Scaling BIRD Routeservers

RIPE 73, Madrid

Benedikt Rudolph, Sebastian Abt

DE-CIX Research and Development

Where networks meet

www.de-cix.net

The BIRD Internet Routing Daemon

- Dynamic IP routing daemon (BGP, RIP, OSPF, Babel, BFD)
- Written in C
- OpenSource Software (GNU GPLv2)
- Widely deployed for Route Servers at IXPs
  - Mid-size to small ones
  - Few large ones with ～1000 BGP neighbors
- Flexible configuration file syntax
- Reliable and stable operation

Motivation and Goals

Avoid:

- **“spiral of death”** syndrome
  - Re-convergence after high churn
- **performance degradation** on protocol reload / maintainance

Help:

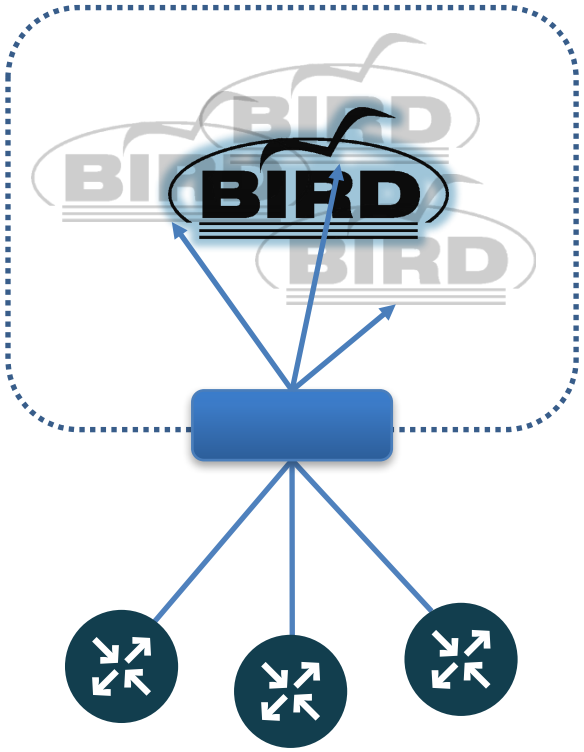
- **Scale out** to unused CPUs
- Reduce load on BIRD processes
  - Fewer peers/prefixes per PID
- Serve more peers

Easy to **test**, **deploy** and **maintain**

- No client-side configuration changes
- No alterations to the SourceCode

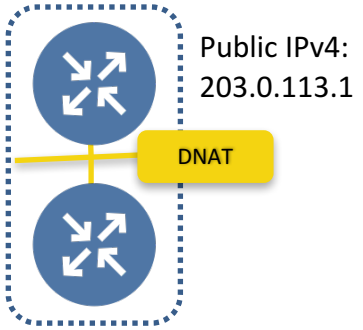
Problem Statement

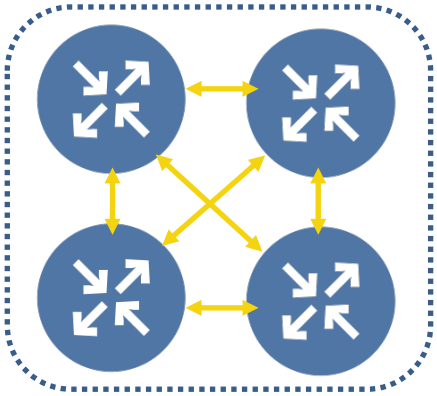
- Make multiple BIRD processes behave like one
  - Share a common RIB (master table)
  - Calculate the same BGP best-paths
  - Appear as the same BGP neighbor
  - Share an IP address
- Balance incoming BGP connections
  - Share load with 1, 2, 3,…, n processes



Solution Building Blocks

- Private subnet on loopback interface
- Link master tables (full-mesh, eBGP Add-Path)
  - iBGP fails at best-path selection
  - eBGP exhibits path-hiding
- “Balance” BGP peers based on IP subnet
  - BGPv4 *incompatible* with TCP proxies
  - DestinationNAT for incomming connections



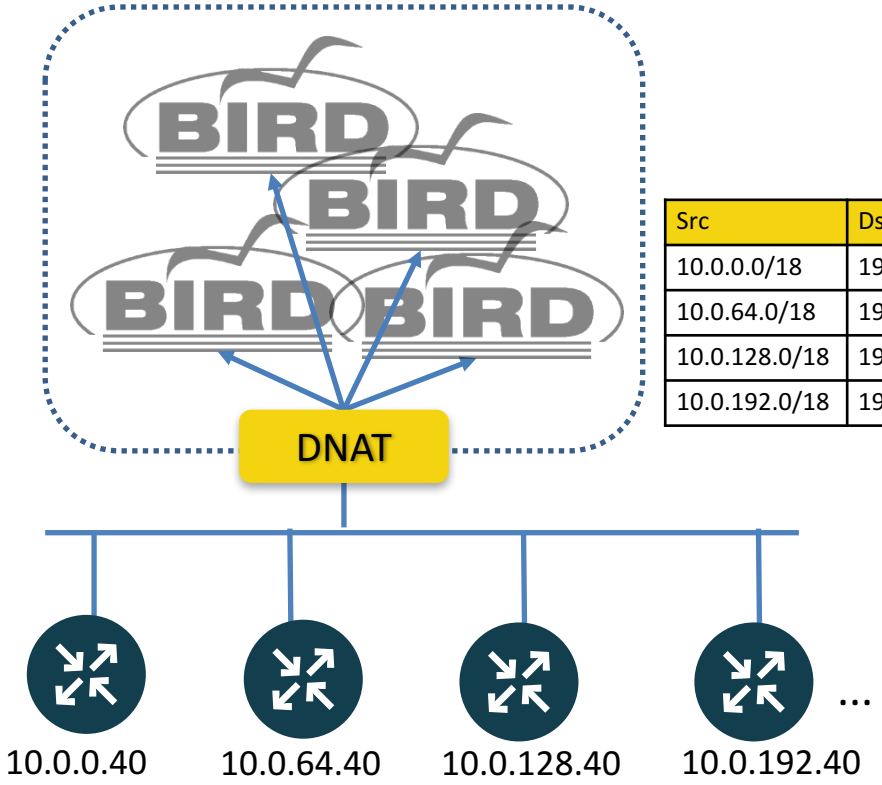


Details: Load Balancing

- Peering LAN split in **n** subnets
- DNAT: subnet to BIRD pocess

Overhead:

- Master table: ***n*** copies
- New best-path: ***n-1*** updates



Src	Dst
10.0.0.0/18	192.168.0.1
10.0.64.0/18	192.168.0.2
10.0.128.0/18	192.168.0.3
10.0.192.0/18	192.168.0.4

Test Execution with bgperf

python

`./bgperf.py`

Initialization

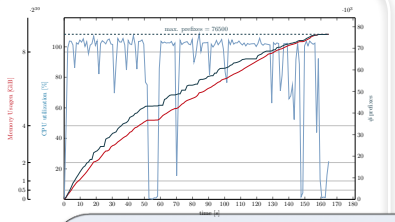
- parameters
- config files

docker

ExaBGP

Setup

- Monitor
- Tester
- Target (BIRD)



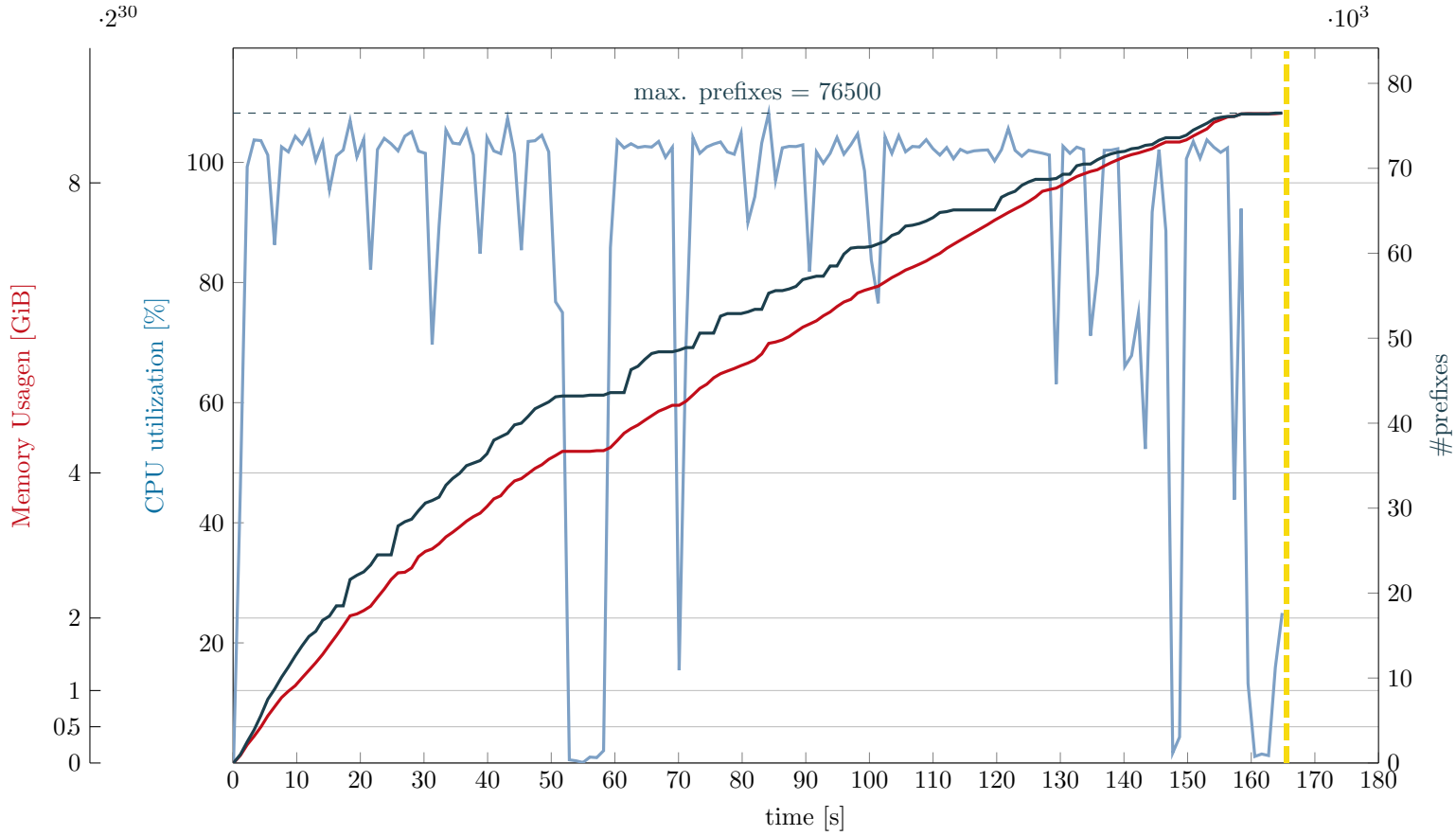
Execution

- Log Time
- Log CPU
- Log Memory

Repeat 3 times, for 1, 2, 4 processes at 252, 507, 756, 1022, 1278, 1518 peers

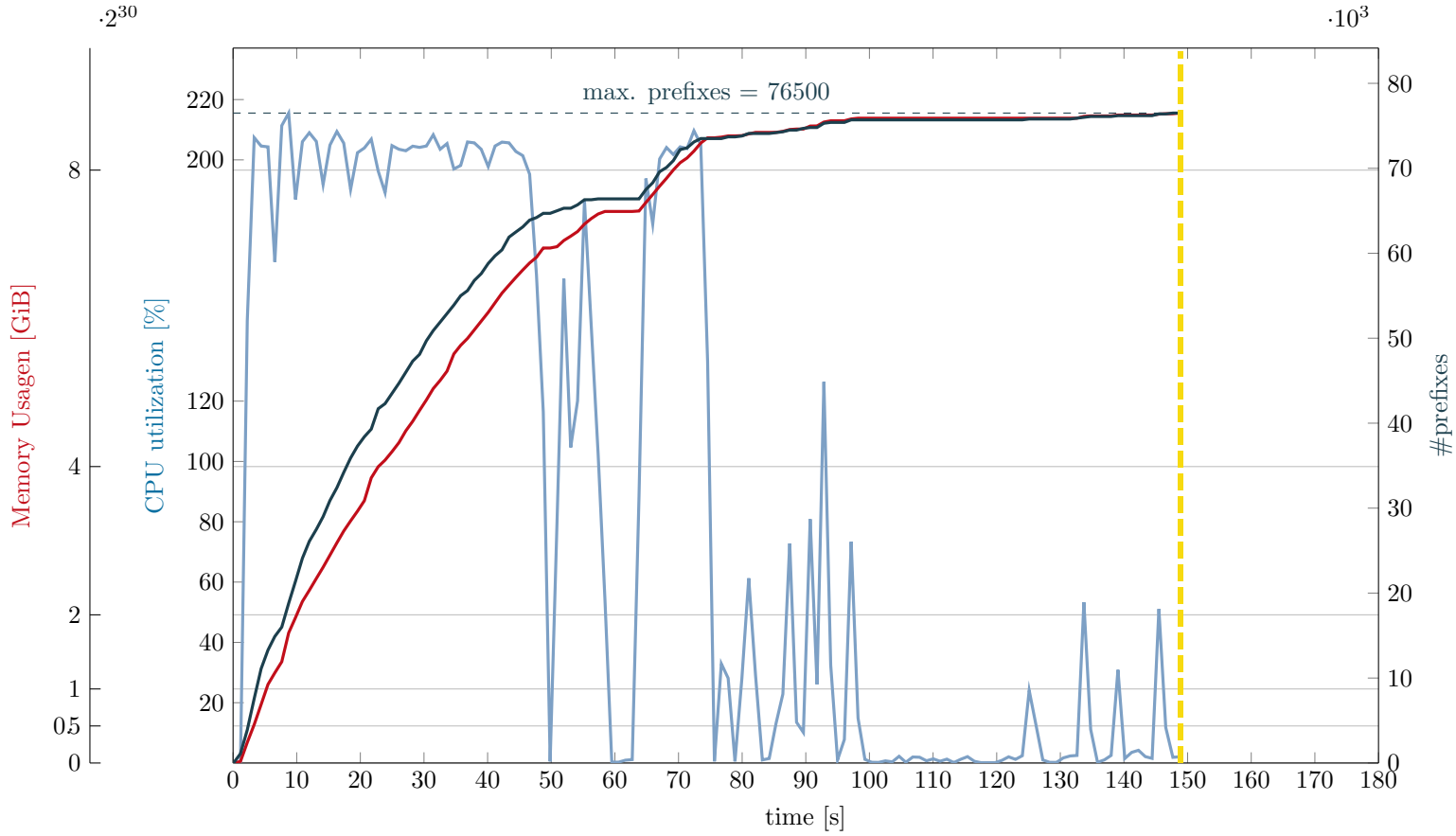
Benchmark: Use of Multiple Processes

» **1 BIRD Process**, Time to learn 76500 prefixes = **165 s**



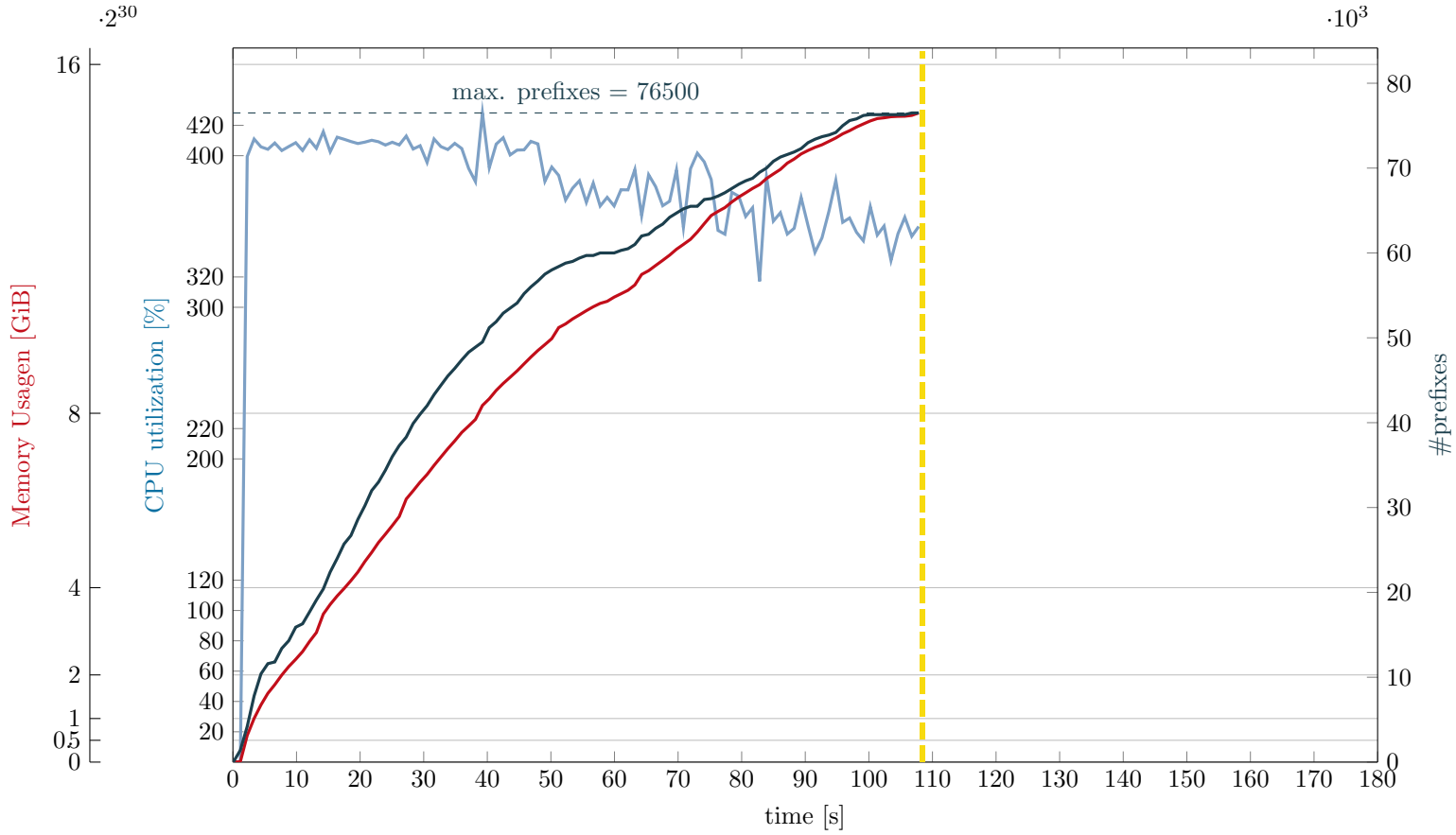
Benchmark: Use of Multiple Processes

» **2 BIRD Processes**, Time to learn 76500 prefixes = **150 s**



Benchmark: Use of Multiple Processes

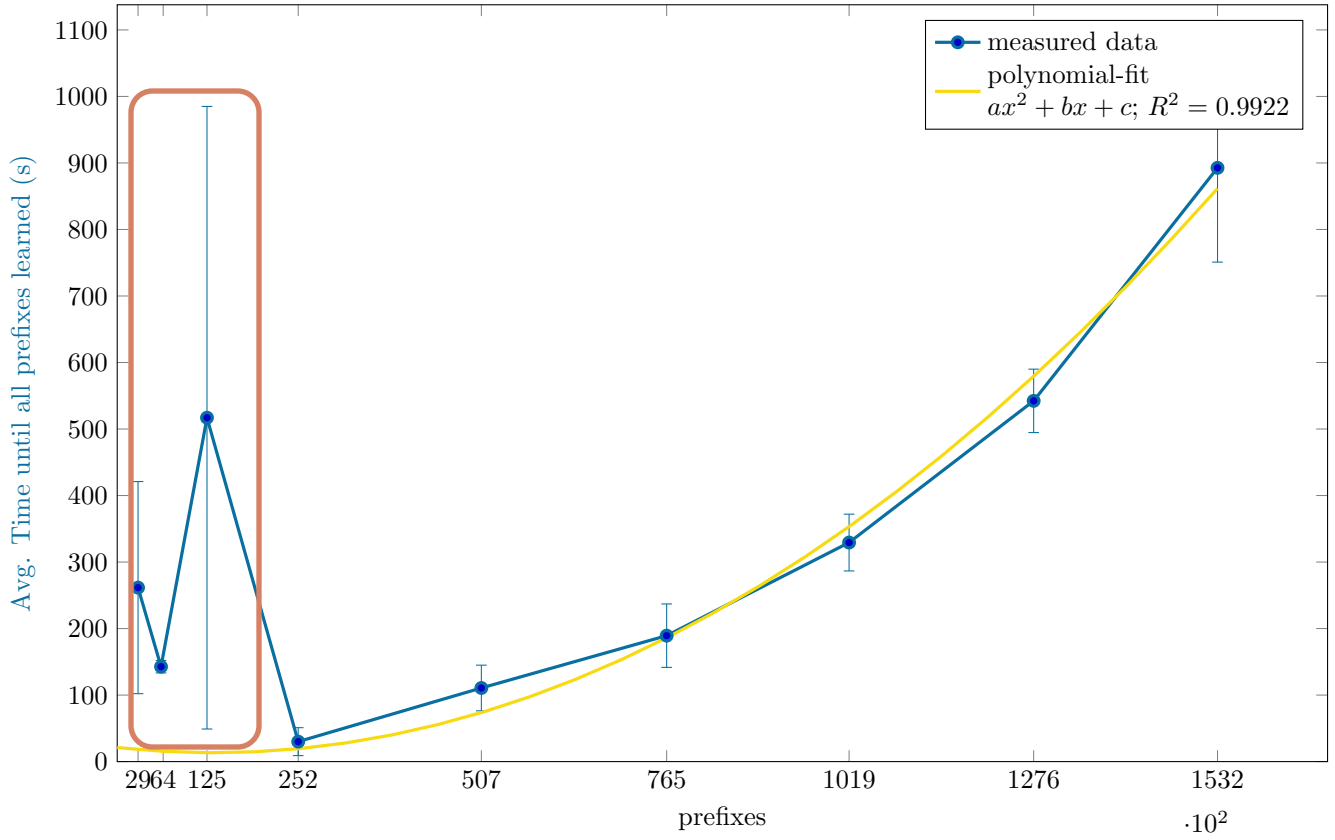
» **4 BIRD Processes**, Time to learn 76500 prefixes = **108 s**



Results: Execution Time, 1 Process

» Large variation for small settings ⇒ excluded

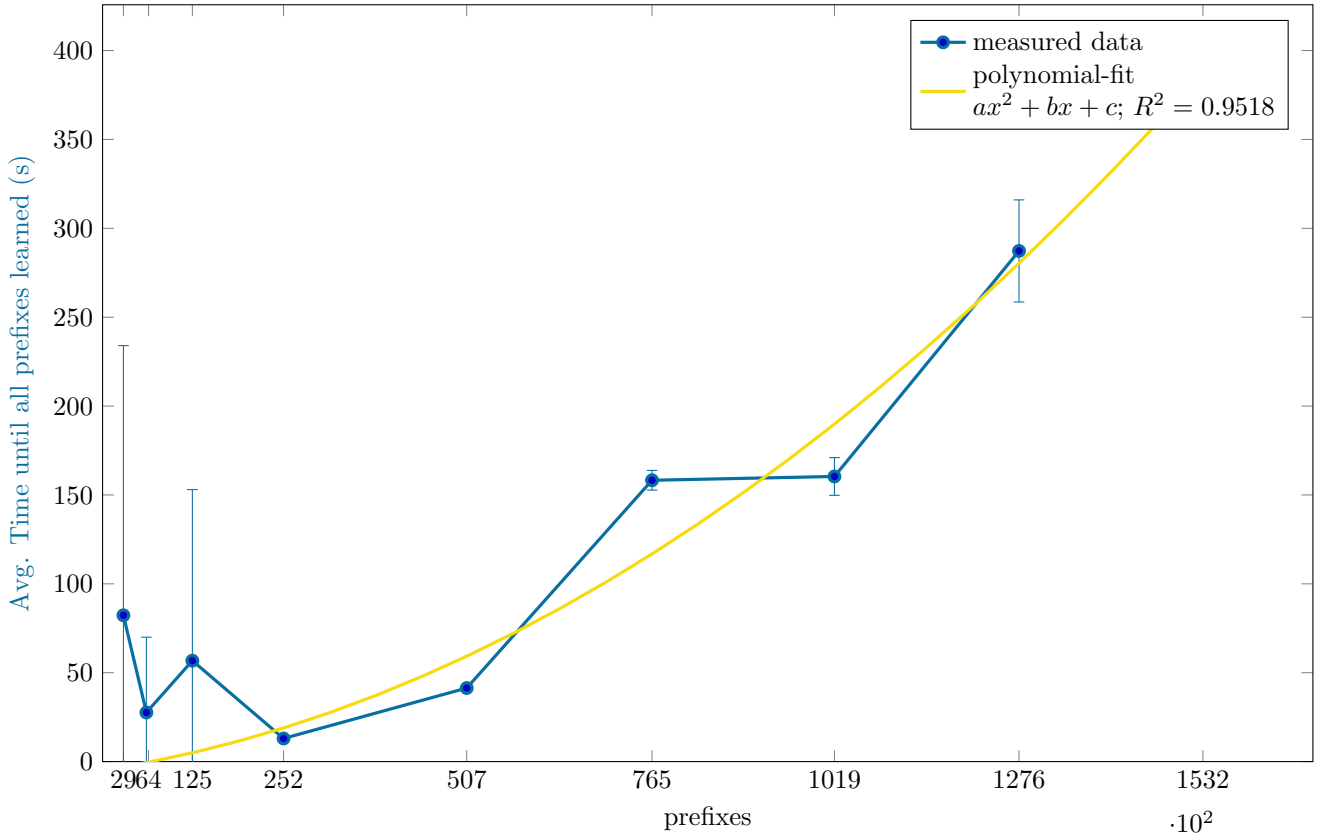
» Super linear scaling



Results: Execution Time, 2 Processes

» Low variation

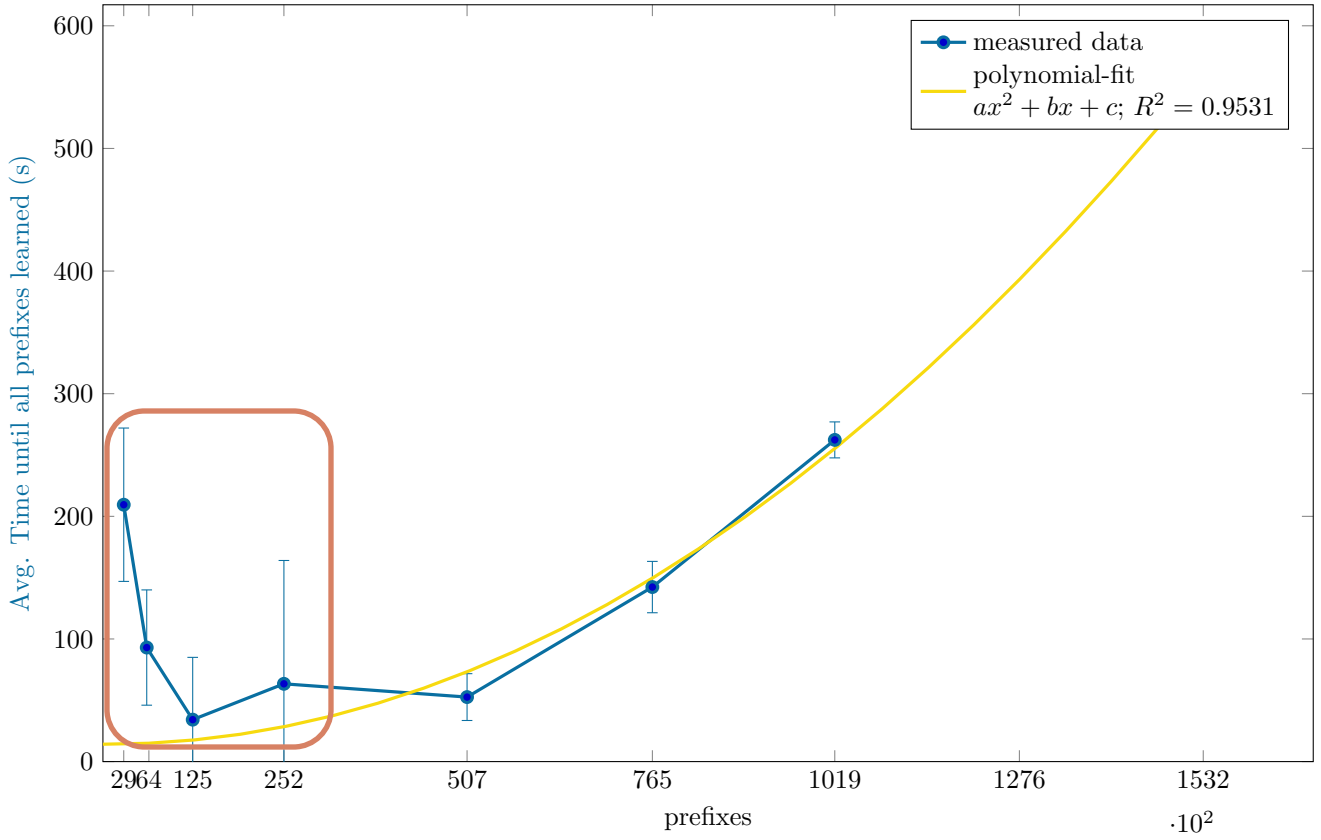
» No results for 1532 peers ⇒ RAM limit



Results: Execution Time, 4 Processes

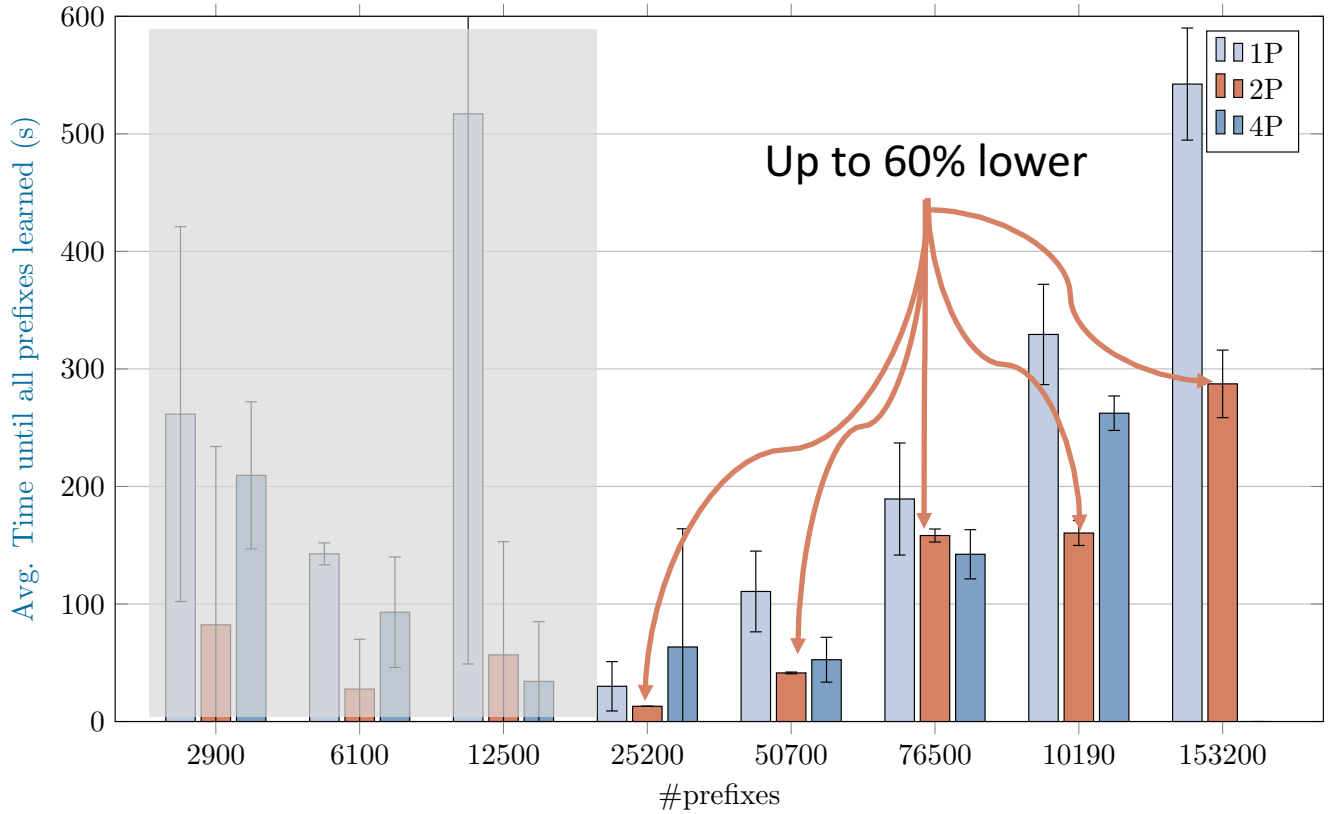
» Anomaly for small settings

» No results for 1532, 1276 peers ⇒ RAM limit



Comparison of Execution Times

- Anomalies for small peer count
- 2 BIRD processes: low overhead
- 4 BIRD processes: faster in some, but not all settings



Summary and Outlook

- Multi-process setup improves response to high load
- Increased ressource usage:
  - Moderate increase in Memory (2 processes)
  - Memory usage doubles (4 processes)
- Speedup is partly egalised by overhead
  - Replication of Master Table
  - Internal Communication: TCP over loopback interface (use Sockets)
- Possible improvements
  - Simple multi-threading in BIRD
  - Use of “realistic” peers
  - Investigate cause for “waiting“ BIRD processes (System CPU?)
  - Use a separate (physical) host for load generation (Tester)
  - Check overhead with 8 BIRD processes

Questions & Answers
Talk to us!

rnd@de-cix.net

Benedikt Rudolph, Sebastian Abt

DE-CIX Research and Development

Where networks meet

www.de-cix.net