Modifying libiptc:
Making large iptables rulesets scale

Netfilter Developers
Workshop 2008
d.1/10-2008

by
Jesper Dangaard Brouer <jdb@comx.dk>
Master of Computer Science
ComX Networks A/S

Presentation overview

- You will learn:
  - That the entire ruleset is copied to userspace (libiptc)
  - That libiptc contained some scalability issues
  - About my modifications to libiptc, to make it scale
  - Remaining issues with iptables userspace locking

Ruleset copied to userspace

- Ran into scalability issues with iptables
  - when having large amount of chain
- Discover *how iptables works*:
  - Entire ruleset copied to userspace
  - After possibly multiple changes, copied back to kernel
- Performed by a IPTables Cache library "libiptc"
  - iptables.c is a command line parser using this library
- Profiling...

libiptc: scalability issues

- Minor:
  - Inline functions iptcc_is_builtin() and set_changed()
  - Don't sort all chains on pull-out, only on insert
- Major:
  - Initial ruleset parsing slow
  - Chain name lookup slow

Issue: Initial ruleset parsing

- First scalability issue identified:
  - Initial ruleset parsing
- Problem:
  - Resolving jump chains is slow O(Chain*Rules)
  - Each jump chain resolved
    - Linearly, offset based, search of chain list
- Postpone fix
  - Take advantage pull-out and commit system
    - Only one “initial ruleset parsing” penalty

Actually like pull-out and commit system

- Take advantage of pull-out and commit system
  1. Pull-out ruleset (*one initial ruleset parsing penalty*)
  2. Make all modification needed
  3. Commit ruleset (to kernel)
  - This is how ***iptables-restore*** works
- Extra bonus:
  - Several rule changes appear atomic
    - Update all rules related to a customer at once
    - No need for temp chains and renaming

Perl - IPTables::libiptc

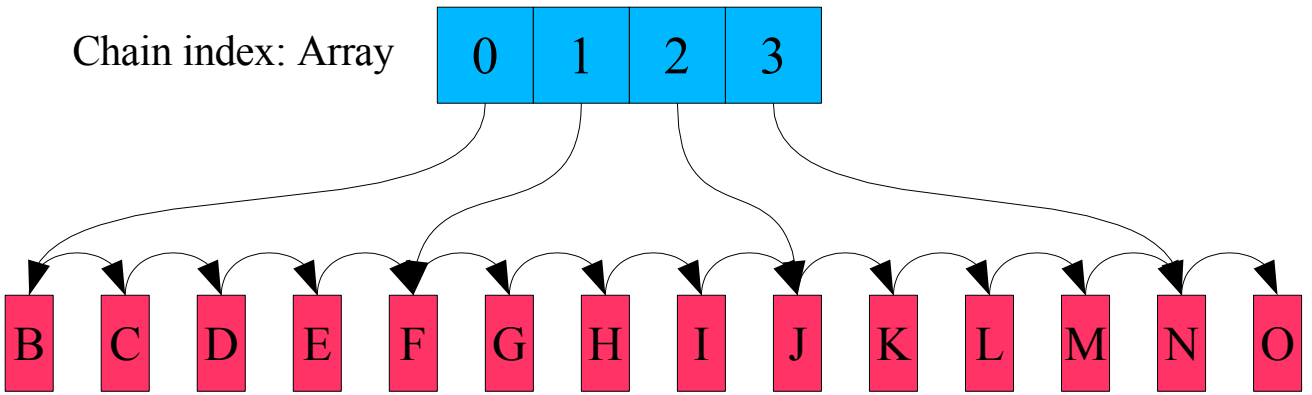
- Cannot use iptables-restore/save
  - SubnetSkeleton must have is_chain() test function
- Created CPAN IPTables::libiptc
  - Chains: Direct libiptc calls
  - Rules: Command like interface via iptables.c linking
  - iptables extensions available on system, dynamic loaded
  - No need to maintain or port iptables extensions
  - Remember to commit()
- Postponed fixing "initial ruleset parsing"

Next scalability issue: Chain lookup

- Slow chain name lookup
  - is_chain() testing (internal iptcc_find_label())
  - Cause by: linearly list search with strcmp()
- Affects: almost everything
    - Rule create, delete, even listing.
    - Multiple rule changes, eg. iptables-restore, SubnetSkeleton
- Rule listing (iptables -nL) with 50k chains:
  - Takes approx 5 minutes!
  - After my fix: reduced to 0.5 sec.

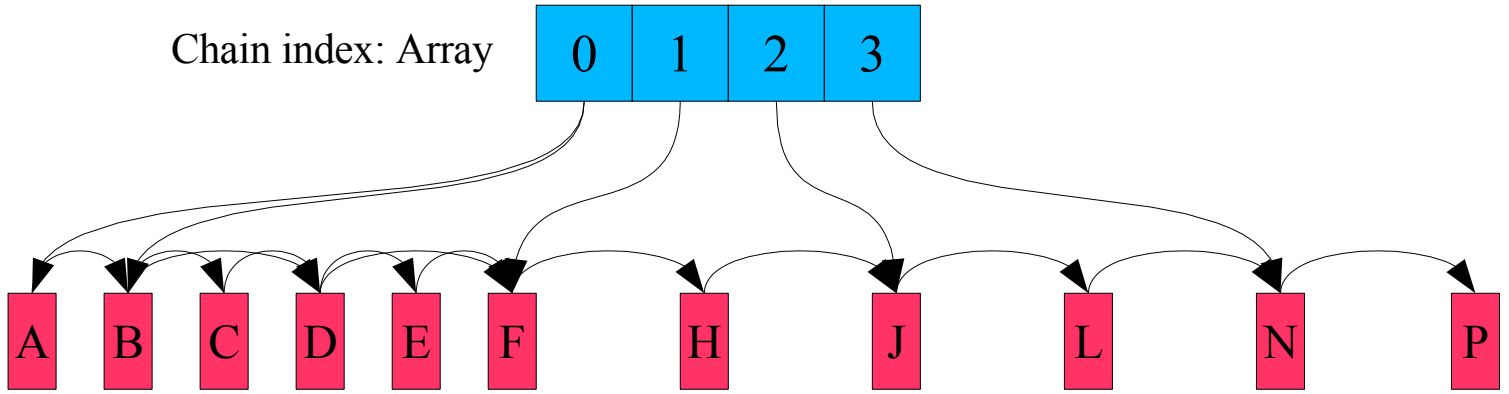
Chains lookup: Solution

- Solution: binary search on chain names
  - Important property chain list is sorted by name
  - Keep original linked list data structure
  - New data structure: "Chain index"
    - Array with pointers into linked list with a given spacing (40)
  - Result: better starting points when searching the linked list



Chain index: Insert chain

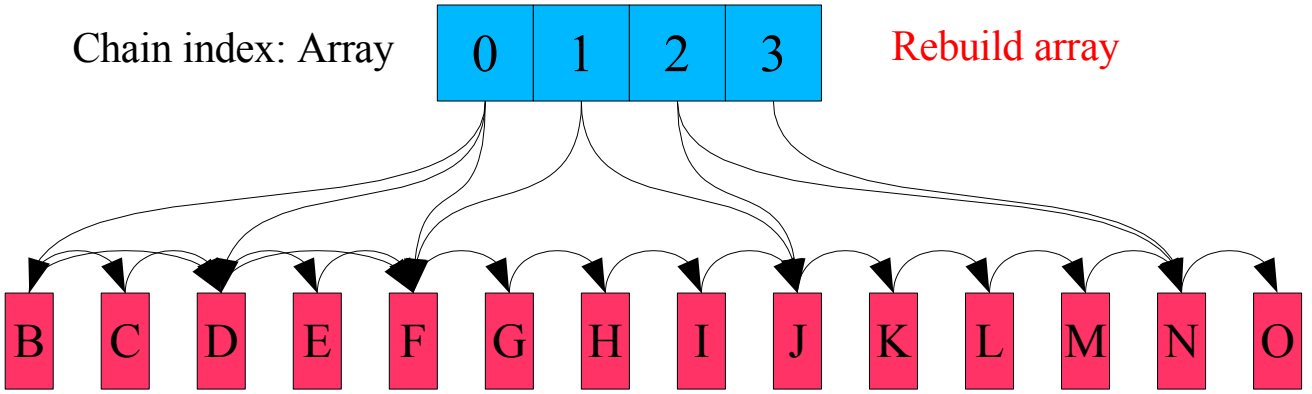
- Handle: Inserting/creating new chains
  - Inserting don't change correctness of chain index
    - only cause longer lists
    - rebuild after threshold inserts (355)
  - Inserting before first element is special



Chain list: linked list, sorted by chain name

Chain index: Delete chain

- Handle: deletion of chains
  - Delete chain ***not*** pointed to by chain index, no effect
  - Delete chain pointed to by chain index, possible rebuild
    - Replace index pointer with next pointer
    - Only if next pointer not part of chain index



Chain list: linked list, sorted by chain name

Solving: Initial ruleset parsing

- Back to fixing "initial ruleset parsing".
  - Did have a fix, but was not 64-bit compliant (2007-11-26)
- Problem: Resolving jump rules is slow
  - For each: Jump Rule
    - Do a linearly, offset based, search of chain list
- Solution:
  - Reuse binary search algorithm and data structure
  - Realize chain list are both sorted by name and offsets
    - Ruleset from kernel already sorted

Summary: Load time

- Personal firewall
  - Reload all rules on a production machine
    - Chains: 5789
    - Rules: 22827

Number of calls	74659	
Total time used	1.92	sec
Average per call	0.00002567	sec

action	**calls**	**time**	**per call**
set_policy	1	0.00007701	0.00007701
append_rule	8399	0.49619532	0.00005908
insert_rule	4463	0.24729586	0.00005541
flush_entries	4726	0.03449988	0.00000730
init	1	0.04638195	**0.04638195**
commit	1	0.08120894	0.08120894
list_rules_IPs	1181	0.02705002	0.00002290
is_chain	46965	0.37487888	**0.00000798**
delete_rule	8922	0.60892868	0.00006825
Sum	**74659**	**1.91651654**	sec

Total time entire script	23.72	sec

Summary: Open Source

- Open Source Status
  - *Chain lookup* fix
    - In iptables version 1.4.1
      - 50k chains, listing 5 min -> 0.5 sec
  - *Initial ruleset parsing* fix
    - In iptables version 1.4.2-rc1
      - Production, reached 10 sec -> 0.053 sec
  - IPTables::libiptc
    - Released on CPAN
  - IPTables::SubnetSkeleton
    - Available via http://people.netfilter.org/hawk/

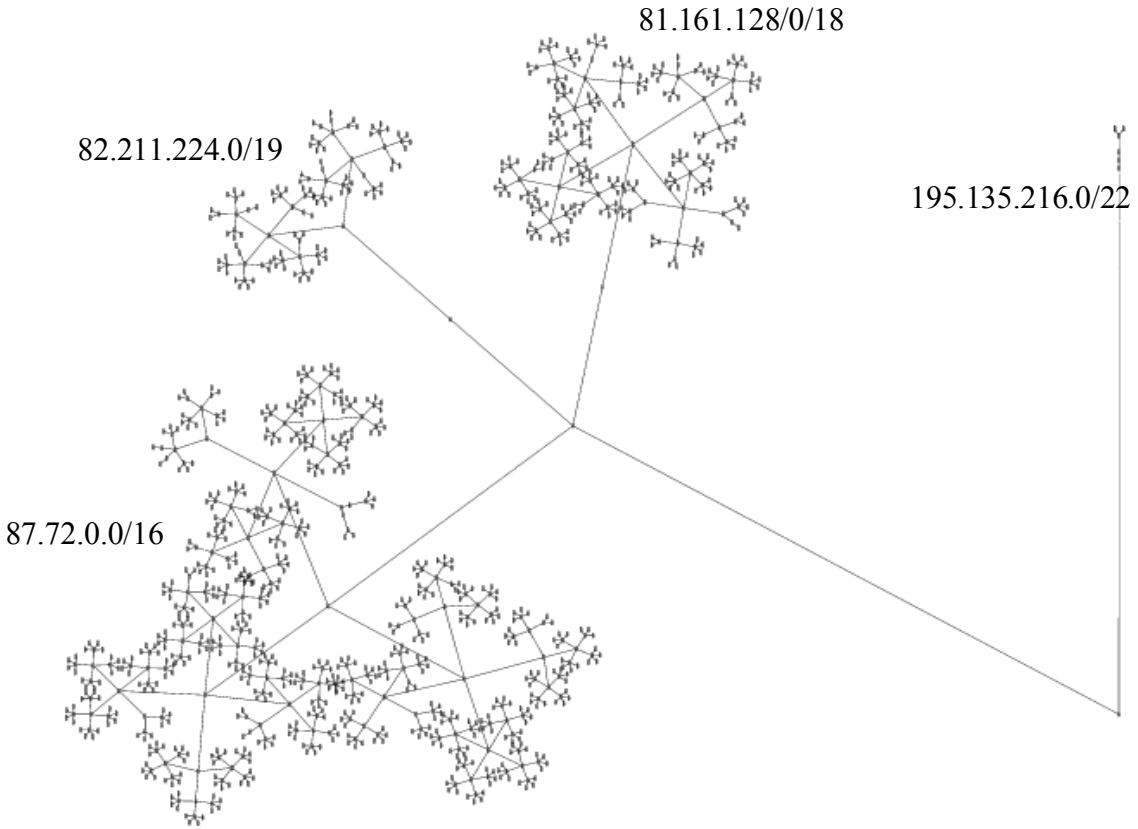
Remaining issue: No locking

- Ruleset pull-out and commit system
  - Problem: Userspace race condition
    1. Two processes pull-out ruleset
    2. Process#1 commit
    3. Process#2 commit ... what happens!?
       - if ruleset entries are the same, p#2 overwrite p#1 rules
         - possibly wrong counter updates
       - if ruleset entries differ, p#2 fail with an errno=EAGAIN
- My solution: Simple file lock (flock) in /var/lock/
- Discussion?
  - Don't lock on “-L” listing, because cannot use in a pipe

The End

Goodbye

and thank you for accepting the patches...



Extra slides

- Bonus slides
  - if time permits
  - or funny questions arise