Policy Anomaly Detection for Distributed IPv6 Firewalls

Claas Lorenz and Bettina Schnor

Potsdam University
Institute for Computer Science
Operating Systems and Distributed Systems

Colmar, 2015-07-20

Outline

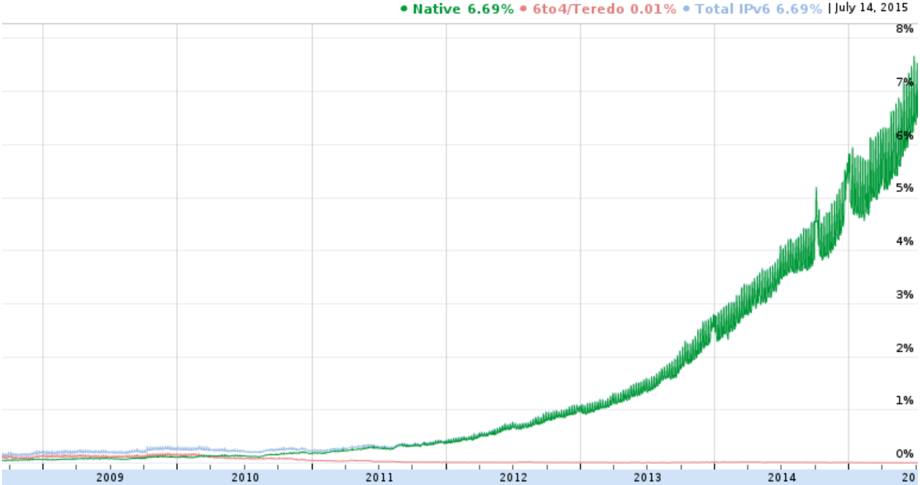


Figure: Share of IPv6 requests for Google services[1].

[1] cf. [Goo15]

Motivation

- Why IPv6?
  - Steady growth of the IPv6 share
  - Internet of Things works only with enough addresses (estimated: 100 trillion entities[2])
- Often historically grown firewall policies
- Manual migration from IPv4 to IPv6 is challenging for large firewall instances
- IDSv6 project[3]:
  - ft6: Tests firewalls for RFC conformity
  - **ad6**: Finds policy anomalies firewall and network configurations

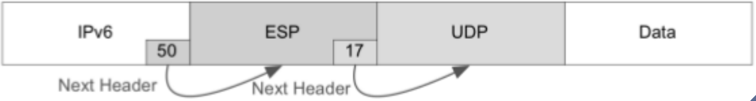
[2] cf. [HFD12] p. 13
[3] cf. [IDS13]

Challenges for Formal Verification

- Goal is the detection of the anomalies:
  - cyclicity

```
ip6tables -P OUTPUT DROP
ip6tables -A OUTPUT -p udp -j ACCEPT
ip6tables -A OUTPUT -p tcp -j OUTPUT
```

  - unreachability
  - shadowing
  - cross-path
- Extension of the formalism of Jeffrey and Samak[4]
- Larger base header with IPv6 (320 vs. 104 bits) $\rightarrow$ enlarges search space
- Extension header chains of arbitrary length[5]

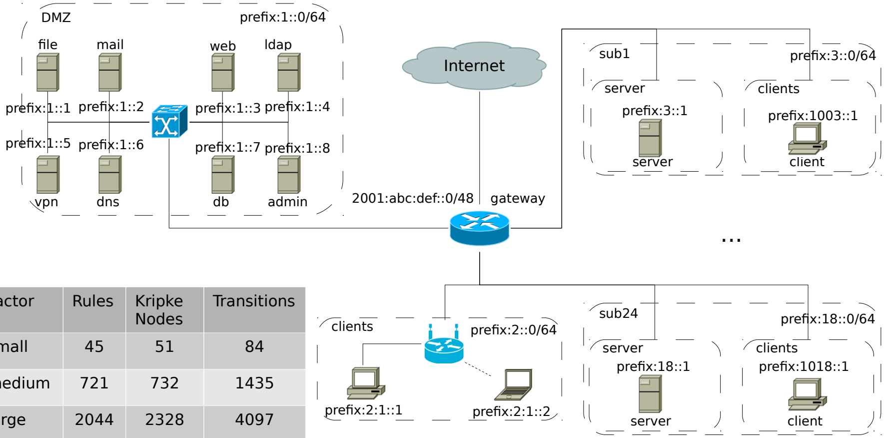


[4] cf. [JS09]

[5] Illustration from [BE06]

Formal Integration - Summary

- Encoding of the firewall rules and the network as Kripke Structure
- Transformation to SAT
- Problem encodings for two additional anomalies: shadowing, cross-path
- For details please refer our paper



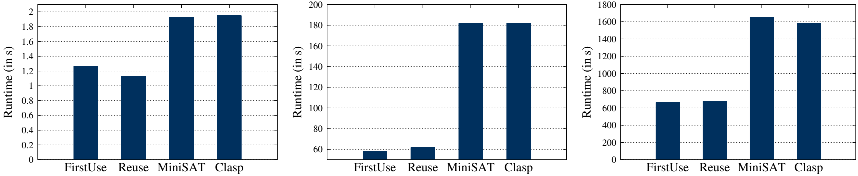
Factor	Rules	Kripke Nodes	Transitions
small	45	51	84
medium	721	732	1435
large	2044	2328	4097

Runtime Measurements - Methodology

- Quantification and scalability estimation
- Synthetic network and firewall topology
- Inspired by our campus' network
- No anomalies inherited ($\rightarrow$ worst-case runtime anticipated)
- Processing was single threaded
- Two independent phases with two parameters each:
  - Building phase: First Use vs. Reuse
  - Solving phase: MiniSAT vs. Clasp
- Measurement environment:

CPU	Intel i7-3630QM	Cores	4
Freq	2.4GHz	RAM	8GB
OS	Arch Linux	Kernel	v3.16.2
Python	v3.4.1	Redis	v2.8.17
MiniSAT	v2.2.0	Clasp	v3.0.3

Runtime Measurements - Results



(a) small (b) medium (c) large

- Runtime is superlinear but subquadratic
- $f(x) = ax^2 + bx + c$ with
  - x is the number of rules
  - $a \approx 0.0002$, $b \approx -0.06$ and $c \approx 3.54$ (Building, First Use)
  - $a \approx 0.0004$, $b \approx -0.04$ and $c \approx 2.85$ (Solving, Clasp)
- Longest total runtime of ~37,2 minutes
- Memory usage was linear

Conclusion

- Extensions of the formalism of Jeffrey and Samak for:
  - Shadowing and cross-path detection
  - IPv6 base headers
  - Extension header chains
- Runtime does not behave exponential but low quadratic

  $\rightarrow$ acceptable for the migration scenario

Future Work

- Performance improvement by:
  - Native interfaces for the solver
  - Parallelization
  - Learning from intermediate results
- Expressiveness: Stateful firewalls, VPN-tunnels, etc.
- Applicability for SDN?

Thank you for your attention!

Literature I

[BE06] Philippe Biondi and Arnaud Ebalard.
Scapy and IPv6 networking.
Talking Slides from
http://www.secdev.org/conf/scapy-IPv6_HITB06.pdf, 2006.

[Goo15] Google IPv6 - Statistics.
https://www.google.com/intl/en/ipv6/statistics.html, 2015.

[HFD12] Kai Hwang, Geoffrey C. Fox, and Jack J. Dongarra.
Distributed and Cloud Computing - From Parallel Processing to the Internet of Things.
Elsevier, 2012.

[IDS13] IPv6 Intrusion Detection System - Attack prevention and validated protection of IPv6 networks.
http://www.idsv6.de/de/index.html, 2013.

Literature II

[JS09] Alan Jeffrey and Taghrid Samak.
Model Checking Firewall Policy Configurations.
In *POLICY*, pages 60–67. IEEE Computer Society, 2009.