A Formally Verified NAT

Background

- Middleboxes are important and everywhere.

- But middleboxes are buggy.

- The network becomes vulnerable due to middlebox crashes.

What does the paper do?

- Formally verifying the functional correctness of a network function (middlebox).
- Verify a NAT (Network Address Translation) middlebox following RFC 3022 standard.
- Propose a framework, called Vigor, for combining symbolic execution with theorem prover.

Choose Verification Technique

- Symbolic execution
  - KLEE: http://klee.github.io/, from an OSDI 2008 paper
- Theorem proving
  - Coq: https://coq.inria.fr/
  - Agda, Z3, F-star

Symbolic Execution

- Human efforts: fully automated, low.
- Machine efforts: exhaustive search, super high.
- An assertion:
  - It is impossible for symbolic execution to verify complicated programs, as it takes forever.

Theorem proving

- Human effort: high, I can show you my personal experience.
  - 19 lines of implementation.
  - 304 lines of proof.
- Machine effort: I would say moderate, a complicated proof usually requires hours to run.
  - Better than never stop.

Find a balance

- Combine the best of symbolic execution and theorem proving.

Vigor

	Bits	Algebras
Human effort	**low**	high / infeasible
Machine effort	high / infeasible	**low**

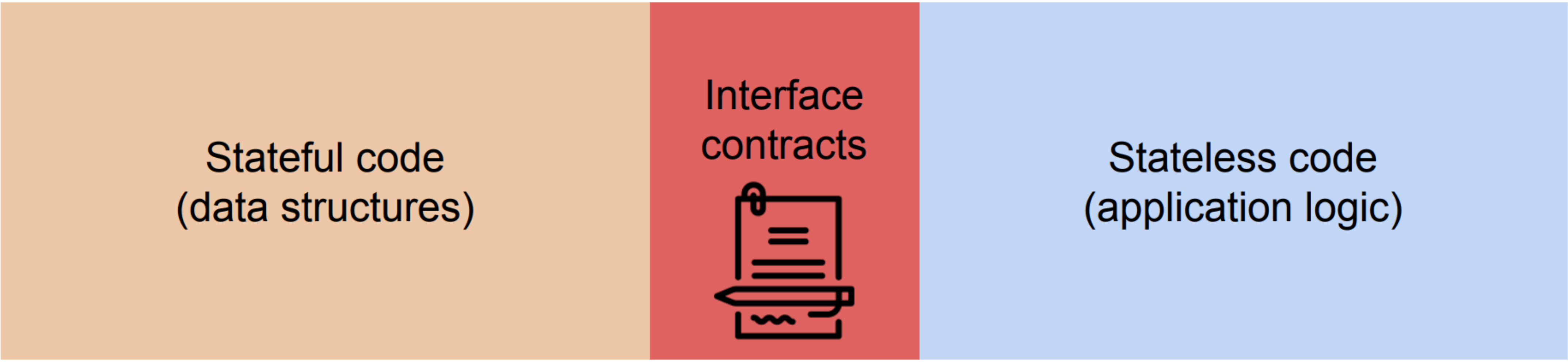
Vigor

- Use symbolic execution to check stateless part.

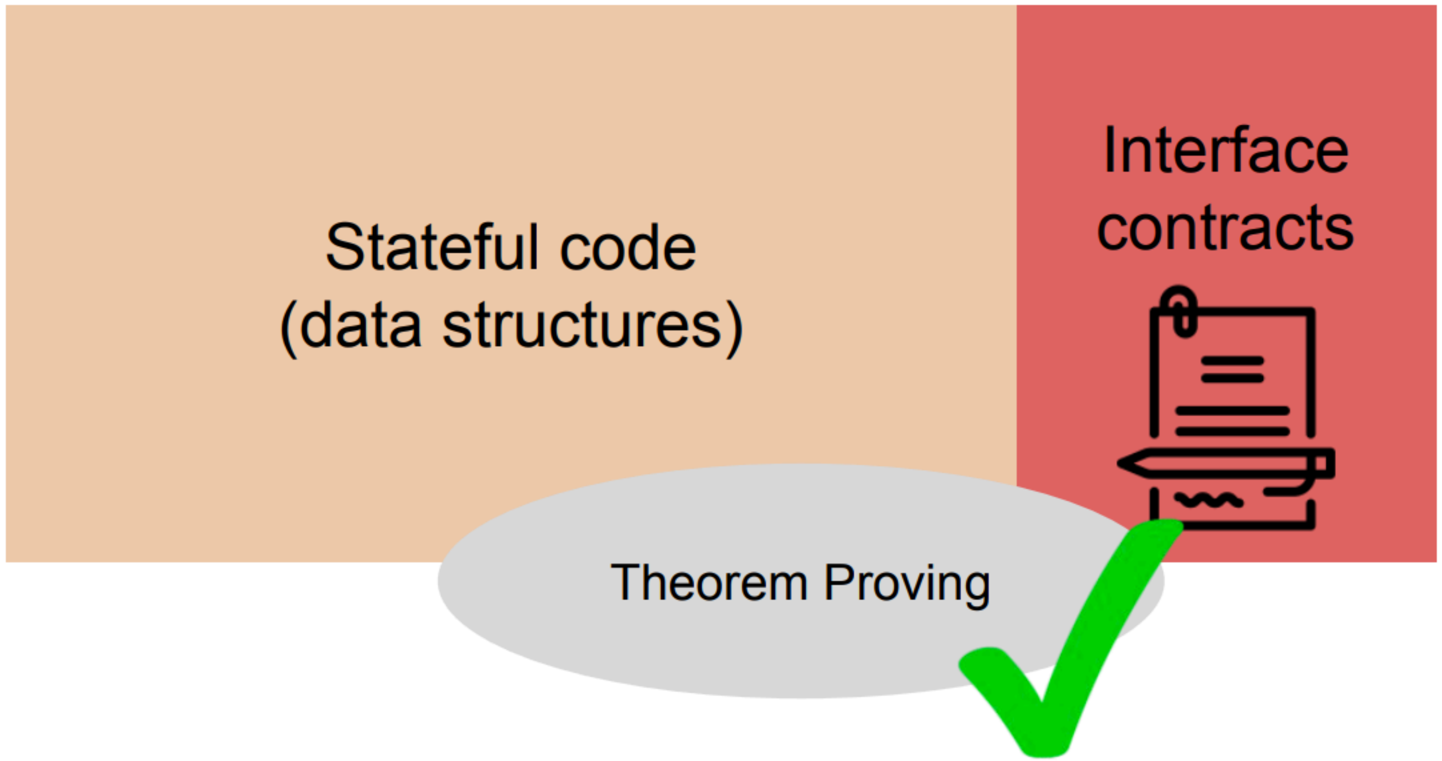
- Use theorem proving to check stateful data structures.

- Combine them together into a full-program verification.

Vigor


Stateful code
(data structures)
Interface
contracts
Stateless code
(application logic)

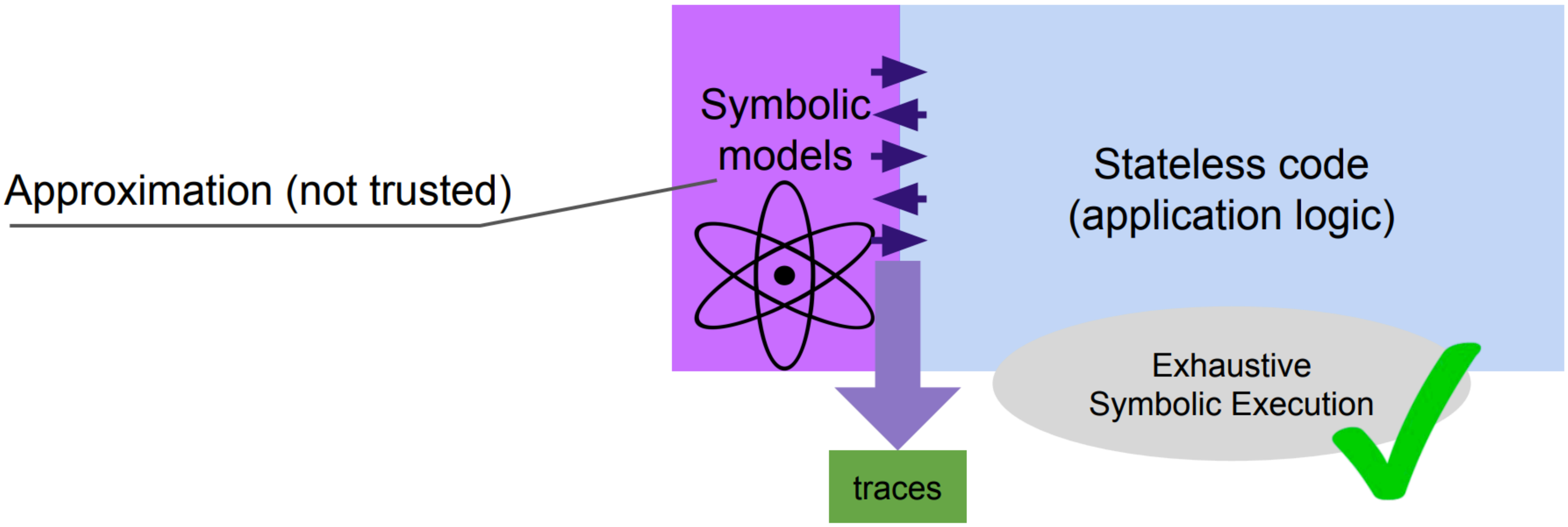
Theorem proving



Theorem proving

- They use Z3 theorem prover.
- C code with annotations.
- Annotations:
  - Pre-condition: what you expect from the function input.
  - Post-condition: what you expect when the function finishes.
  - Function body annotations: for auxiliary

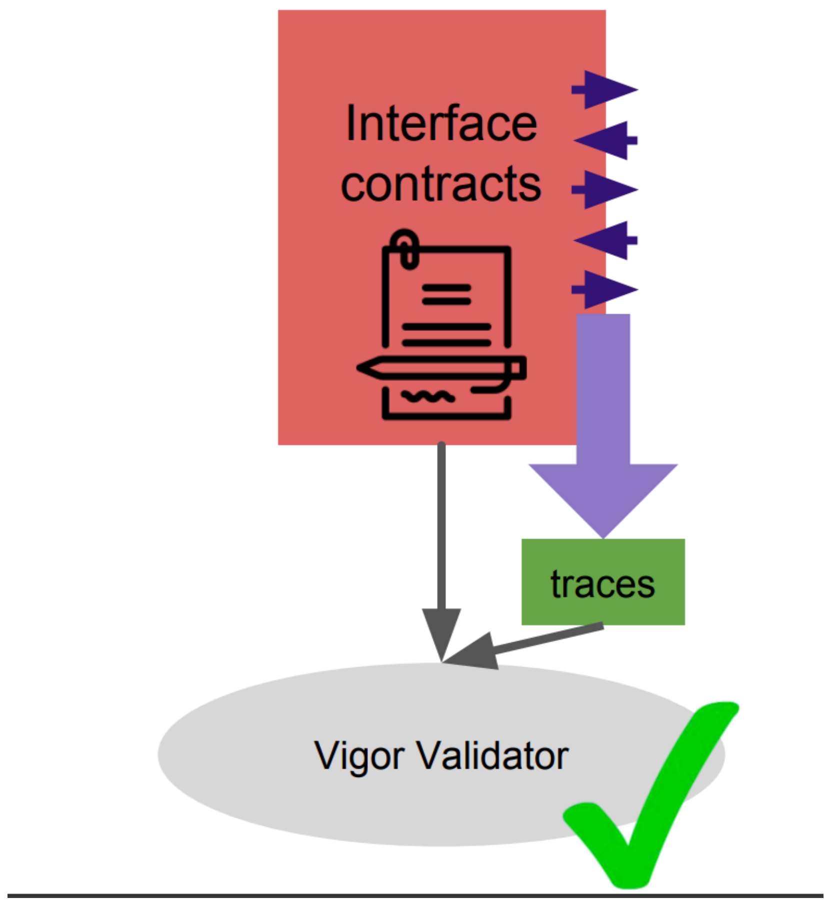
Symbolic execution



Symbolic Execution

- They use KLEE symbolic execution engine.
- Must avoid state explosion.
  - Replace stateful code with symbolic modules.
- Add another source to verify.

Verify the Correctness of Symbolic Modules

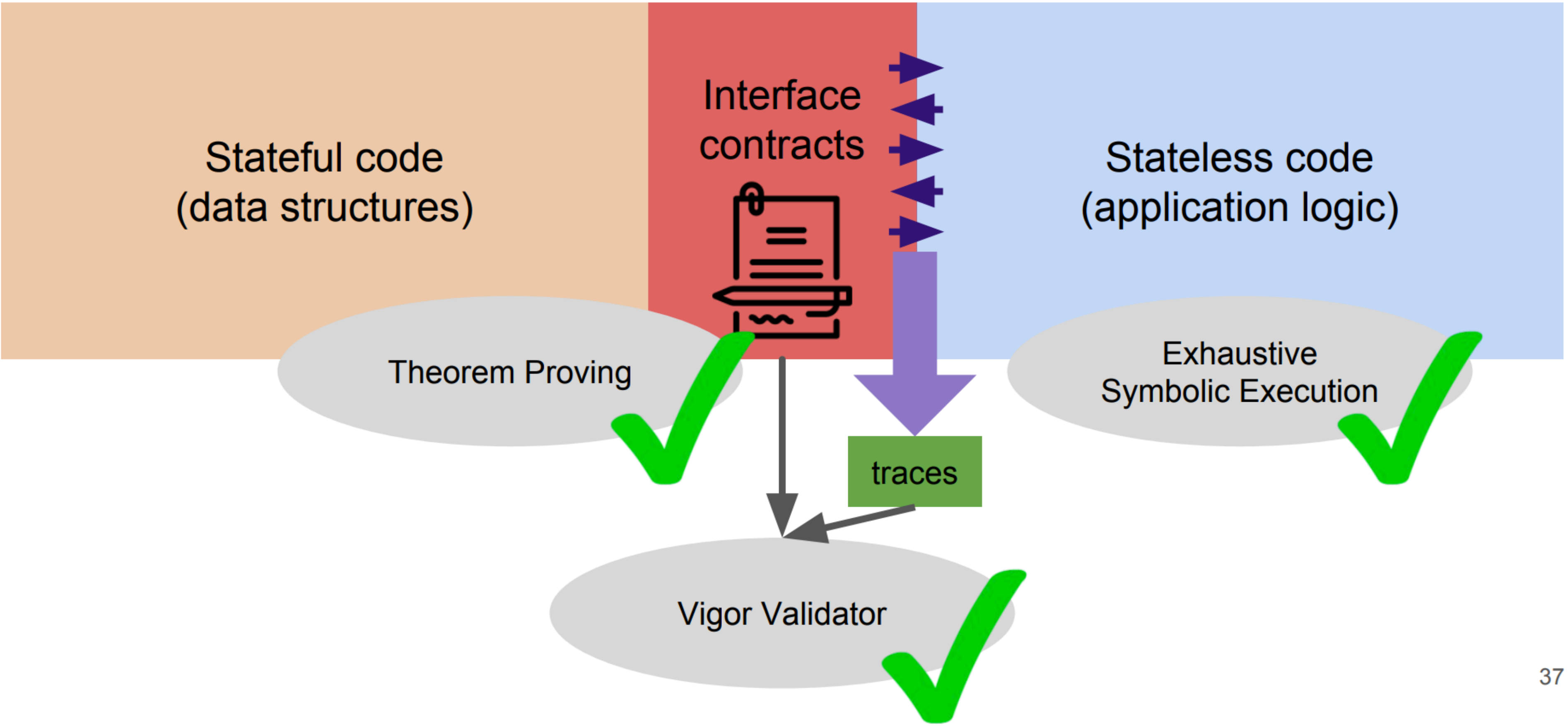


Verify the Correctness of Symbolic Modules

- They design a validator written in OCaml.

- Verify the traces collected during symbolic execution, against the interface contract they designed when verifying the stateful code.

Vigor



Stateful code
(data structures)
Interface
contracts
Stateless code
(application logic)
Theorem Proving
Exhaustive
Symbolic Execution
traces
Vigor Validator

Evaluation

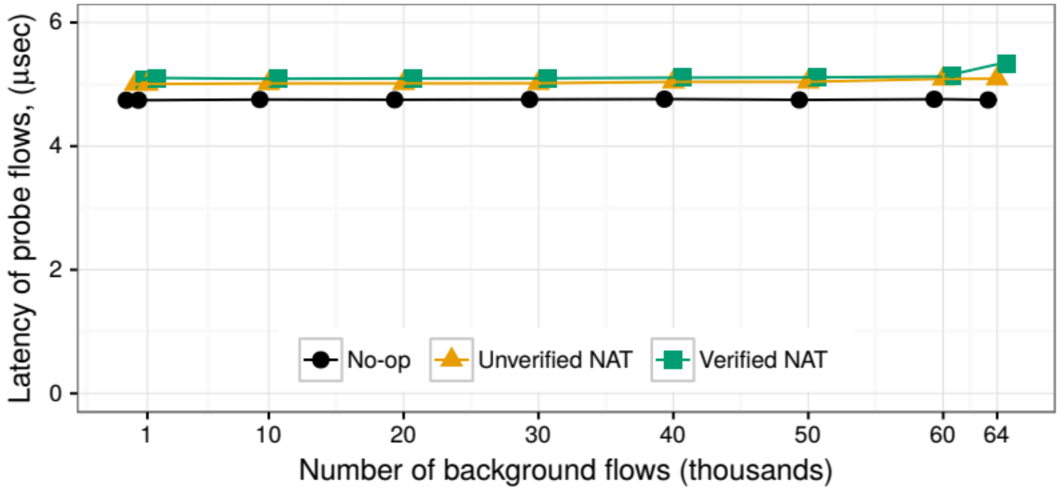


Figure 12: Average latency for probe flows. Confidence intervals are approximately 20 nanosec, not visible at this scale.

Evaluation

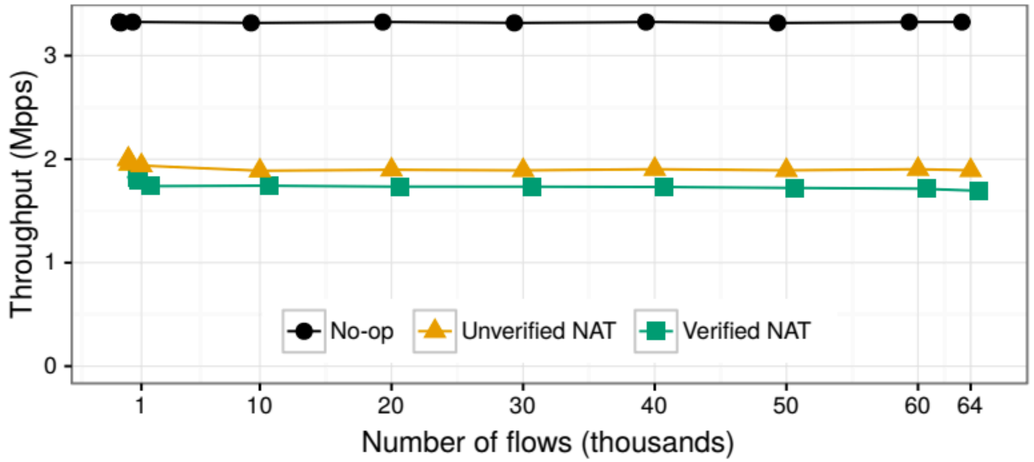


Figure 14: Maximum throughput with a maximum loss rate of 0.1%.

Discussion

- Vigor’s approach is new, but:
    - Ad-hoc.
    - Broke code modularity.
    - Can not scale to more complicated NFs, like Snort IDS.

- I would prefer a systematic programming framework, where you can do everything within this framework, not stitch things together.

Discussion

- Theorem provers are more powerful than before.
- “**machine-checked mathematical proofs are the next big enabler of practical program structuring techniques**” – Adam Chlipala
- Proof automation:
  - You can provide some hints, the prover figures out the rest of the proof.
  - SMT solver

F-star Language

- Developed by Microsoft and INRIA.
- Support most of the modern theorem proving features.
- A subset of F-star can be compiled to C for efficiency.

Full Stack NFV Verification in F-star

- User space packet I/O framework (like netmap/dpdk).
- User space TCP/IP stack (like mTCP).
- Verifying complicated NF (like Snort IDS).

Full Stack NFV Verification in F-star

- Huge correctness guarantee.

- Still good performance:
  - Verify in F-star.
  - Extract to efficient C code.
  - Compile with modern C compilers for speed.

- Good research opportunities.