Facilitating The Internet Of Things With Policy Programming

Supplementing the works presented in:

"Integrated development environment for debugging policy-based applications in wireless sensor networks"

...and lessons learned since.

Daniel Smullen

Nidal Qwasmi

Ramiro Liscano

Funding provided by:

NSERC CRSNG

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Main Research and Development Products

- Policy IDE
  - Real-time debugging of policy programming
  - GUI
- TOSServ
  - Centralized TOSSIM Service
- Finger2IPv6
  - Policy programming could now occur on real wireless sensor devices, using IPv6

Code available on GitHub (2 dev. branches):

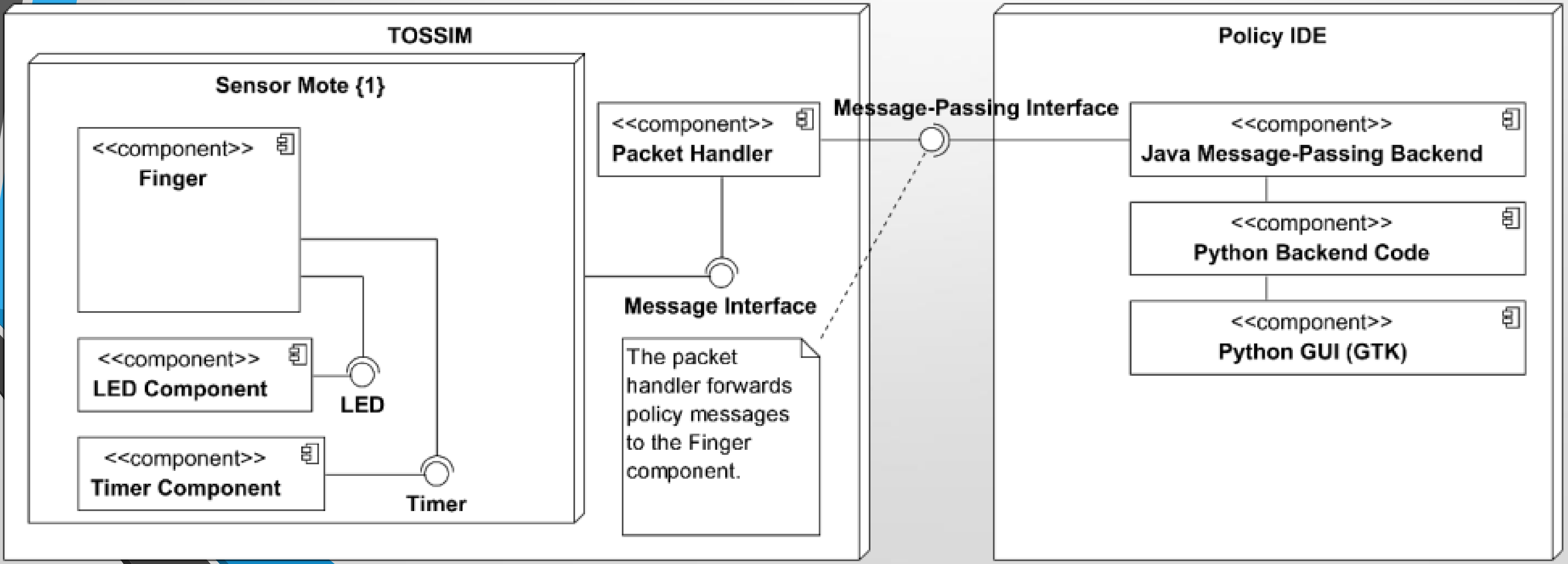
github.com/drspangle/tinyos-main/tree/Finger2IPv6

github.com/drspangle/tinyos-main/tree/TOSServ

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Policy IDE

- Real-time debugging of policy programming using packet injection became possible.
- Main thrust of paper.
- Debug messages collected from TOSSIM, displayed on GUI in real time.


TOSSIM
Sensor Mote {1}
<<component>> Finger
<<component>> LED Component
LED
<<component>> Timer Component
Timer
<<component>> Packet Handler
Message Interface
The packet handler forwards policy messages to the Finger component.
Message-Passing Interface
Policy IDE
<<component>> Java Message-Passing Backend
<<component>> Python Backend Code
<<component>> Python GUI (GTK)

Policy IDE

- Viewing these messages and manipulating policies on simulated devices allowed introspection into the results of policies.

- Policy programming was conducted using emulated sensor motes running the Finger2 policy engine core (by Themistoklis Bourdenas).

What’s the point of policy programming?

- If, then – logical inferences:
  - Something like a rule-based system.
- Classically useful for network security:
  - Firewalls.
  - IDS.
- Special niche in the tinyOS world.

Our special niche in tinyOS…

- Allows for extensible control logic.
- tinyOS is all about creating components:
  - Pseudo-objects with events and actions.
- … but if you want to change anything except a variable, you have to re-flash the entire binary onto the device.
- Policy programming lets you tie it all together.
- Not as useful for very simple if conditions.
  - Great for chains of inference.
  - Great for making software robust to changing circumstances.

Leveraging the Event Driven System

- Policy programming is an application development platform that lets you sandwich together the best of two worlds:
  - Event driven architecture of tinyOS.
  - ‘Dumb’ functions.

Why a Policy IDE became a logical necessity…

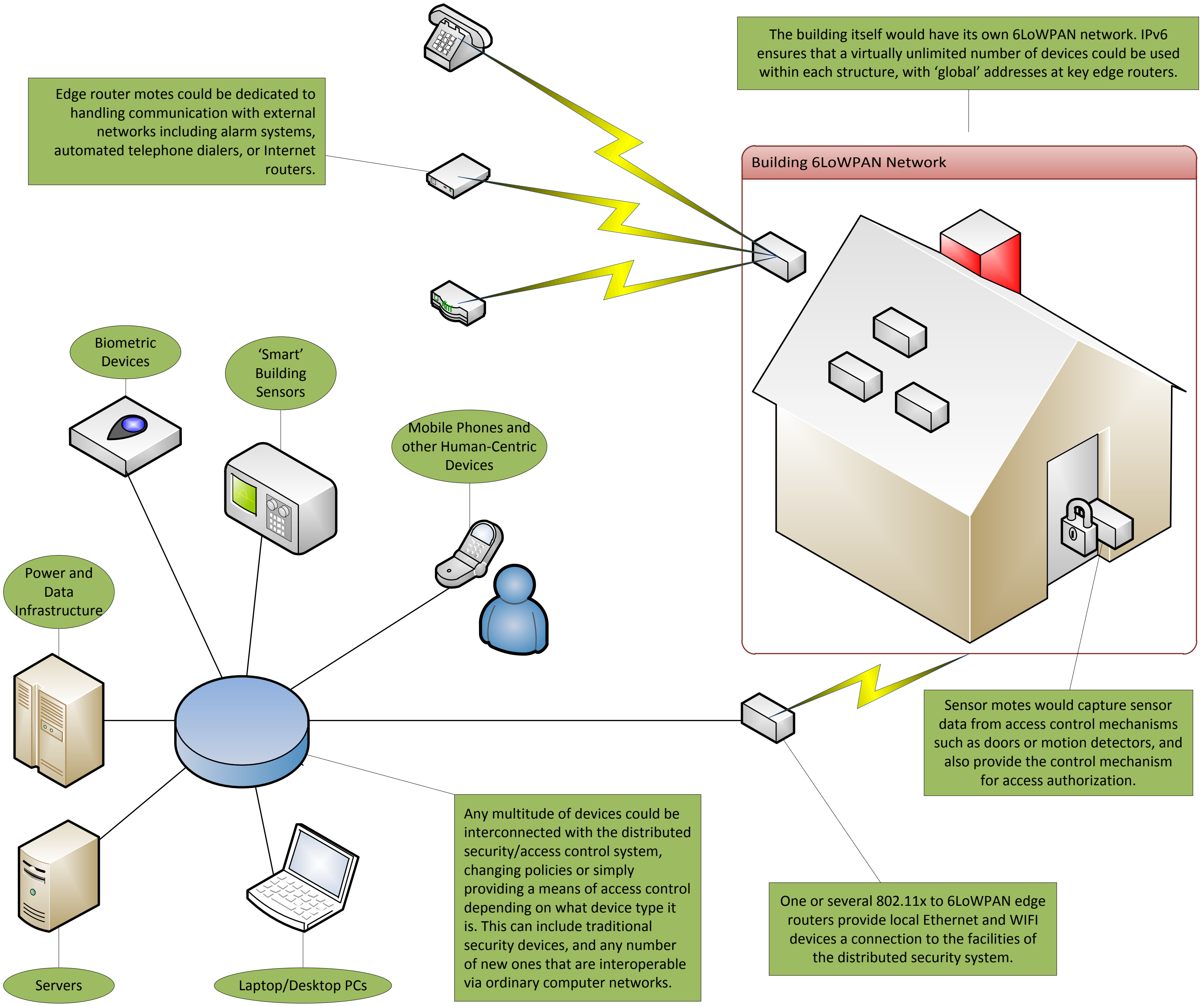
- Policy programming is essentially an informal programming language.
- Policies quickly become complicated
  - Chains of inference can get long, fast.
- Debugging and testing…
  - Facilities for triggering events, so actions can be evaluated.
  - Facilities for viewing the output in an easy way.
  - Rejoice - No more manual memory introspection!

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Making Policy-Based Applications

Key design principles:

- Reprogramming over the air can be avoided by introducing new policies, and/or updating old ones.
- Policies can be chained to one another, providing a robust event-driven rule-based architecture for larger system designs.
- Policies can be used in conjunction with highly optimized low-level nesC code units as extensible control logic.

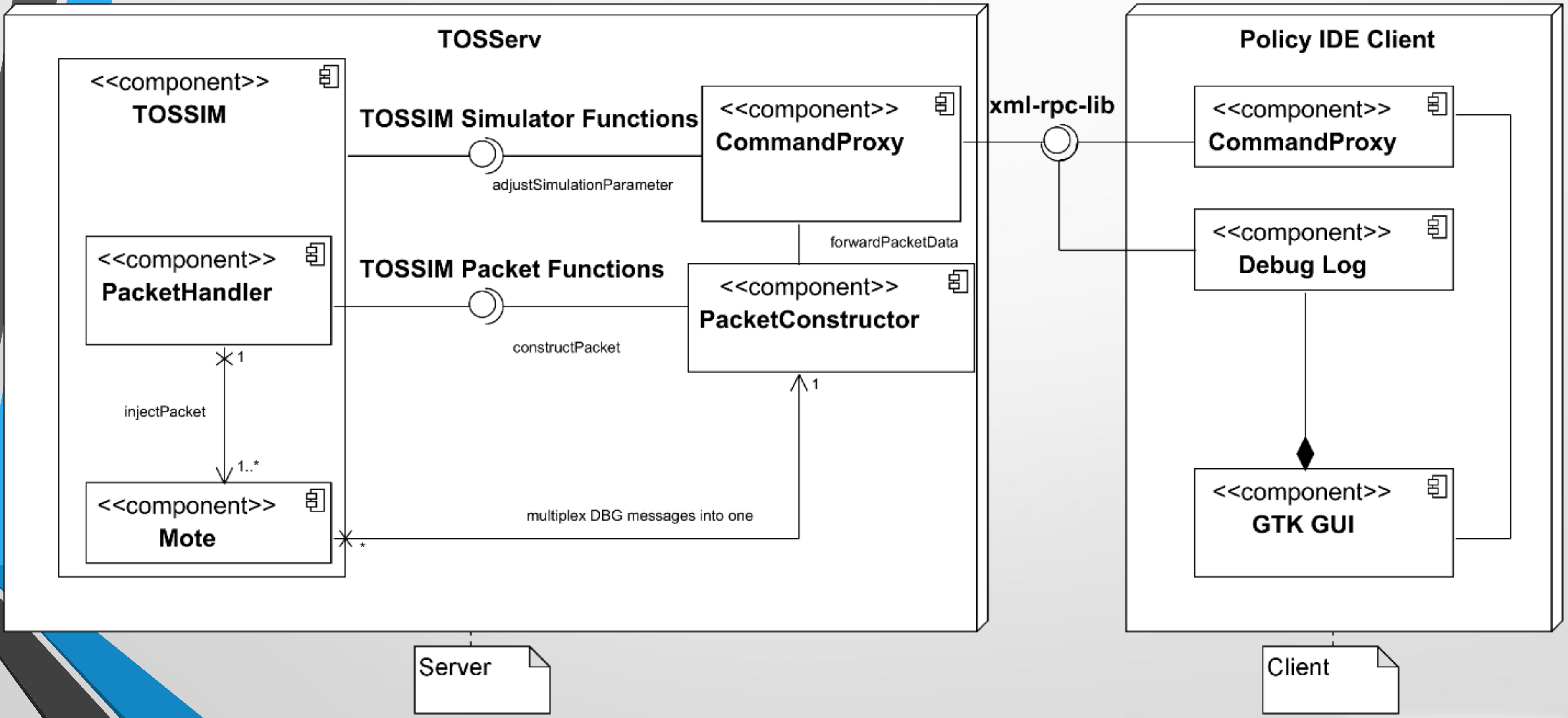

The building itself would have its own 6LoWPAN network. IPv6 ensures that a virtually unlimited number of devices could be used within each structure, with 'global' addresses at key edge routers.
Edge router motes could be dedicated to handling communication with external networks including alarm systems, automated telephone dialers, or Internet routers.
Building 6LoWPAN Network
Biometric Devices
'Smart' Building Sensors
Mobile Phones and other Human-Centric Devices
Power and Data Infrastructure
Sensor motes would capture sensor data from access control mechanisms such as doors or motion detectors, and also provide the control mechanism for access authorization.
Any multitude of devices could be interconnected with the distributed security/access control system, changing policies or simply providing a means of access control depending on what device type it is. This can include traditional security devices, and any number of new ones that are interoperable via ordinary computer networks.
One or several 802.11x to 6LoWPAN edge routers provide local Ethernet and WIFI devices a connection to the facilities of the distributed security system.
Servers
Laptop/Desktop PCs

TOSServ

- Allows TOSSIM to sit in its own environment, happy with it’s old version of gcc 3 and peculiar build chain.
- Distributed from the Policy IDE client using xml-rpc (Python).
- Needs further development.

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

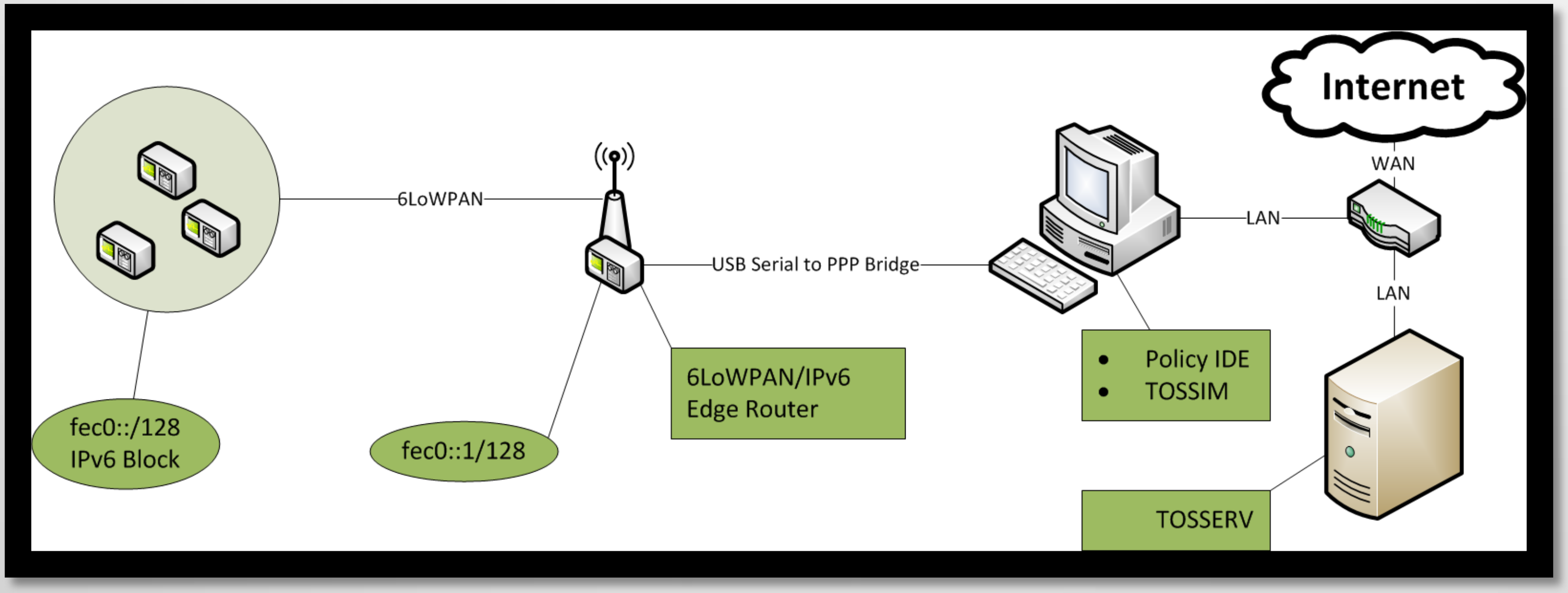
TOSServ



UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Transitioning to IPv6

The basic picture laid out…



Finger2IPv6

- Binding the Finger2 policy engine core with blip (Berkeley Low-Power Internet Stack).
- Finger2 was previously bound to AMPacket, to work with TOSSIM.
- TOSSIM doesn't support blip (due to low-level code optimizations, very convenient but quirky tinyOS API hacks).

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Finger2IPv6

- Now we can do what Policy IDE does from the command line.
  - Command-line interpreter similar to Python, except for Policy Programming.
  - Load, delete, enable, disable, and test policies.
- Next steps:
  - GUI integration.
  - A new language? Policy Definition Language.
  - Self-discovery and network policy discovery.

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY