDN MAC Learning

- Done Wrong

- Principle: only send to ctrl if destination unknown

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstmac=h1</td>
<td>Forward(1)</td>
</tr>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>

- Now, when **h3 sends to h2**: 
Example: SDN MAC Learning

Done Wrong

- Principle: only send to ctrl if destination unknown

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstmac=h1</td>
<td>Forward(1)</td>
</tr>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>

Now, when h3 sends to h2:
- Dest unknown: goes to controller which learns about h3
- And then floods

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstmac=h3</td>
<td>Forward(3)</td>
</tr>
<tr>
<td>dstmac=h1</td>
<td>Forward(1)</td>
</tr>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>
Example: SDN MAC Learning
Done Wrong

- Principle: only send to ctrl if destination unknown

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstmac=h3</td>
<td>Forward(3)</td>
</tr>
<tr>
<td>dstmac=h1</td>
<td>Forward(1)</td>
</tr>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>

Now, if h2 sends to h3 or h1:
Example: SDN MAC Learning
Done Wrong

- Principle: only send to ctrl if destination unknown

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstmac=h3</td>
<td>Forward(3)</td>
</tr>
<tr>
<td>dstmac=h1</td>
<td>Forward(1)</td>
</tr>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>

- Now, if h2 sends to h3 or h1:
  - Destinations known: controller does not learn about h2
Example: SDN MAC Learning

- **Done Wrong**

- **Wrong Pattern**

- **Controller**

- **OpenFlow switch**

- **Controller**

- **Principle:** only send to ctrl if destination unknown

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstmac=h3</td>
<td>Forward(3)</td>
</tr>
<tr>
<td>dstmac=h1</td>
<td>Forward(1)</td>
</tr>
<tr>
<td>*</td>
<td>send to controller</td>
</tr>
</tbody>
</table>

**Ouch!** Controller cannot learn about h2 anymore: whenever h2 is source, destination is known. All future requests to h2 will all be flooded: inefficient!
Example: SDN MAC Learning
Done Wrong

Principle: only send to ctrl if destination unknown.

Controller

h1

h2

h3

How to efficiently detect such problems? And which rules to use to overcome them? An algorithmic problem!

Ouch! Controller cannot learn about h2 anymore: whenever h2 is source, destination is known. All future requests to h2 will all be flooded: inefficient!
There Are Many More Reasons Why A Controller May Have Inconsistent View

- Rules inserted using switch CLI
- Operator misconfigurations
- Software/hardware bugs
- Updates that have been acknowledged wrongfully
- Malicious behavior, etc.

A problem because like in security: at most as consistent as least consistent part!
There Are Many More Reasons Why A Controller May Have Inconsistent View

- Rules inserted using switch CLI
- Operator misconfigurations
- Software/hardware bugs
- Updates that have been acknowledged wrongfully
- Malicious behavior, etc.

Logical verification not enough: need active and passive testing. How to do this efficiently?

A problem because like in security: at most as consistent as least consistent part!
There Are Many More Reasons Why A Controller May Have Inconsistent View

- Rules inserted using switch CLI
- Operator misconfigurations
- Software/hardware bugs
- Updates that have been acknowledged wrongfully
- Malicious behavior, etc.

A problem because *like in security*: at most as consistent as least consistent part!

Further reading:
[Towards Meticulous Data Plane Monitoring](#) (Poster Paper)
Apoorv Shukla, Said Jawad Saidi, Stefan Schmid, Marco Canini, and Anja Feldmann.
**EuroSys** PhD Forum, Belgrade, Serbia, April 2017.
Bad News: Automated Testing and Verification Can Be Non-Trivial!

- Seemingly simple *reachability questions* are hard in SDN:
  - «Is it possible to reach egress port y from ingress port x for certain *header spaces*?»
  - Or what-if-analysis: «What is reachability matrix if there are $f$ link failures?»

- Tools like NetKat, UPPAAL, ...: *PSPACE complete*, tools like wNetKAT can even encounter *undecidability*!

- ... and this is only on the *logical level* and for *stateless* data planes!
  - Still need to actually *test dataplane* consistency (e.g., using packet generation)
  - What if dataplane is *stateful*?
Tractability of Automation/Verification

Even without failures: reachability test is **undecidable** in SDN!

**Proof:** Can emulate a Turing machine.
Even without failures: reachability test is **undecidable** in SDN!

**Proof:** Can emulate a Turing machine.

Self-loop: could be replaced by “dummy switch”.

Tractability of Automation/Verification
Even without failures, reachability test is **undecidable** in SDN!

Idea: packet header stores Turing machine configuration (tape, head, state).
Even without failures: reachability test is **undecidable** in SDN!

**Proof:** Can emulate a Turing machine.

Switch action: each time packet arrives, performs one Turing machine step and updates header.
Even without failures: reachability test is **undecidable** in SDN!

**Proof:** Can emulate a Turing machine.

Only if accept or reject, forwarded to out. Is it ever reached? Undecidable!
Even without failures: reachability test is **undecidable** in SDN!

**Proof:** Can emulate a Turing machine.

- Only if accept or reject, forwarded to out. Is it ever reached?

---

**Further reading:**

*WNetKAT: A Weighted SDN Programming and Verification Language*

Kim G. Larsen, Stefan Schmid, and Bingtian Xue.

Many Open Research Questions

- **Tradeoff** expressiveness of rule and verification complexity?

- Is it worth using less general rules so fast (automated) verification is possible?

- Example: **MPLS** is not hard to verify!

- What about more **programmable and stateful dataplanes**?
A Mental Model for This Talk

Challenge: Decoupling
A Mental Model for This Talk

Challenge: Decoupling

Asynchronous!
A Mental Model for This Talk

Challenge: Decoupling

Despite centralization: SDN stays a distributed system!

Credits: He et al., ACM SOSR 2015: without network latency
Example “Route Updates”: What can possibly go wrong?

Invariant: Traffic from untrusted hosts to trusted hosts via firewall!
Example “Route Updates”: What can possibly go wrong?

Controller Platform

untrusted hosts

trusted hosts

In NFV: Not necessarily deployed at edge!

Invariant: Traffic from untrusted hosts to trusted hosts via firewall!
Problem 1: Bypassed Waypoint

Invariant: Traffic from untrusted hosts to trusted hosts via **firewall**!
Problem 2: Transient Loop

Controller Platform

Invariant: Traffic from untrusted hosts to trusted hosts via firewall!
Tagging: A Universal Solution?

- Old route: red
- New route: blue
- 2-Phase Update:
  - Install blue flow rules internally
  - Flip tag at ingress ports

Tagging: A Universal Solution?

- Old route: red
- New route: blue
- 2-Phase Update:
  - Install blue flow rules internally
  - Flip tag at ingress ports

Where to tag? Header space? Overhead!

Cost of extra rules!

Time till new link becomes available!

Tagging: A Universal Solution?

- Old route: red
- New route: blue

2-Phase Update:
- Install blue flow rules internally
- Flip tag at ingress ports

Cost of extra rules!

Where to tag? Header space? Overhead!

Possible solution without tagging, and at least preserve weaker consistency properties?

Time till new link becomes available!

Idea: Schedule “Safe” Subsets of Nodes Only, Then Wait for ACK!

Packet may take a mix of old and new path, as long as, e.g., Loop-Freedom (LF) and Waypoint Enforcement (WPE) are fulfilled

Round 1
Controller Platform

Round 2
Controller Platform

...
Loop-Free Update Schedule

insecure Internet → secure zone
Loop-Free Update Schedule

R1:
insecure Internet → R2: insecure Internet → secure zone

Forward edges (wrt old policy)! Always safe.

R2:
insecure Internet → secure zone

Backward edge: risky!
Loop-Free Update Schedule

R1:
- insecure Internet → red square → fire icon → secure zone
- LF ok! But: WPE violated in Round 1!

R2:
- insecure Internet → gray square → fire icon → secure zone

Forward edges (wrt old policy)! Always safe.

Backward edge: risky!
Waypoint Respecting Schedule
Waypoint Respecting Schedule

Don’t cross the waypoint: safe!
Waypoint Respecting Schedule

... ok but may violate LF in Round 1!
Can we have both LF and WPE?
Yes: but it takes 3 rounds!
Yes: but it takes 3 rounds!

Is there always a WPE+LF schedule?
What about this one?
LF and WPE may conflict!

- Cannot update any **forward edge** in R1: WP
- Cannot update any **backward edge** in R1: LF

No schedule exists!
Resort to tagging...
What about this one?
What about this one?

- Forward edge after the waypoint: safe!
- No loop, no WPE violation
What about this one?

Now this backward is safe too!

- No loop because exit through 1
What about this one?

Now this is safe: 2 ready back to WP!

No waypoint violation
What about this one?

- Ok: loop-free and also not on the path (exit via 1)
What about this one?

Ok: loop-free and also not on the path (exit via 1)
What about this one?
Back to the start: What if....
Back to the start: What if…. also this one?!
Back to the start: What if.... also this one?!

Update any of the 2 backward edges? LF 😞
Back to the start: What if... also this one?!

☐ Update any of the 2 backward edges? LF 😞
Back to the start: What if.... also this one?!

☐ Update any of the 2 backward edges? LF 😞
Back to the start: What if.... also this one?! 

- Update any of the 2 backward edges? LF 😞
- Update any of the 2 other forward edges? WPE 😞
- What about a combination? No...
In General: NP-Hard!

Bad news: Even decidability hard: cannot quickly test feasibility and if infeasible resort to say, tagging solution!

Open question: What is complexity in „typical networks“, like datacenter or enterprise networks?
What about loop-freedom only? Always possible in $n$ rounds!
What about **loop-freedom only**?
Always possible in $n$ rounds!

From the destination! Invariant: path suffix updated!
What about **loop-freedom only**?
Always possible in $n$ rounds!

From the destination! Invariant: path suffix updated!
What about loop-freedom only?
Always possible in $n$ rounds!

From the destination! Invariant: path suffix updated!
What about loop-freedom only? Always possible in $n$ rounds!

From the destination! Invariant: path suffix updated!
What about **loop-freedom only**?
Always possible in *n* rounds!

From the destination! Invariant: path suffix updated!
But how to minimize # rounds?
But how to minimize # rounds?

2 rounds easy, 3 rounds NP-hard. Everything else: We don’t know today!
What about capacity constraints?
What about capacity constraints?

Flow 1
What about capacity constraints?

Can you find an update schedule?

Flow 1
Flow 2
What about capacity constraints?

e.g., cannot update red: congestion! Need to update blue first!

Can you find an update schedule?
What about capacity constraints?

Round 1: prepare

Schedule:
1. red@w, blue@u, blue@v
What about capacity constraints?

Round 2

Schedule:
1. red@w, blue@u, blue@v
2. blue@s
What about capacity constraints?

Round 3

Capacity 2: ok!
No flow!

Schedule:
1. red@w, blue@u, blue@v
2. blue@s
3. red@s
What about capacity constraints?

Schedule:
1. red@w, blue@u, blue@v
2. blue@s
3. red@s
4. blue@w

Round 4
What about capacity constraints?

Note: this (non-trivial) example was just a DAG, without loops!

Schedule:
1. red@w, blue@u, blue@v
2. blue@s
3. red@s
4. blue@w

Round 4
Many Open Problems!

- We know for DAG:
  - For $k=2$ flows, polynomial-time algorithm to compute schedule with minimal number of rounds!
  - For general $k$, NP-hard
  - For general $k$ flows, polynomial-time algorithm to compute feasible update

- Everything else: unknown!
  - In particular: what if flow graph is not a DAG?
What’s new about this problem?

- Much classic literature on, e.g.,
  - Disruption-free IGP route changes
  - *Ship-in-the-Night* techniques

- SDN: new *model* (centralized and direct control of routes) and new *properties*
- Not only *connectivity consistency* but also *policy consistency* (e.g., waypoints) and *performance consistency*

*Further reading:* 35-page survey!

**Survey of Consistent Network Updates**
Further Reading:

**Can't Touch This: Consistent Network Updates for Multiple Policies**
Szymon Dudycz, Arne Ludwig, and Stefan Schmid.
46th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Toulouse, France, June 2016.

**Transiently Secure Network Updates**
Arne Ludwig, Szymon Dudycz, Matthias Rost, and Stefan Schmid.
42nd ACM SIGMETRICS, Antibes Juan-les-Pins, France, June 2016.

**Scheduling Loop-free Network Updates: It's Good to Relax!**
Arne Ludwig, Jan Marcinkowski, and Stefan Schmid.

**Good Network Updates for Bad Packets: Waypoint Enforcement Beyond Destination-Based Routing Policies**
Arne Ludwig, Matthias Rost, Damien Foucard, and Stefan Schmid.
13th ACM Workshop on Hot Topics in Networks (HotNets), Los Angeles, California, USA, October 2014.

**Congestion-Free Rerouting of Flows on DAGs**
Saeed Akhoondian Amiri, Szymon Dudycz, Stefan Schmid, and Sebastian Wiederrecht.

**Survey of Consistent Network Updates**
Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio.
A Mental Model for This Talk

Challenge: How to maintain connectivity?
In-band Management

How to provide connectivity between the planes?

- In large-scale networks: distributed...

- ... and inband control
  - Control and data plane traffic interleaved: arrives at the same port
  - No need for dedicated infrastructure

Ideally, self-stabilizing: ensure channels to switches and between controllers *from any initial state!*
Self-Stabilizing Connectivity is Non-Trivial

Ctrl A  Mgmt port  Switch  Ctrl Module  rules  Switch Fabric  Data plane links

Ctrl Module  updates & stats

internal link for inband control

Data plane links

Ctrl B

Data plane links
Self-Stabilizing Connectivity is Non-Trivial

Out-of-band control traffic

Ctrl A

Mgmt port

Switch

Ctrl Module

updates & stats

rules

Switch Fabric

internal link for inband control

Dataplane traffic

Data plane links

Data plane links

Ctrl B

In-band control traffic
Self-Stabilizing Connectivity is Non-Trivial

Out-of-band control traffic

Switch

Ctrl Module

Mgmt port

updates & stats

rules

internal link for inband control

Data plane links

Dataplane traffic

In-band control traffic

Data plane links

How to distinguish ctrl and data traffic?
E.g., tagging!

Ctrl A

Ctrl B
Self-Stabilizing Connectivity is Non-Trivial

How can Ctrl B know that Ctrl A failed?

Data plane links

Mgmt port

internal link for inband control

updates & stats

rules

Switch Fabric

Ctrl Module

Switch

Ctrl B

Back at

Data plane

Ctrl A

How can Ctrl B know that Ctrl A failed?
Self-Stabilizing Connectivity is Non-Trivial

How to get out of bad configurations?

If Match=*: Drop!
Self-Stabilizing Connectivity is Non-Trivial

How to get out of bad configurations?

Further reading:
A Self-Organizing Distributed and In-Band SDN Control Plane (Poster Paper)
Marco Canini, Iosif Salem, Liron Schiff, Elad M. Schiller, and Stefan Schmid.
37th IEEE International Conference on Distributed Computing Systems (ICDCS), Atlanta, Georgia, USA, June 2017.

If Match=*: Drop!
A Mental Model for This Talk

Opportunity: innovative services and algorithms
Example Benefit 1: Lying

- Cannot only find innovative routing algorithms etc. ...
- ... but also interact with and manipulate legacy networks in novel ways («hybrid SDNs»)
- E.g., trick it into better traffic engineering, faster failover, etc.
Example Benefit 1: Lying

STP in legacy network: loop-free

Improved capacity: STP in legacy network still loop-free
Example Benefit 1: Lying

STP in legacy network:

Further reading:
SHEAR: A Highly Available and Flexible Network Architecture: Marrying Distributed and Logically Centralized Control Planes
Michael Markovitch and Stefan Schmid.
23rd IEEE International Conference on Network Protocols (ICNP), San Francisco, California, USA, November 2015.
Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks
Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann.
Example Benefit 2: Flexible Waypoint Routing

For example, service chain: traffic is steered (e.g., using SDN) through a sequence of (virtualized) middleboxes to compose a more complex network service.
What is new and interesting here?

Generalizes call admission!

Which calls to admit? And how to route them? Limited resources!
What is new and interesting here?

Online call admission:

H → B
B → K
F → B
F → M

time

Admit and route requests through waypoints:

A → Gear → C
10 Gbps

A → Gear → B
1 Gbps

C → Gear → Database → B
15 Gbps

B → Database → Gear → A
8 Gbps
What is new and interesting here?

Online call admission:

Admit and route requests through waypoints:

Harder than embedding segments like in calls: need to admit **all or no** segment!
What do we know today... about complex requests?

How to embed s.t. resource footprint is minimal?
What do we know today...
...about complex requests?

How to embed s.t. resource footprint is minimal?

Fairly well-understood if approximations are allowed. E.g., reduce to flow problem using a product graph (and randomized rounding)!
What do we know today... 
... *about complex requests*?

Approximate function chain embedding: fairly well-understood
What do we know today... about complex requests?

What about if requests allow for alternatives and different decompositions?

Approximate function chain embedding: fairly well-understood
What do we know today... 
... about complex requests?

Approximate function chain embedding: fairly well-understood

What about if requests allow for alternatives and different decompositions?

Known as PR (Processing and Routing) Graph: allows to model different choices and implementations!
What do we know today... about complex requests?

Approximate function chain embedding: fairly well-understood

Further reading:
*An Approximation Algorithm for Path Computation and Function Placement in SDNs*
Guy Even, Matthias Rost, and Stefan Schmid.
23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO), Helsinki, Finland, July 2016.

Known as PR (Processing and Routing) Graph: allows to model different choices and implementations!
IETF Draft:

- Service chain for mobile operators
- Load-balancers are used to route (parts of) the traffic through cache

Credits: https://tools.ietf.org/html/draft-ietf-sfc-use-case-mobility-06
Service chain for mobile operators

Load-balancers are used to route (parts of) the traffic through cache

Has loops: the standard approach no longer works! There are first insights on advanced techniques for such graphs, but it’s an open question how far they can be pushed.

Credits: https://tools.ietf.org/html/draft-ietf-sfc-use-case-mobility-06
What about this one?!

IETF Draft:

- Service chain for mobile operators
- Load-balancers are used to route (parts of) the traffic through cache

Has loops: the standard approach **no longer works**! There are first insights on advanced techniques for such graphs, but it’s an **open question** how far they can be pushed.

Further reading:

[Service Chain and Virtual Network Embeddings: Approximations using Randomized Rounding](https://arxiv.org/)

Matthias Rost and Stefan Schmid.

Credits: https://tools.ietf.org/html/draft-ietf-sfc-use-case-mobility-06
Example: admission control and embedding

Substrate:

10 Gbps   5 Gbps
C

10 Gbps
A

Which ones can be admitted and embedded?
Example: admission control and embedding

Substrate:

10 Gbps

Requests:

A → C

B → C

Which ones can be admitted and embedded?
Example: admission control and embedding

Substrate:

Requests:

Which ones can be admitted and embedded?
Example: admission control and embedding

Substrate:

Requests:

Which ones can be admitted and embedded?
Example: admission control and embedding

Substrate:

Requests:

Which ones can be admitted and embedded?
Example: admission control and embedding

Substrate:

Requests:

Which ones can be admitted and embedded?
Good News 1: If approximation is good enough, can use product graphs and randomized rounding *for “Fairly Simple” Requests!*

Chains, alternative chains, but even trees. Trick: **reduction to flow problem using product graphs.**
Good News 1: If approximation is good enough, can use product graphs and randomized rounding for “Fairly Simple” Requests!

Substrate:

Product graph:
Good News 1: If approximation is good enough, can use product graphs and randomized rounding for "Fairly Simple" Requests!

Substrate:

Product graph:

\( i^{\text{th}} \) request \( r_i \):

Placement constraint

Super-source

Copy graph for each edge of chain
Good News 1: If approximation is good enough, can use product graphs and randomized rounding for “Fairly Simple” Requests!

Substrate:

Product graph:

Routing edge: graph edge on same layer

Processing edge: processing happens on C: connect C to C in next layer!
Good News 1: If approximation is good enough, can use product graphs and randomized rounding for “Fairly Simple” Requests!

Substrate:

Product graph:

Any valid \((s_i, t_i)\) path presents a valid realization of the request \(r_i\)!
Good News 1: If approximation is good enough, can use product graphs and randomized rounding for “Fairly Simple” Requests!

Substrate:

Product graph:

Any valid $(s_i, t_i)$ path presents a valid realization of the request $r_i$!
Good News 1: If approximation is good enough, can use product graphs and randomized rounding for “Fairly Simple” Requests!

Substrate:

Product graph:

Any valid \((s_i, t_i)\) path presents a valid realization of the request \(r_i\)!
Good News 1: If approximation is good enough, can use product graphs and randomized rounding for “Fairly Simple” Requests!

This problem can be solved using mincost unsplittable multi-commodity flow (approximation) algorithms (e.g., randomized rounding).

Any valid \((s_i,t_i)\) path presents a valid realization of the request \(r_i\)!
Good News 1: If approximation is good enough, can use product graphs and randomized rounding for “Fairly Simple” Requests!

Substrate:  

\[ \text{i}^{\text{th}} \text{ request } r_i: \]

This problem can be solved using mincost unsplittable multi-commodity flow (approximation) algorithms (e.g.,

Further reading:
An Approximation Algorithm for Path Computation and Function Placement in SDNs
Guy Even, Matthias Rost, and Stefan Schmid.
23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO), Helsinki, Finland, July 2016

Any valid \((s_i, t_i)\) path presents a valid realization of the request \(r_i\)!
What if requests arrive over time?
Can we admit and embed requests efficiently?
Good News 2: Yes, given offline embedding algorithm, *can do it online, over time, as well!*

Even without knowing anything about future requests, we can approximate an optimal offline solution that knows the future.
The Buchbinder-Naor Approach

Primal and Dual

\[
\begin{align*}
\text{(I)} \\
\min Z_j^T \cdot 1 + X^T \cdot C \\ Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
X, Z_j \geq 0
\end{align*}
\]

\[
\begin{align*}
\text{(II)} \\
\max B_j^T \cdot Y_j \\ A_j \cdot Y_j \leq C \\
D_j \cdot Y_j \leq 1 \\
Y_j \geq 0
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \( j \)th round:

1. \( f_{j,\ell} \leftarrow \arg\min\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   (a) \( y_{j,\ell} \leftarrow 1 \).
   (b) For each row \( e \) : If \( A_{e,\gamma(j,\ell)} \neq 0 \) do
      \[ x_e \leftarrow x_e \cdot 2^{A_{e,\gamma(j,\ell)} / c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,\gamma(j,\ell)} / c_e} - 1). \]
   (c) \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   (a) \( z_j \leftarrow 0 \).
The Buchbinder-Naor Approach

**Algorithm**

**Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).** Upon the \( j \)th round:

1. \( f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\} \) \( \text{(oracle procedure)} \)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   (a) \( y_{j,\ell} \leftarrow 1 \).
   (b) For each row \( e \) : If \( A_{e,(j,\ell)} \neq 0 \) do
      \[
      x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).
      \]
   (c) \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   (a) \( z_j \leftarrow 0 \).

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Offline embedding!
The Buchbinder-Naor Approach

Primal and Dual

\[
\begin{align*}
\text{(I)} & \quad \min Z_j^T \cdot 1 + X^T \cdot C \quad \text{s.t.} \\
& \quad Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& \quad X, Z_j \geq 0 \\
\text{(II)} & \quad \max B_j^T \cdot Y_j \quad \text{s.t.} \\
& \quad A_j \cdot Y_j \leq C \\
& \quad D_j \cdot Y_j \leq 1 \\
& \quad Y_j \geq 0
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \( j \)th round:

1. \( f_{j,e} \leftarrow \arg \min \{ \gamma(j, \ell) : f_{j,e} \in \Lambda \} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   (a) \( y_j, \ell \leftarrow 1 \).
   (b) For each row \( e \) : If \( A_{e,(j,\ell)} \neq 0 \) do
     \[
x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).
\]
   (c) \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   (a) \( z_j \leftarrow 0 \).
The Buchbinder-Naor Approach

\[
\begin{align*}
\text{min } & Z_j^T \cdot 1 + X^T \cdot C \quad \text{s.t.} \\
& Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& X, Z_j \geq 0 \\
\text{max } & B_j^T \cdot Y_j \quad \text{s.t.} \\
& A_j \cdot Y_j \leq C \\
& D_j \cdot Y_j \leq 1 \\
& Y_j \geq 0 \\
\end{align*}
\]

(I) \quad (II)

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \( j \)th round:

1. \( f_{j,e} \leftarrow \arg\min \{\gamma(j, \ell) : f_{j,e} \in A_j\} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   a. \( y_{j,\ell} \leftarrow 1 \).
   b. For each row \( e \) : If \( A_{e,(j,\ell)} \neq 0 \) do
      \[
x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)} / c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)} / c_e} - 1).
\]
   c. \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   a. \( z_j \leftarrow 0 \).

Embedding cost vs profit?
The Buchbinder-Naor Approach

Primal and Dual

Fairly well-understood! Some caveats! 😊

Further reading:
Competitive and Deterministic Embeddings of Virtual Networks

Algorithm
Upon the jth round:

1. \( f_{j,e} \leftarrow \arg \min \{ \gamma(j, \ell) : f_{j,e} \in \Lambda \} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   (a) \( y_{j,\ell} \leftarrow 1 \).
   (b) For each row \( e : A_{e,(j,\ell)} \neq 0 \) do

\[
x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).
\]

(c) \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   (a) \( z_j \leftarrow 0 \).
What about using randomized rounding?

- **Problem 1:** relaxed solutions may not be very meaningful
  - see example for splittable flows before

- **Problem 2:** also for unsplittable flows, if using a standard *Multi-Commodity Flow (MCF)* formulation of VNEP, the integrality gap can be huge
  - Tree-like VNets are still ok
  - VNets with cycles: randomized rounding not applicable, since problem not decomposable

The linear solutions can be decomposed into convex combinations of valid mappings.
Non-Decomposability

- Relaxations of classic MCF formulation cannot be decomposed into convex combinations of valid mappings (so we need different formulations!)

- Example:
Non-Decomposability

Relaxations of classic MCF formulation cannot be decomposed into convex combinations of valid mappings (so we need different formulations!)

Example:

Valid LP solution: virtual node mappings sum to 1 and each virtual node connects to its neighboring node with half a unit of flow...

LP Solution
Impossible to decompose and extract any single valid mapping. Intuition: Node i is mapped to \( u_1 \) and the only neighboring node that hosts j is \( u_2 \), so i must be fully mapped on \( u_1 \) and j on \( u_2 \). Similarly, k must be mapped on \( u_3 \). But flow of virtual edge (k,i) leaving \( u_3 \) only leads to \( u_4 \), so i must be mapped on both \( u_1 \) and \( u_4 \). This is impossible, even if capacities are infinite.

Example:
Impossible to decompose and extract any single valid mapping. Intuition: Node $i$ is mapped to $u_1$ and the only neighboring node that hosts $j$ is $u_2$, so $i$ must be fully mapped on $u_1$ and $j$ on $u_2$. Similarly, $k$ must be mapped on $u_3$. But flow of virtual edge $(k,i)$ leaving $u_3$ only leads to $u_4$, so $i$ must be mapped on both $u_1$ and $u_4$. This is impossible, even if capacities are infinite.

Example:

How to devise a Linear Programming formulations, such that convex combinations of valid mappings can be recovered?!
Impossible to decompose and extract **any single valid mapping**. **Intuition:** Node $i$ is mapped to $u_1$ and the only neighboring node that hosts $j$ is $u_2$, so $i$ must be fully mapped on $u_1$ and $j$ on $u_2$. Similarly, $k$ must be mapped on $u_3$. But flow of virtual edge (k,i) leaving $u_3$ only leads to $u_4$, so **$i$ must be mapped on both $u_1$ and $u_4$**. This is **impossible**, even if capacities are infinite.

- **Example:**

  **LP Solution**

  ![Diagram](https://via.placeholder.com/150)

  Further reading:
  
  **An Approximation Algorithm for Path Computation and Function Placement in SDNs**
  
  Guy Even, Matthias Rost, and Stefan Schmid.
  
  23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO), Helsinki, Finland, July 2016

  that convex combinations of valid mappings can be recovered?!
Novelty:

- Traditionally: routes form simple paths (e.g., shortest paths)
- Now: routing through middleboxes may require more general paths, with loops: a walk

How to compute a shortest route through a waypoint?
Comuting A Shortest Walk Through A Single Given Waypoint is Non-Trivial!

Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!
Computing A Shortest Walk Through A Single Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

*Greedy fails:* choose shortest path from s to w...
Comuting A Shortest Walk Through A Single Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

Greedy fails: ... now need long path from w to t
Computing A Shortest Walk Through A Single Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

```
Comuting A Shortest Walk Through A Single Given Waypoint is Non-Trivial!

Greedy fails: ... now need long path from w to t
```

![Diagram showing a network with nodes and edges, illustrating the computation of shortest routes through waypoints.](image)
Comuting A Shortest Walk Through A Single Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

**Total length:** $4+2=6$

A *better solution*: jointly optimize the two segments!
Comuting A Shortest Walk Through A Single Given Waypoint is Non-Trivial!

- Computing shortest routes through waypoints is non-trivial!

Assume unit capacity and demand for simplicity!

Similar to computing **shortest disjoint paths** (if capacities are 1, segments need to be disjoint): a well-known combinatorial problem!

NP-hard on directed networks (feasibility in P on undirected networks, optimality unknown).

A **better solution**: **jointly** optimize the two segments!
NP-hard on Directed Networks: Reduction from Disjoint Paths Problem

Reduction: *From* joint shortest paths \((s_1, t_1), (s_2, t_2)\) *to* shortest walk \((s, w, t)\) problem

*Fact:* computing 2-disjoint paths is NP-hard on directed graphs.

*We show:* If waypoint routing was be in P, we could solve it fast.

*Contradiction!*
NP-hard on Directed Networks: Reduction from Disjoint Paths Problem

**Reduction:** From joint shortest paths \((s_1,t_1),(s_2,t_2)\) to shortest walk \((s,w,t)\) problem

Reduction: To find shortest paths \((s_1,t_1),(s_2,t_2)\), introduce **waypoint w** and connect \(t_1\) to \(s_2\) via \(w\)....
NP-hard on Directed Networks:
Reduction from Disjoint Paths Problem

Reduction: *From* joint shortest paths \((s_1,t_1),(s_2,t_2)\) *to* shortest walk \((s,w,t)\) problem

... and ask for shortest waypoint route \((s_1,w,t_2)\)
Reduction: To find shortest paths \((s_1, t_1)\), \((s_2, t_2)\), introduce \text{waypoint} \(w\) and connect \(t_1\) to \(s_2\) via \(w\).
What about waypoint routes on *undirected* networks?
What about waypoint routes on *undirected* networks?

Option 1: If *feasibility* good enough: reduce it to disjoint paths problem!

- Replace capacitated links with undirected parallel links:

  ![Diagram of undirected network with parallel links](image)

- Even works for *multiple waypoints*: Feasibility in P for constant number of flows

- So each path segment becomes a (disjoint) path

![Diagram of multiple waypoints and path segments](image)
What about waypoint routes on *undirected* networks?

Option 1: If *feasibility* good enough: reduce it to disjoint paths problem!

- Replace capacitated links with undirected parallel links:

  Good news: For a single waypoint, *shortest* paths can be computed even *faster*!

- So each path segment becomes a (disjoint) path
Good news: Not NP-hard on Undirected Networks: Suurballe’s Algorithm

- Suurballe’s algorithm: finds two (edge-)disjoint shortest paths between same endpoints:
Suurballe’s algorithm: finds two (edge-)disjoint shortest paths between same endpoints:

Good news: Not NP-hard on Undirected Networks: Suurballe’s Algorithm

How to compute a shortest (s,w,t) route with this algorithm??
Good news: Not NP-hard on Undirected Networks: Suurballe’s Algorithm

- **Step 1:** replace capacities with parallel edges: paths will become edge-disjoint

```
  w  -->  s  -->  t  
     |     |     |
     2    2     |

  w  -->  s  -->  t
```

```
Waypoint Routing on Steroids

- **Step 2:** Reduction to Suurballe’s algorithm:

  To find shortest \((s,w,t)\) route...
Step 2: Reduction to Suurballe’s algorithm:

- Connect $S^+$ to $s$ and $t$.
- Connect $w$ to $T^+$.

Waypoint Routing on Steroids
Step 2: Reduction to Suurballe’s algorithm:

Waypoint Routing on Steroids

... ask Suurballe for 2 disjoint paths from S+ to T+...
Step 2: Reduction to Suurballe’s algorithm:

Solution! Undirected: direction does not matter.
Open Question

For which other service chains can we compute optimal embeddings fast?

Further reading:
Charting the Complexity Landscape of Waypoint Routing
Walking Through Waypoints
You: Great, I can embed service chains at low resource cost and providing minimal bandwidth guarantees!
You: Great, I can embed service chains at low resource cost and providing minimal bandwidth guarantees!

Boss: So can I promise our customers a predictable performance?
You: Great, I can embed service chains at low resource cost and providing minimal bandwidth guarantees!

Boss: So can I promise our customers a predictable performance?

You: hmmm....
The Many Faces of Performance Interference

Consider: 2 SDN-based virtual networks (vSDNs) sharing physical resources!

Assume: perfect performance isolation on the network!

An Experiment: 2 vSDNs with bw guarantee!
The Many Faces of Performance Interference

To enable multi-tenancy, take existing network hypervisor (e.g. Flowvisor, OpenVirteX): provides network abstraction and control plane translation!

An Experiment: 2 vSDNs with bw guarantee!
The Many Faces of Performance Interference

Intercepts control plane messages.

Translation could include, e.g., switch DPID, port numbers, ...

An Experiment: 2 vSDNs with bw guarantee!
The Many Faces of Performance Interference

It turns out: the network hypervisor can be source of unpredictable performance!

An Experiment: 2 vSDNs with bw guarantee!
The Many Faces of Performance Interference

Experiment: web latency depends on hypervisor CPU load!
Performance also depends on hypervisor type…

*multithreaded or not, which version of Nagle’s algorithm, etc.*)
Conclusion: For a predictable performance, a complete system model is needed! But this is hard: depends on specific technologies, uncertainties in demand, etc.
The Many Faces of Performance Interference

Performance also depends on hypervisor type…

(Number of tenants…

Further reading:

Logically Isolated, Actually Unpredictable? Measuring Hypervisor Performance in Multi-Tenant SDNs

uncertainties in demand, etc.)
A Mental Model for This Talk

Simple, open, verifiable
A Mental Model for This Talk

Really?!
A Mental Model for This Talk

Failover via controller too slow.
A Mental Model for This Talk

Failover via controller too slow.

OpenFlow allows to preconfigure conditional failover rules: 1st line of defense!
A Mental Model for This Talk

Failover via controller too slow.

The Crux: How to define conditional rules which have local failure knowledge only?

OpenFlow allows to preconfigure conditional failover rules: 1st line of defense!
A Mental Model for This Talk

Open problem: How many link failures can be tolerated in $k$-connected network without going through controller?

Failover via controller too slow.

The Crux: How to define conditional rules which have local failure knowledge only?

OpenFlow allows to preconfigure conditional failover rules: 1st line of defense!
Solution: Use Arborescences (Chiesa et al.)

- Assume:
  - $k$-connected network $G$
  - destination $d$
  - $G$ decomposed into $k$ $d$-rooted arc-disjoint spanning arborescences

**Known result:** always exist in $k$-connected graphs (efficient).

**Basic principle:**
- Route along fixed arborescence ("directed spanning tree") towards the destination $d$
- If packet hits a failed edge at vertex $v$, reroute along a different arborescence

The Crux: which arborescence to choose next? Influences resiliency!
Simple Example: Hamilton Cycle

Chiesa et al.: if \( k \)-connected graph has \( k \) arc disjoint Hamilton Cycles, \( k-1 \) resilient routing can be constructed!
Example: 3-Resilient Routing Function for 2-dim Torus

$\text{k}=4 \text{ connected}$
Example: 3-Resilient Routing Function for 2-dim Torus
Example: 3-Resilient Routing Function for 2-dim Torus

spans all nodes: each node visited exactly once!
Example: 3-Resilient Routing Function for 2-dim Torus
Example: 3-Resilient Routing Function for 2-dim Torus

Edge-disjoint: Together span all edges!
Example: 3-Resilient Routing Function for 2-dim Torus

Make Hamilton cycles directed: so 4 Arc-Disjoint Hamilton Cycles.
Failover: In order to reach destination $d$: go along 1$^{st}$ directed HC, if hit failure, reverse direction, if again failure switch to 2$^{nd}$ HC, if again failure reverse direction: no more failures possible!
Example: 3-Resilient Routing Function for 2-dim Torus

Torus 4-connected, has 4 arc disjoint Hamilton cycles, so can construct optimal 3-resilient routing!

Further reading:
Exploring the Limits of Static Failover Routing
Load-Aware Local Fast Failover: Non-Trivial Already in the Clique!
Local Fast Failover with Load

The network:
Local Fast Failover with Load

Traffic demand: \{1,2,3\} -> 6

Without failures!
Local Fast Failover

Traffic demand: \{1,2,3\} -> 6

Preinstalled failover rules for red flow
Local Fast Failover

Traffic demand: \{1,2,3\} -\> 6

Preinstalled failover rules for blue flow

Failover table:
flow 1 -\> 6: 2,3,4,5,...
flow 2 -\> 6: 3,4,5,...
Local Fast Failover

Traffic demand: \{1,2,3\}->6

Failover table:
flow 1->6: 2,3,4,5,…
flow 2->6: 3,4,5,…
flow 3->6: 4,5,…

Preinstalled failover rules for green flow
Local Fast Failover

Traffic demand: \{1,2,3\} -> 6

Local failover @1: Does not know failures downstream!
Local Fast Failover

Traffic demand:
{1,2,3}→6

Failover table:
flow 1→6: 2,3,4,5,…

Local failover @1: Does not know failures downstream!
Local Fast Failover

Traffic demand: 
\{1,2,3\}→6

Local failover @1: 
Reroute to 2!

Failover table:
flow 1→6: 2,3,4,5,…

Failover table:
flow 1→6: 2,3,4,5,…
Traffic demand: \( \{1,2,3\} \rightarrow 6 \)

But also from 2: 6 not reachable.

Next: 3.
Traffic demand: \{1,2,3\}\rightarrow6

Finally, 6 can be reached!

Failover table:
flow 1->6: 2,3,4,5,...
Local Fast Failover

Traffic demand: \{1,2,3\}\rightarrow 6

Failover table:
flow 1\rightarrow 6: 2,3,4,5,...
flow 2\rightarrow 6: 3,4,5,...
flow 3\rightarrow 6: 4,5,...

Similarly for the other two flows.
Local Fast Failover

Traffic demand: \{1,2,3\} \rightarrow 6

Max load: 3 😞
Local Fast Failover

Traffic demand: \{1,2,3\} -> 6

Failover table:
flow 1 -> 6: 2, 5, ...
flow 2 -> 6: 3, 4, 5, ...
flow 3 -> 6: 4, 5, ...

A better solution:
load 2 😊
Local Fast Failover

Traffic demand: \{1,2,3\} - > 6

Tables statically defined, without global failure knowledge: a local algorithm without communication!

A better solution: load 2 😊
Local Fast Failover

Traffic demand: \{1,2,3\}->6

In order to load balance: prefixes of rows should be different!

A better solution: load 2 😊
Local Fast Failover

Traffic demand:

Bad news (intriguing!): High load unavoidable even in well-connected residual networks: a price of locality. 

*Given L failures, load at least √L, although network still highly connected (n-L connected). E.g., L=n/2, load could be 2 still, but due to locality at least √n.*

A better solution: load 2 😊
Local Fast Failover

Traffic demand:

**Bad news (intriguing!):** High load unavoidable even in well-connected residual networks: a price of locality.

Given $L$ failures, load at least $\sqrt{L}$, although network still highly connected ($n-L$ connected). E.g., $L=n/2$, load could be 2 still, but due to locality at least $\sqrt{n}$.

**Good news:** Theory of local algorithms without communication: symmetric block design theory.

Failover table:
- flow 1->6: 2,5,...
- flow 2->6: 3,4,5,...
- flow 3->6: 4,5,...

A better solution: load 2 😊
Local Fast Failover

Traffic demand:

**Bad news (intriguing!):** High load unavoidable even in well-connected residual networks: a price of locality.

*Given L failures, load at least √L, although network still highly connected (n-L connected). E.g., L=n/2, load could be 2 still, but due to locality at least √n.*

**Good news:** Theory of local algorithms without communication: symmetric block design theory.

What about multihop networks? See Chiesa et al.
Local Fast Failover

Traffic demand:

Bad news (intriguing!): High load unavoidable even in well-connected residual networks: a price of locality. Given $L$ failures, load at least $\sqrt{L}$, although network still highly connected!

Further reading:
Load-Optimal Local Fast Rerouting for Dependable Networks
Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Denver, Colorado, USA, June 2017.

What about multihop networks? See Chiesa et al.
Open Problems

- Optimal resiliency on general networks
  - An open conjecture!

- Beyond resilience:
  - Stretch («space-filling curves»?)
  - Load
  - Combination

- Optimized to specific networks again
A Mental Model for This Talk

Increasingly virtualized
A Mental Model for This Talk

Challenge 2: security

Increasingly virtualized
Virtual switches reside in the **server’s virtualization layer** (e.g., Xen’s Dom0). Goal: provide connectivity and isolation.
Increasing Complexity:  
# Parsed Protocols

Number of parsed high-level protocols constantly increases:
Increasing Complexity: 
*Introduction of middlebox functionality*

**Increasing workloads** and advancements in network virtualization drive virtual switches to **implement middlebox functions** such as load-balancing, DPI, firewalls, etc.
Increasing Complexity: 
Unified Packet Parsing

How to parse all these protocols without lowering forwarding performance?!
Increasing Complexity:
Unified Packet Parsing

Unified packet parsing allows parse more and more protocols efficiently: in a single pass!
Increasing Complexity: Unified Packet Parsing

Unified packet parsing allows parse more and more protocols efficiently: in a single pass!

This centralization is fast! But more complex to get it right.

- Ethernet
- LLC
- VLAN
- MPLS
- IPv4
- ICMPv4
- TCP
- UDP
- ARP
- SCTP
- IPv6
- ICMPv6
- IPv6 ND
- GRE
- LISP
- VXLAN
- PBB
- IPv6 EXT HDR
- TUNNEL-ID
- IPv6 ND
- IPv6 EXT HDR
- IPv6HOPOPTS
- IPv6ROUTING
- IPv6Fragment
- IPv6DESTOPT
- IPv6ESP
- IPv6 AH
- RARP
- IGMP
Complexity: The Enemy of Security!

- Data plane security not well-explored (in general, not only virtualized): most security research on control plane

- Two conjectures:
  1. Virtual switches increase the attack surface.
  2. Impact of attack larger than with traditional data planes.
The Attack Surface: Closer...

Attack surface becomes closer:

- Packet parser typically integrated into the code base of virtual switch

- First component of the virtual switch to process network packets it receives from the network interface

- May process attacker-controlled packets!
The Attack Surface: ... More Complex ...

- Ethernet
- LLC
- VLAN
- MPLS
- IPv4
- ICMPv4
- TCP
- UDP
- ARP
- SCTP
- IPv6
- ICMPv6
- IPv6 ND
- IPv6 HOPOPTS
- IPv6 ROUTING
- IPv6 Fragment
- IPv6 DESTOPT
- IPv6 ESP
- RARP
- IGMP
- PBB
- IPv6 EXT HDR
- TUNNEL-ID
- IPv6 ND
- IPv6 EXT HDR
- IPv6 HOPOPTS
- IPv6 ROUTING
- IPv6 Fragment
- IPv6 DESTOPT
- IPv6 ESP
- IGMP

Ctrl

L2, L2.5, L3, L4

VM

L2, L2.5, L3, L4

L2, L2.5, L3, L4
Collocated (at least partially) with hypervisor’s Dom0 kernel space, guest VMs, image management, block storage, identity management, ...
Collocated (at least partially) with hypervisor’s Dom0 kernel space, guest VMs, image management, block storage, identity management, ...

... the controller itself.
Collocated with hypervisor's (Dom0 kernel space), guest VMs, image management, block storage, identity management, …

Available communication channels to (SDN/Openstack) controller! Controller needs to be reachable from all servers.

… the controller itself.
Larger Impact: Case Study OVS

1. Rent a VM in the cloud (cheap)
2. Send **malformed MPLS packet** to virtual switch (**unified parser** parses label stack packet beyond the threshold)
3. **Stack buffer overflow** in (unified) MPLS parsing code: enables remote code execution
4. Send malformed packet to server (virtual switch) where controller is located (use existing communication channel)
Larger Impact: Case Study OVS

5. Spread
A Novel Threat Model

- Limited skills required
  - Use standard fuzzer to find crashes
  - Construct malformed packet
  - Build ROP chain

- Limited resources
  - Rent a VM in the cloud

- No physical access needed

No need to be a state-level attacker to compromise the dataplane (and beyond)!

Similar problems in NFV: need even more complex parsing/processing. And are often built on top of OvS.
Countermeasures

- Software countermeasures already exist
  - but come at overhead

- Better designs
  - Virtualize dataplane components: decouple them from hypervisor?
  - Remote attestation for OvS Flow Tables?
  - Control plane communication firewalls?
  - ...

Further Reading

The vAMP Attack: Taking Control of Cloud Systems via the Unified Packet Parser

Reigns to the Cloud: Compromising Cloud Systems via the Data Plane
Conclusions

Opportunities

E.g., innovative services

Challenges

E.g., waypoint routing, traffic engineering
**Conclusions**

**Opportunities**
- E.g., decouping: evolve control plane independently of dataplane

**Challenges**
- E.g., keeping controller up-to-date
- E.g., consistent network update

**Diagram:**
- Control Programs
- Ctrl
- Dataplane nodes connected to Ctrl and Control Programs
Conclusions

Opportunities

- E.g., simple and open interface

Challenges

- E.g., functionality that should stay here?
- E.g., complexity of verification, local failover, ...?
Stepping Back Even A Little Bit More…

- SDN + virtualization offer great flexibilities: are enablers

- Exploiting and analyzing them is still complex:
  - Algorithms are non-trivial (e.g., waypoint routing)
  - Interfaces / abstractions / languages still quite low-level (e.g., configuration of conditional failover rules)
  - Networked systems are still complex and hard to model (e.g., hypervisor interference)
  - Many uncertainties: hardware, demand, interference

Maybe we need a different approach to networking? Self-adjusting, data-driven, machine-learning, … networks!
A Better Vision of Future Networked Systems?

Analogy to self-driving cars: more high-level task-, measurement-, data- and learning-driven rather than model-driven?

Also: self-stabilizing, self-adjusting, self-optimizing....
A Better Vision of Future Networked Systems?

Analogy to self-driving cars:
more high-level task, measurement, data- and learning-driven rather than model-driven?

Also: self-stabilizing, self-adjusting, self-optimizing....

Further Reading:
'o'zapft is: Tap Your Network Algorithm's Big Data!
Andreas Blenk, Patrick Kalmbach, Stefan Schmid, and Wolfgang Kellerer. ACM SIGCOMM 2017 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (Big-DAMA), Los Angeles, California, USA, August 2017.
Thank you! Questions?