Kirigami, the Verifiable Art of Network Cutting

Tim Alberdingk Thijm, Ryan Beckett, Aarti Gupta, David Walker

ICNP 2022
AWS revenue jumps 33%, but growth slows

As enterprises face a possible recession, will uptake of cloud services slow?

By Anirban Ghoshal
Senior Writer, InfoWorld | JUL 29, 2022 1:56 PM PDT

Microsoft weathers the financial storm with 12% revenue growth

Despite some tricky global headwinds, Microsoft continues to post strong results, buoyed by its cloud business, which surpassed $25 billion in quarterly revenue for the first time.
AWS revenue jumps 33%, but growth slows
As enterprises face a possible recession, will uptake of cloud services slow?

By Anirban Ghoshal

Microsoft weathers the financial storm with 12% revenue growth
Despite some tricky global headwinds, Microsoft continues to post strong results, buoyed by its cloud business, which surpassed $25 billion in quarterly revenue for the first time.

Cloudflare outage on June 21, 2022
2022-06-21

What is BGP, and what role did it play in Facebook’s massive outage

The Register

After config error takes down Rogers, it promises to spend billions on reliability
Routers flooded with internet traffic in filter blunder, watchdog told

Brandon Vogler

Mon 25 Jul 2022 18:45 UTC
Analyzing Distributed Control Planes

Lots of great work, including…

**Batfish** [Fogel et al., NSDI 2015]

**Bagpipe** [Weitz et al., OOPSLA 2016]

**Minesweeper** [Beckett, Gupta, Mahajan, Walker, SIGCOMM 2018]

**NV** [Giannarakis, Loehr, Beckett, Walker, PLDI 2020]

**Tiramisu** [Abhashkumar, Gember-Jacobson, Akella, NSDI 2020]

**Plankton** [Prabhu et al., NSDI 2020]

But all these tools must **analyze entire network at once!**
Analyzing Distributed Control Planes

Lots of great work, including…

**Batfish** [Fogel et al., NSDI 2015]

**Bagpipe** [Weitz et al., OOPSLA 2016]

**Minesweeper** [Beckett, Gupta, Mahajan, Walker, SIGCOMM 2018]

**NV** [Giannarakis, Loehr, Beckett, Walker, PLDI 2020]

**Tiramisu** [Abhashkumar, Gember-Jacobson, Akella, NSDI 2020]

**Plankton** [Prabhu et al., NSDI 2020]

But all these tools must **analyze entire network at once**!
Modularity Is Essential

Cloud providers have networks with millions of nodes, and they are growing...

Thinking about our networks one piece at a time makes them easier to reason over, and supports incremental changes and updates.
Modular Network Verification

Identify the network’s components

Annotate component boundaries with an interface

Break up the network into fragments to analyze separately
Modular Network Verification

Identify the network’s **components**

Annotate component boundaries with an **interface**

Break up the network into **fragments** to analyze separately
Modular Network Verification

Identify the network’s **components**

Annotate component boundaries with an **interface**

Break up the network into **fragments** to analyze separately
Modular Network Verification

Identify the network’s **components**

Annotate component boundaries with an **interface**

Break up the network into **fragments** to analyze separately
Modular Network Verification

Satisfiability Modulo Theories (SMT)-based verification time

- **Nonlinear** in size of network (worst case **exponential**)
- **Bottlenecks** analysis (NP-complete problem)

Splitting up network takes **linear time**

...but with **better-than-linear** improvements!

Our experiments saw SMT times **improve by over 100,000x**
Modular Network Verification

Satisfiability Modulo Theories (SMT)-based verification time

Nonlinear in size of network (worst case exponential)

Bottlenecks analysis (NP-complete problem)

Splitting up network takes linear time

...but with better-than-linear improvements!

Our experiments saw SMT times improve by over 100,000x
Modular Network Verification

network

interface

$a_4$ sends route $\langle d,3 \rangle$ to $c_0$

$c_1$ sends route $\langle d,2 \rangle$ to $a_5$

...
Modular Network Verification

network

interface

$a_4$ sends route $\langle d,3 \rangle$ to $c_0$
$c_1$ sends route $\langle d,2 \rangle$ to $a_5$

...
Modular Network Verification

Network sends route \( \langle d, 3 \rangle \) to \( c_0 \)

Interface

\( a_4 \) sends route \( \langle d, 3 \rangle \) to \( c_0 \)

\( c_1 \) sends route \( \langle d, 2 \rangle \) to \( a_5 \)

\( \ldots \)
Modular Network Verification

network

interface

a₄ sends route ⟨d,3⟩ to c₀

C₁ sends route ⟨d,2⟩ to a₅

...

network verifier

fragment 1

fragment 2
Modular Network Verification

Verification counterexamples localized to particular fragments.

- $a_4$ sends route $\langle d,3 \rangle$ to $c_0$
- $c_1$ sends route $\langle d,2 \rangle$ to $a_5$
- ...
Modular Network Verification

Users can specify arbitrary cuts for fragments of different granularities, accommodating annotation cost.
Our Contributions

A theory of network interfaces and fragments

based on assume-guarantee reasoning,

proved fragment verification sound and complete w.r.t. monolithic verification

A checking procedure to verify properties using fragments

and to check if a given interface is correct

An extension Kirigami for the network verification language NV

evaluated on benchmarks, with over 100,000x improvement in SMT time
Our Contributions

A theory of network interfaces and fragments

based on assume-guarantee reasoning,

proved fragment verification sound and complete w.r.t. monolithic verification

A checking procedure to verify properties using fragments

and to check if a given interface is correct

An extension Kirigami for the network verification language NV

evaluated on benchmarks, with over 100,000x improvement in SMT time
Our Contributions

A theory of network interfaces and fragments
based on assume-guarantee reasoning,
proved fragment verification sound and complete w.r.t. monolithic verification

A checking procedure to verify properties using fragments
and to check if a given interface is correct

An extension Kirigami for the network verification language NV
evaluated on benchmarks, with over 100,000x improvement in SMT time
Roadmap

An Example Modular Verification Problem
A Theory of Network Fragments
Implementation
Evaluation
Takeaways
Verifying a Data Center
The Stable Routing Problem

topology graph with nodes \( V \) and edges \( E \)

\[
G = (V, E)
\]
Verifying a Data Center
The Stable Routing Problem

topology graph with nodes $V$ and edges $E$

$G = (V, E)$

routes (routing announcements) $\langle p, x \rangle$:
identifier $p$ and a cost metric $x$
Verifying a Data Center
The Stable Routing Problem

Suppose \( e_6 \) announces a route to itself.
Verifying a Data Center

The Stable Routing Problem

Other nodes start with no route ($\infty$).
Nodes broadcast updated routes to all neighbors, incrementing the route’s cost.
Verifying a Data Center
The Stable Routing Problem

Nodes compare received routes to select the route with the smallest cost.
Verifying a Data Center
The Stable Routing Problem

...and so on...
Verifying a Data Center
The Stable Routing Problem

...and so on...

\[
\langle e_6,0 \rangle \quad \langle e_6,1 \rangle \\
\langle e_6,0 \rangle \quad \langle e_6,2 \rangle \\
\langle e_6,0 \rangle \quad \langle e_6,1 \rangle \\
\langle e_6,0 \rangle \quad \langle e_6,1 \rangle \\
\langle e_6,2 \rangle \quad \langle e_6,2 \rangle \\
\langle e_6,2 \rangle \quad \langle e_6,2 \rangle \\
\langle e_6,2 \rangle \quad \langle e_6,2 \rangle \\
\langle e_6,2 \rangle \\
\]

\[\infty\]
Verifying a Data Center
The Stable Routing Problem

until every node has a stable, locally-best route (a solution)
Verifying a Data Center
The Stable Routing Problem

Routing converges to **network solution**

Check **properties** on nodes’ solutions

- all-pairs path length
- for any choice of identifier $p$, all nodes converge to a route $\langle p, x \rangle$ with a metric $x \leq 4$. 
Verifying a Data Center
The Stable Routing Problem

Routing converges to network solution
Check properties on nodes’ solutions
all-pairs path length
for any choice of identifier $p$, all
nodes converge to a route $\langle p, x \rangle$ with
a metric $x \leq 4$. 
Routing converges to network solution

Check properties:

- all-pairs path length:
  - for any choice of identifier \( p \), all nodes converge to a route \( \langle p, x \rangle \) with a metric \( x \leq 4 \).

Let’s modularize it!
Cut fattree SRP $S$ into fragments

each pod $i$ in its own fragment $T_{pi}$,

spine nodes in a fragment $T_{spines}$

Represent routes that cross the cut interface annotating every cut edge
Cutting Down Fattrees

Cut fattree SRP $S$ into fragments

each pod $i$ in its own fragment $T_{pi}$,

spine nodes in a fragment $T_{spines}$

Represent routes that cross the cut interface annotating every cut edge
Cutting Down Fattrees

input nodes: assume the annotated route,

output nodes: guarantee (check) node converges to the annotated route
Cutting Down Fattrees

Interface defines

**input nodes: assume** the annotated route,

**output nodes: guarantee (check)** node converges to the annotated route
If we assume an annotation in one fragment, we guarantee it in another.
Catching Bugs with Modular Verification

Suppose we annotated our fattree network with this interface...
Imagine \( c_0 \) is reconfigured incorrectly, causing it to drop incoming routes (i.e., blackhole)…
Catching Bugs with Modular Verification

Given the assumptions on $c_0$’s input nodes, it will converge to the $\infty$ route.
Catching Bugs with Modular Verification

$c_0$’s $\infty$ route isn’t what the interface guaranteed, so verification fails, and the solver returns the $\infty$ route as a counterexample.
Catching Bugs with Modular Verification

We can now identify and fix the bug at $c_0$, without checking any other fragment!
A Theory of Network Fragments

Proven **sound**

if we cut SRP $S$ using interface $I$ into fragments $T_1, T_2, \ldots,$

if the fragments have solutions, their combined solutions are a solution to $S$.

Proven **complete**

if we cut SRP $S$ using interface $I$ into fragments $T_1, T_2, \ldots,$

if $I$ annotates the nodes with their solutions in $S$, the fragments have solutions.
Implementation

Kirigami

an extension to NV, a network modelling system & analysis tool
lets users define interfaces in the NV language for their networks

The “end-to-end” NV verification pipeline

preprocess network

if interface defined, cut into fragments using Kirigami

code network/fragments as SMT formulae

hand off encoding(s) to the Z3 SMT solver to check properties & guarantees
Evaluation

We wanted to find out...

Does Kirigami scale better than NV?
How do different cuts affect verification time?

Evaluated on a variety of benchmarks
Fattrees, random networks, wide-area networks
Simple shortest-path, valley-free routing, 1-node fault tolerance
Single-node reachability, all-ToR reachability

Kirigami improves maximum Z3 solve time by up to 100,000x, and end-to-end NV verification time by up to 10x.
All-ToR Reachability
Evaluation set-up

**k-fattree** topologies

- 20 (k=4) to 500 (k=20) nodes

Simple shortest-path Border Gateway Protocol (BGP) routing to a *symbolic* destination ToR node

4 different cuts considered for fattrees

- Finer cuts require *more annotations*, but should take *less time to solve*

Generated annotations using a script, using node tier and pod to determine shortest path to destination
Time taken by the **slowest SMT query** among all fragments (1 query/fragment)

**Smaller fragments ⇒ faster queries**

At 500 nodes, monolithic benchmark times out after 2h

... pods queries take at most 3.54s

... full query take at most 0.24s

SMT results are similar across other benchmarks
All-ToR Reachability
End-to-end Performance

Time taken by **NV verification pipeline**
Parallelized over 32 CPU cores, 128GB/core
Partitioned networks **scale past monolithic**
At 320 nodes, ~10x speedup
At 500 nodes, full cut spends 87% of time cutting network
Pods cut fully parallelizable, balances cutting time with solving time to achieve best overall time
Practical Kirigami Usage

Annotations are a small burden relative to writing the rest of the config

Users should annotate during development

Caveat: how difficult is it to come up with the correct annotations?

Easiest in a highly-structured network such as a data center

May need to cut more granularly to obtain interface with correct guarantees

Counterexamples can help refine interface if annotations don’t match network behavior
Limitations

Assumes networks converge to **unique solutions**

Uncommon in practice?

Easy to see for some protocols, *e.g.*, distance-vector protocols

Requires **exact annotations**

Stable routes ensure we don’t admit **spurious (incorrect) annotations**
Takeaways

Modularity has **critical benefits** for network verification

- Makes interactive behavior **explicit and easier to reason about**
- **Localizes** verification and error correction
- **Accelerates and parallelizes** analysis time

Kirigami brings modularity to network control plane verification

- ...with a **sound theoretical framework**
- ...and **proven benefits** on many topologies and policies!
## Comparison of Related Work

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Underlying technique</td>
<td>Encode BGP network to SMT</td>
<td>Encode network to SMT</td>
<td>Simulate policy over multi-layer graph</td>
<td>Use explicit-state model checking over policy model</td>
<td>Cut network, encode fragments to SMT</td>
</tr>
<tr>
<td>Arbitrary symbolic reasoning?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No*</td>
<td>Yes</td>
</tr>
<tr>
<td>Scales to large networks?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Modular?</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>


A Closer Look at the Implementation

```ocaml
include "fat.nv"

(* Associate each node with a fragment *)
let partition node = match node with
    | 0n | 1n | 2n | 3n -> 0 (* spines *)
    | 4n | 5n | 6n | 7n -> 1 (* pod 0 *)
    | 8n | 9n | 10n | 11n -> 2 (* pod 1 *)
    | 12n | 13n | 14n | 15n -> 3 (* pod 2 *)
    | 16n | 17n | 18n | 19n -> 4 (* pod 3 *)

(* Associate each edge with an annotation *)
let interface edge route = match edge with
    | 0~_ | 1~_ | 2~_ | 3~_ -> route = { id = d; cost = 2; }
    | 4~_ | 5~_ -> route = { id = d; cost = if d >= 4 && d <= 7 then 1 else 3; }
    | 8~_ | 9~_ -> route = { id = d; cost = if d >= 8 && d <= 9 then 1 else 3; }
    | 12~_ | 13~_ -> route = { id = d; cost = if d >= 12 && d <= 15 then 1 else 3; }
    | 16~_ | 17~_ -> route = { id = d; cost = if d >= 16 && d <= 19 then 1 else 3; }
```
Why Exact Annotations?

Limitation of our approach: couldn’t we overapproximate?

Exact routes ensure spurious annotations are not admitted

Other modular techniques also require a well-founded ordering

Different tradeoffs to provide this ordering