Introducing NPF in NetBSD 6.0

Mindaugas Rasiukevicius
The NetBSD Project
rmind@netbsd.org

October 17, 2012

Introduction

What is NPF?

NPF - is a **Net**BSD packet filter, which can do TCP/IP traffic filtering, stateful inspection and network address translation.

Introduction

Motivation: multi-core world

Rapid growth of multi-core systems demands improvements to network stack and packet filtering.

- NetBSD and other systems are innovating into the network stack parallelisation. Packet filters become a bottleneck.
- There was no SMP optimised packet filter in *BSD.
- In 2009, Linux Netfilter project invents *nftables*, with one of the main features: *"the core is completely lockless ..."*.

Introduction

Motivation: 3rd party extensions

Users and vendors often need custom solutions.

- There was no packet filter in *BSD with an emphasis on modularity.
- Linux Netfilter provided the most convenient framework for custom extensions.
- GPL might be an issue. There are known GPL-related legal disputes with vendors using Netfilter/iptables.

Features

Highlights

Hence, **NPF**:

- Written from scratch with a focus on high performance and SMP optimisations.
- Stateful packet filtering and network address translation.
- Modularity and support for extensions.
- Protocol independence in the NPF core engine.
- Support for “tables”: storage designed for large IP sets and frequent updates.
- 2-clause BSD license: liberal and vendor-friendly.
- IPv6 support, extensions for normalisation, logging and more!

Technical points

Packet classification

NPF packet classification engine is based on instruction processing.

- Inspired by the original Berkeley Packet Filter (BPF), NPF uses its own n-code. It consists of CISC-like instructions for the common patterns to reduce the processing overhead.
- BPF byte-code with just-in-time (JIT) compiler support is planned for a future release.
- This design allows us to have protocol independence, e.g. support for a new protocol can be added without any modifications to the kernel part.

Similar approach was taken by Linux *nftables*.

Technical points

Multi-processing and performance

Approach to performance and concurrency on SMP.

- Ruleset reload is performed as a single, one step commit with minimum performance impact on the packet processing.
- Tracked connections are stored in a hashed tree, with distributed locks and thus minimised lock contention.
- Various other components are lockless.
- Large IP sets can be stored in NPF tables for very efficient lookups. Storage can be chosen to be either a **hash table** or a **Patricia radix tree**.

NPF tables are similar to the “ipset” module of Linux Netfilter.

Technical points

Modularity

- NPF is modular, each component is abstracted and has its own strict interface.
- *Rule procedures* in NPF are the main interface to implement custom extensions. The syntax of npf.conf supports arbitrary procedures with their parameters, as supplied by the modules.
- An extension consists of two parts: a dynamic module (.so file) supplementing the npfctl(8) utility and a kernel module.
- Just ∼160 lines of code for a demo extension, which blocks an arbitrary percentage of traffic. No modifications to the NPF core.
- API will be fully available in NetBSD 6.1 release and is already available in NetBSD -current tree.

Technical points

Stateful inspection and NAT

- NPF implements stateful filtering. It performs full tracking of TCP connections. This means not only tracking of source and destination IP addresses with port numbers, but also TCP state, sequence numbers and window sizes.
- Currently, NPF supports dynamic NAT: network address port translation (NAPT or also known as masquerading), port forwarding and bi-directional NAT.
- Patches for preliminary support of NPTv6 and NAT64 were developed as part of Google Summer of Code 2012.

Testing

Running and debugging NPF in the userspace

- For testing, NPF uses NetBSD's RUMP (Runnable Userspace Meta Programs) framework – a kernel virtualisation and isolation technique, which enables running of the NetBSD kernel or parts of it in the userspace, like a regular program.
- Makes debugging or profiling significantly easier due to availability of tools such as gdb(1).
- NPF regression tests are integrated into NetBSD's test suite and thus are part of periodic automated runs.

Testing

Debugging

- For every NPF subsystem, unit tests are implemented and available within npftest(8) – a program containing both the tests and NPF kernel part running as a userspace program.
- npftest(8) can also read and process tcpdump pcap files with a passed npf.conf configuration. This enables analysis of a particular stream in the userspace.
- The npfctl(8) utility has a 'debug' command which can print disassembled n-code and write the configuration in the format sent to the kernel.
- Development, debugging and testing becomes much easier.

Future directions

- More testing: improving stability and reliability. Expect API and ABI changes for some time period.
- NPTv6 and NAT64 support in a future NetBSD release.
- BPF byte-code with just-in-time (JIT) compilation.
- High availability, load balancing.
- More extensions.

End

The **Net**BSD Project

`http://www.NetBSD.org/`

2012