Formal Methods for Network Performance Analysis

Mina Tahmasbi Arashloo
University of Waterloo

Ryan Beckett
Microsoft

Rachit Agarwal
Cornell University
The shift towards automated formal analysis

| Stateful and programmable data plane verification | SymNet | VMN | p4v | NetSMC |
| Control plane verification | ERA | ARC | Bonsai | Origami | Tiramisu |
| Data plane verification | Batfish | Bagpipe | Minesweeper | FastPlane | Plankton |

<table>
<thead>
<tr>
<th>Atomic Predicates</th>
<th>NetPlumber</th>
<th>Veriflow</th>
<th>Atomic Predicates w/ Transformers</th>
<th>Symmetry &amp; Surgery</th>
<th>Delta-net</th>
<th>RCDC</th>
</tr>
</thead>
</table>

“Capturing the state of research on network verification”
Ryan Beckett and Ratul Mahajan, netverify.fun
The shift towards automated formal analysis

Create a mathematical model of the network

\[ \forall t \ (dstip(t) = A \land at(s_1, t)) \rightarrow at(s_2, t + 1) \]
\[ \forall t \ dstip(t) = B \land \forall s \ \neg at(s, t + 1) \]

...
The shift towards automated formal analysis

1. Create a mathematical model of the network

$$\forall t \ (dstip(t) = A \land at(s_1, t)) \rightarrow at(s_2, t + 1)$$
$$\forall t \ dstip(t) = B \land \forall s \ \neg at(s, t + 1)$$

2. Specify desired property

Does property $P$ always hold?
The shift towards automated formal analysis

1. Create a mathematical model of the network

\[
\forall t \ (dstip(t) = A \land at(s_1, t)) \rightarrow at(s_2, t + 1) \\
\forall t \ dstip(t) = B \land \forall s \ \neg at(s, t + 1)
\]

2. Specify desired property

Does property \( P \) always hold?

3. Automatically analyze the entire input space.

- Model checking
- Symbolic execution
- ...
The shift towards automated formal analysis

1. Create a mathematical model of the network
   \[
   \forall t \ (\text{dstip}(t) = A \land \text{at}(s_1, t)) \rightarrow \text{at}(s_2, t + 1)
   \]
   \[
   \forall t \ \text{dstip}(t) = B \land \forall s \ \neg\text{at}(s, t + 1)
   \]
   ...

2. Specify desired property
   Does property \( P \) always hold?

3. Automatically analyze the entire input space.
   - Model checking
   - Symbolic execution
   - ...

   e.g., packets entering the network
The shift towards automated formal analysis

1. Create a mathematical model of the network

\[ \forall t \ (\text{dst}(t) = A \land \text{at}(s_1, t)) \rightarrow \text{at}(s_2, t + 1) \]
\[ \forall t \ \text{dst}(t) = B \land \forall s \ \neg \text{at}(s, t + 1) \]
...

2. Specify desired property

Does property \( P \) always hold?

3. Automatically analyze the entire input space.

- Model checking
- Symbolic execution
- ...

4. Prove or disprove the property

✔ Property \( P \) always holds

❌ An example input for which \( P \) does not hold

e.g., packets entering the network
Existing work focuses on functional correctness

1. Create a mathematical model of the network

\[
\forall t \ (d_{\text{stip}}(t) = A \land a(t_{s1}, t) \rightarrow a(t_{s2}, t + 1)) \\
\forall t \ d_{\text{stip}}(t) = B \land \forall s \ \neg a(t, t + 1) \\
\ldots
\]

2. Specify desired property

Does property \( P \) always hold?

Lots of progress on analyzing \textit{functional correctness}
Existing work focuses on functional correctness

1. Create a mathematical model of the network

\[
\forall t \ (dstip(t) = A \land at(s_1, t) \rightarrow at(s_2, t + 1)) \\
\forall t \ src(t) = B \land \forall s \ \neg at(s, t + 1) \\
\ldots
\]

2. Specify desired property

Does property \( P \) always hold?

Lots of progress on analyzing functional correctness

- Is A reachable from B?
- Are there cyclic zone dependencies in DNS configurations?
- Is VLAN X traffic isolated from VLAN Y?
- ...

Existing work focuses on functional correctness

Lots of progress on analyzing functional correctness

- Is A reachable from B?
- Are there cyclic zone dependencies in DNS configurations?
- Is VLAN X traffic isolated from VLAN Y?
- ...

Does property \( P \) always hold?
Existing work focuses on functional correctness

1. Create a mathematical model of the network
   \[ \forall t \ (dstip(t) = A \land at(s_1, t)) \rightarrow at(s_2, t + 1) \]
   \[ \forall t \ dstip(t) = B \land \forall s \ \neg at(s, t + 1) \]
   ...

2. Specify desired property
   Does property \( P \) always hold?

Lots of progress on analyzing functional correctness
- Is A reachable from B?
- Are there cyclic zone dependencies in DNS configurations?
- Is VLAN X traffic isolated from VLAN Y?
- ...

But, what about performance?
Existing work focuses on functional correctness

1. Create a mathematical model of the network
   \[
   \forall t \ (dstip(t) = A \land at(s_1,t) \rightarrow at(s_2,t + 1))
   \forall t \ dstip(t) = B \land \forall s \ \neg at(s,t + 1)
   \ldots
   \]

2. Specify desired property
   Does property \( P \) always hold?

Lots of progress on analyzing **functional correctness**
- Is A reachable from B?
- Are there cyclic zone dependencies in DNS configurations?
- Is VLAN X traffic isolated from VLAN Y?
- ... 

But, what about **performance**?
- Can flow A's throughput drop below R?
- Can packets in traffic class B experience latency > L?
- Can flow X get a much larger share of the bandwidth than Y?
- ... 

Existing work focuses on functional correctness

- Lots of progress on analyzing **functional correctness**
  - Is A reachable from B?
  - Are there cyclic zone dependencies in DNS configurations?
  - Is VLAN X traffic isolated from VLAN Y?
  - ...

But, what about **performance**?
- Can flow A’s throughput drop below R?
- Can packets in traffic class B experience latency > L?
- Can flow X get a much larger share of the bandwidth than Y?
- ...
This work:
Using formal methods to analyze performance properties
Finding the “right” model

1. Create a mathematical model of the network

2. Specify desired property
   Does property $P$ always hold?

3. Automatically analyze the entire input space.
   - Model checking
   - Symbolic execution
   - …

4. Prove or disprove the property
   - ✔ Property $P$ always holds
   - ❌ An example input for which $P$ does not hold
Finding the “right” model

1. Create a mathematical model of the network

2. Specify desired property

Does property $P$ always hold?

Extensively explored for packet forwarding.

A Switch

<table>
<thead>
<tr>
<th>dstip</th>
<th>port</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>$P_1$</td>
</tr>
<tr>
<td>$X_2$</td>
<td>$P_2$</td>
</tr>
<tr>
<td>$X_3$</td>
<td>$P_3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Finding the “right” model

1. Create a mathematical model of the network

   ??

2. Specify desired property

   Does property $P$ always hold?

For performance analysis, we need more than just forwarding.
Finding the “right” model

1. Create a mathematical model of the network

2. Specify desired property

Does property $P$ always hold?

For performance analysis, we need more than just forwarding.
Finding the “right” model

1. Create a mathematical model of the network

2. Specify desired property

   Does property $P$ always hold?

For performance analysis, we need more than just forwarding

![Diagram of a switch with packet sequences and queues]
Finding the “right” model

1. Create a mathematical model of the network

2. Specify desired property

For performance analysis, we need more than just forwarding

- Model checking
- Symbolic execution
- ...

Prove or disprove the property

- ✔ Property always holds
- ❌ An example input for which does not hold

Timed packet sequences

A Switch

<table>
<thead>
<tr>
<th>dstip</th>
<th>port</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>P₁</td>
</tr>
<tr>
<td>X₂</td>
<td>P₂</td>
</tr>
<tr>
<td>X₃</td>
<td>P₃</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Queues

For performance analysis, we need more than just forwarding
Our model: Composition of “queuing modules”

Queuing Module
(Building block)

Processing
- Pull from input queues
- Process
- Push to output queues

N input queues

M output queues
Our model: Composition of “queuing modules”
Our model: Composition of “queuing modules”
Our model: Composition of “queuing modules”

Queues are modeled explicitly:

$q.elem[i][t] \rightarrow i$-th packet in the queue at time $t$
Our model: Composition of “queuing modules”

How do we make it tractable to analyze?

- **Abstract** time over dequeues
- **Bounded** time analysis
- **Efficient** queue encoding
- **Optimizing** compositions
Finding the “right” model

1. Create a mathematical model of the network
   - Composition of queuing modules

2. Specify desired property
   - Does property $P$ always hold?

3. Automatically analyze the entire input space.
   - Model checking
   - Symbolic execution
   - ...

4. Prove or disprove the property
   - ✔ Property $P$ always holds
   - ❌ An example input for which $P$ does not hold
Specifying performance properties of interest

1. Create a mathematical model of the network
   - Composition of queuing modules

2. Specify desired property
   - ??

3. Automatically analyze the entire input space.
   - Model checking
   - Symbolic execution
   - ...

4. Prove or disprove the property
   - ✔ Property $P$ always holds
   - ❌ An example input for which $P$ does not hold
Specifying performance properties of interest

1. Create a mathematical model of the network

   Composition of queuing modules

2. Specify desired property

   ??

- Pre- or user-defined metrics over queues
  - Queue size: $\text{queue\_size}(q, t)$
  - Number of enqueued packets: $\text{total\_packets}(q, t)$
  - Arrival inter-packet gap: $\text{inter\_packet\_gap}(q, t)$
  - $<$insert your metric of interest$>$

Automatically analyze the entire input space.

- Model checking
- Symbolic execution
- …

- Prove or disprove the property

  ✔ Property $P$ always holds
  ❌ An example input for which $P$ does not hold
Specifying performance properties of interest

1. Create a mathematical model of the network
2. Specify desired property

- Composition of queuing modules

- Properties compare metrics to certain values

??
Specifying performance properties of interest

1. Create a mathematical model of the network

   Composition of queuing modules

2. Specify desired property

   ??

- Properties compare metrics to certain values

- Model checking
- Symbolic execution
- …

- Prove or disprove the property

  ✔ Property always holds

  ❌ An example input for which does not hold

Composition of queuing modules
Specifying performance properties of interest

1. Create a mathematical model of the network
   - Composition of queuing modules

2. Specify desired property
   - ??

- Properties compare metrics to certain values
  - \( \text{inter\_packet\_gap}(q_1, t_1) \geq 10 \)
Specifying performance properties of interest

1. Create a mathematical model of the network
   - Composition of queuing modules

2. Specify desired property
   - ??

- Properties compare metrics to certain values
  - \( \text{inter\_packet\_gap}(q_1, t_1) \geq 10 \)
  - \( \text{queue\_size}(q_1, t_5) \leq \text{queue\_size}(q_2, t_6) \)
Specifying performance properties of interest

1. Create a mathematical model of the network

Composition of queuing modules

2. Specify desired property

??

• Properties compare metrics to certain values
  - \(\text{inter\_packet\_gap}(q_1, t_1) \geq 10\)
  - \(\text{queue\_size}(q_1, t_5) \leq \text{queue\_size}(q_2, t_6)\)
  - \(\sum_{q \in \{u_1, \ldots, u_k\}} \text{total\_packets}(q, t_{10}) \geq 20\)
Specifying performance properties of interest

1. Create a mathematical model of the network
   - Composition of queuing modules

2. Specify desired property
   - Property $P$: $\text{queue\_size}(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.
   - Model checking
   - Symbolic execution
   - ...

4. Prove or disprove the property
   - ✓ Property $P$ always holds
   - ❌ An example input for which $P$ does not hold
Analyzing \( model \land \neg property \)

1. Create a mathematical model of the network
   - Composition of queuing modules

2. Specify desired property
   - Property \( P: \text{queue\_size}(q_1, t_1) \leq 10 \)

3. Automatically analyze the entire input space.

4. Prove or disprove the property
   - ✔ Property \( P \) always holds
   - ❌ An example input for which \( P \) does not hold
Analyzing $model \land \neg property$

2. Specify desired property

Property $P$: $queue\_size(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.

4. Prove or disprove the property

- ✔ Property $P$ always holds
- ❌ An example input for which $P$ does not hold

- $model \land \neg property$ is a quantifier-free SMT formula with integer arithmetic
- We use Z3 to analyze its satisfiability.
Analyzing $model \land \lnot property$

1. Create a mathematical model of the network
   Composition of queuing modules

2. Specify desired property
   Property $P$: $\text{queue\_size}(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.
   Bounded Model checking with Z3

4. Prove or disprove the property
   ✔ Property $P$ always holds
   ❌ An example input for which $P$ does not hold
When the property doesn’t hold...

1. Create a mathematical model of the network
   Composition of queuing modules

2. Specify desired property
   Property $P$: $\text{queue\_size}(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.
   Bounded Model checking with Z3

4. Prove or disprove the property
   - ✔ Property $P$ always holds
   - ❌ An example input for which $P$ does not hold
When the property doesn’t hold…

1. Create a mathematical model of the network
   Composition of queuing modules

2. Specify desired property
   Property $P$: \( \text{queue\_size}(q_1, t_1) \leq 10 \)

3. Automatically analyze the entire input space.
   Bounded Model checking with Z3

4. Prove or disprove the property
   ✔ Property $P$ always holds
   ❌ An example input for which $P$ does not hold
A single trace is not an informative output
A single trace is not an informative output
A single trace is not an informative output

Property: F3 should not be blocked for dequeue (get starved) for $X$ consecutive time steps.
A single trace is not an informative output

Property: F3 should not be blocked for dequeue (get starved) for $X$ consecutive time steps.

Output: Does not hold.
A single trace is not an informative output

Property: F3 should not be blocked for dequeue (get starved) for $X$ consecutive time steps.

Output: Does not hold. e.g., for this particular input:
A single trace is not an informative output

- Timed packet sequences
  - Needed in the model
  - Not necessarily useful in the output

Property: \( F_3 \) should not be blocked for dequeue (get starved) for consecutive time steps.

Output: Does not hold.

E.g., for this particular input:
A single trace is not an informative output

- Not all details matter with respect to the property
A single trace is not an informative output

- Not all details matter with respect to the property

For this particular input:

- Property: F3 should not be blocked for dequeue (get starved) for consecutive time steps.

Diagram:

- A single trace is not an informative output
- Not all details matter with respect to the property
- A Priority Scheduler
  - F1
  - F2
  - F3
  - F4
- Time:
  - t₁
  - t₂
  - t₃
  - t₄
  - t₅
A single trace is not an informative output

- Not all details matter with respect to the property
- Unclear if it points to an “important” problem
- Note the contrast to functional correctness properties

Property: $F_3$ should not be blocked for dequeue (get starved) for consecutive time steps.
A single trace is not an informative output

Property: F3 should not be blocked for dequeue (get starved) for X consecutive time steps.

Output: Does not hold. e.g., for this particular input:
Alternative? Conditions on the input

Property: F3 should not be blocked for dequeue (get starved) for X consecutive time steps.

Output: Does not hold.

e.g., for this particular input:
e.g., for these set of conditions on the input:
Alternative? Conditions on the input

Property: F3 should not be blocked for dequeue (get starved) for $X$ consecutive time steps.

Output: Does not hold.

- e.g., for this particular input:
- e.g., for these set of conditions on the input:

- F1 or F2 have packets for $X$ consecutive time steps
- F3 has at least a packet

A Priority Scheduler
Alternative? Conditions on the input

- **Workload**: Conjunction of constraints on the input

  - F1 or F2 have packets for $X$ consecutive time steps
  - F3 has at least a packet

Output: Does not hold.

e.g., for these set of conditions on the input:

- F1
- F2
- F3
- F4

Workload:

Conjunction of constraints on the input
Alternative? Conditions on the input

- **Workload:** Conjunction of constraints on the input

\[
\forall t \in [1,X] \sum_{q \in \{F_1, F_2\}} \text{total_packets}(q, t) \geq t \\
\wedge \forall t \in [1,X] \text{total_packets}(F_3, t) \geq 1
\]

E.g., for these set of conditions on the input:

- **F1** or **F2** have packets for \( X \) consecutive time steps
- **F3** has at least a packet
**Alternative? Conditions on the input**

- **Workload**: Conjunction of constraints on the input
  - (Concisely) represents a set of traces
    - More informative
    - Indicative of a more prominent problem.

  \[ \forall t \in [1, X] \sum_{q \in \{F_1, F_2\}} \text{total_packets}(q, t) \geq t \]
  \[ \land \forall t \in [1, X] \text{total_packets}(F_3, t) \geq 1 \]

  **e.g., for these set of conditions on the input:**

  - F1 or F2 have packets for \( X \) consecutive time steps
  - F3 has at least a packet
Synthesizing workloads

1. Create a mathematical model of the network

   Composition of queuing modules

2. Specify desired property

   Property $P$: $\text{queue}\_\text{size}(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.

   Bounded Model checking with Z3

4. Prove or disprove the property

   - ✔ Property $P$ always holds
   - ❌ An example input for which $P$ does not hold
Synthesizing workloads

1. Create a mathematical model of the network
   Composition of queuing modules

2. Specify desired property
   Property $P$: $\text{queue\_size}(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.
   Bounded Model checking with Z3

4. Prove or disprove the property
   - Property $P$ always holds
   - A workload $wl$ such that $P$ does not hold for any trace in $wl$
Synthesizing workloads

1. Create a mathematical model of the network
   - Composition of queuing modules

2. Specify desired property
   - Property $P$: $\text{queue\_size}(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.

4. Prove or disprove the property
   - ✔ Property $P$ always holds
   - ✗ A workload $wl$ such that $P$ does not hold for any trace in $wl$
Synthesizing workloads

1. Create a mathematical model of the network.
   Composition of queuing modules

2. Specify desired property
   Property $P$: \( \text{queue\_size}(q_1, t_1) \leq 10 \)

3. Automatically analyze the entire input space.

4. Prove or disprove the property
   - ✔ Property $P$ always holds
   - ❌ A workload $wl$ such that $P$ does not hold for any trace in $wl$
Synthesizing workloads

1. Create a mathematical model of the network
   Composition of queuing modules

2. Specify desired property
   Property $P$: $\text{queue\_size}(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.
   Workload Search

4. Prove or disprove the property
   ✔ Property $P$ always holds
   ❌ A workload $wl$ such that $P$ does not hold for any trace in $wl$
Synthesizing workloads

1. Create a mathematical model of the network

2. Specify desired property
   
   Property $P$: $\text{queue\_size}(q_1,t_1) \leq 10$

3. Automatically analyze the entire input space.

4. Prove or disprove the property
   
   ✔ Property $P$ always holds
   
   ✗ A workload $wl$ such that $P$ does not hold for any trace in $wl$

Syntax-Guided Synthesis

Composition of queuing modules

Workload Search

Bounded Model checking with Z3
Synthesizing workloads

1. Create a mathematical model of the network components.

2. Specify desired property.
   
   Property $P$: $\text{queue\_size}(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.

4. Prove or disprove the property.
   
   - ✔ Property $P$ always holds
   - ❌ A workload $wl$ such that $P$ does not hold for any trace in $wl$
FPerf: Formal Performance Analyzer

1. Create a mathematical model of the network
   Composition of queuing modules

2. Specify desired property
   Property $P$: $\text{queue\_size}(q_1, t_1) \leq 10$

3. Automatically analyze the entire input space.
   - Workload Search
   - Bounded Model checking with Z3

4. Prove or disprove the property
   - ✔ Property $P$ always holds
   - ✗ A workload $wl$ such that $P$ does not hold for any trace in $wl$
See the paper for

- Details of the search algorithm
  - Randomized search
  - Guided by a cost function over workloads
- Generating example traces for the search cost function
- Optimizations for the search and verification process
- Constraining the input search space to the user’s interest
- ...
## Case study - Packet scheduling

<table>
<thead>
<tr>
<th>Property</th>
<th>Priority</th>
<th>Round-Robin</th>
<th>FQ in FQ-CoDel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starvation</td>
<td>Fairness</td>
<td>Fairness</td>
<td></td>
</tr>
</tbody>
</table>
## Case study - Packet scheduling

<table>
<thead>
<tr>
<th>Property</th>
<th>Priority</th>
<th>Round-Robin</th>
<th>FQ in FQ-CoDel</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starvation</td>
<td></td>
<td>Fairness</td>
<td>Fairness</td>
<td>Starvation + Fairness</td>
</tr>
</tbody>
</table>
Case study - Packet scheduling

- 11 queuing modules
- Host + NIC scheduling
- Inspired from Loom (NSDI’19)
## Case study - Packet scheduling

<table>
<thead>
<tr>
<th>Property</th>
<th>Priority</th>
<th>Round-Robin</th>
<th>FQ in FQ-CoDel</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starvation</td>
<td></td>
<td>Fairness</td>
<td>Fairness</td>
<td>Starvation + Fairness</td>
</tr>
</tbody>
</table>
# Case study - Packet scheduling

10s of thousands variables and constraints

<table>
<thead>
<tr>
<th>Property</th>
<th>Priority</th>
<th>Round-Robin</th>
<th>FQ in FQ-CoDel</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># variables</td>
<td>1.5K</td>
<td>2.6K</td>
<td>4.5K</td>
<td>17.9K</td>
</tr>
<tr>
<td># constraints</td>
<td>7K</td>
<td>13K</td>
<td>21K</td>
<td>94K</td>
</tr>
</tbody>
</table>
Case study - Packet scheduling

Search time is reasonable
Example generation is a bottleneck

<table>
<thead>
<tr>
<th>Property</th>
<th>Priority</th>
<th>Round-Robin</th>
<th>FQ in FQ-CoDel</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Starvation</td>
<td>Fairness</td>
<td>Fairness</td>
<td>Starvation + Fairness</td>
</tr>
<tr>
<td>Model Size</td>
<td># variables</td>
<td>1.5K</td>
<td>2.6K</td>
<td>4.5K</td>
</tr>
<tr>
<td></td>
<td># constraints</td>
<td>7K</td>
<td>13K</td>
<td>21K</td>
</tr>
<tr>
<td>The Search Algorithm</td>
<td># rounds</td>
<td>65</td>
<td>268</td>
<td>769</td>
</tr>
<tr>
<td></td>
<td>time (sec.)</td>
<td>3</td>
<td>59</td>
<td>223</td>
</tr>
</tbody>
</table>
Case study - Packet scheduling

<table>
<thead>
<tr>
<th>Property</th>
<th>Priority</th>
<th>Round-Robin</th>
<th>FQ in FQ-CoDel</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Starvation</td>
<td>Fairness</td>
<td>Fairness</td>
<td>Starvation + Fairness</td>
</tr>
<tr>
<td>Model Size</td>
<td>1.5K</td>
<td>2.6K</td>
<td>4.5K</td>
<td>17.9K</td>
</tr>
<tr>
<td></td>
<td>7K</td>
<td>13K</td>
<td>21K</td>
<td>94K</td>
</tr>
<tr>
<td>The Search Algorithm</td>
<td>65</td>
<td>268</td>
<td>769</td>
<td>361</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>59</td>
<td>223</td>
<td>461</td>
</tr>
<tr>
<td>Verifying Candidate Workloads (avg) (sec.)</td>
<td>0.03</td>
<td>0.04</td>
<td>0.10</td>
<td>0.81</td>
</tr>
</tbody>
</table>

Workload verification (and model analysis) is efficient!
## Case study - Packet scheduling

It is possible to synthesize workloads in a few minutes

<table>
<thead>
<tr>
<th>Property</th>
<th>Priority</th>
<th>Round-Robin</th>
<th>FQ in FQ-CoDel</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Starvation</td>
<td>Fairness</td>
<td>Fairness</td>
<td>Starvation + Fairness</td>
</tr>
<tr>
<td>Model Size</td>
<td># variables</td>
<td>1.5K</td>
<td>2.6K</td>
<td>4.5K</td>
</tr>
<tr>
<td></td>
<td># constraints</td>
<td>7K</td>
<td>13K</td>
<td>21K</td>
</tr>
<tr>
<td>The Search Algorithm</td>
<td># rounds</td>
<td>65</td>
<td>268</td>
<td>769</td>
</tr>
<tr>
<td></td>
<td>time (sec.)</td>
<td>3</td>
<td>59</td>
<td>223</td>
</tr>
<tr>
<td>Verifying Candidate Workloads (avg) (sec.)</td>
<td>0.03</td>
<td>0.04</td>
<td>0.10</td>
<td>0.81</td>
</tr>
<tr>
<td>Total Time (min.)</td>
<td>0.2</td>
<td>6.2</td>
<td>9.6</td>
<td>18.5</td>
</tr>
</tbody>
</table>
Case study - A (small) leaf-spine network

• Modeled with ~23 queuing modules with 66 queues
  • ~twice larger than the packet composition case study
• Asked about properties related to throughput and latency
• Observed similar trends
Case study - A (small) leaf-spine network

- The trend is (unsurprisingly) exponential
- Modular analysis will be crucial for scale
Concluding remarks

• **Our goal:** Exploring the transition from reasoning about functional correctness to **performance properties**

• **Our findings:** Intriguing implications on modeling and analysis techniques.
  • e.g., *workloads* as opposed to individual counter examples

• We are excited about the possibilities ahead!
  • FPerf’s code is available on GitHub: [https://github.com/minmit/fperf](https://github.com/minmit/fperf)
  • And we are actively looking for more use cases to improve