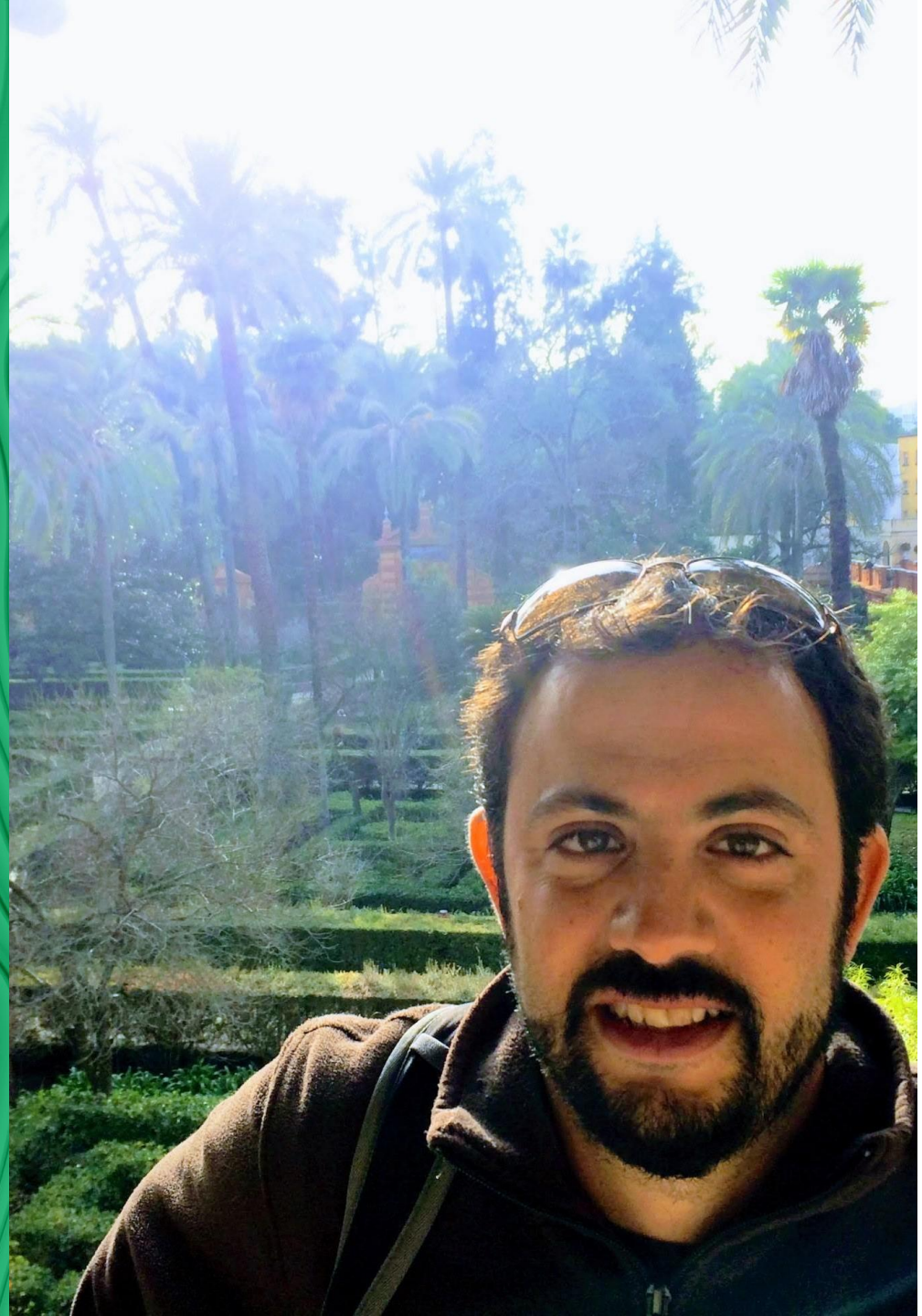
Opkg: Debian's Little Cousin

Package Management on Embedded

Alejandro del Castillo

- Chief Software Engineer



whoami

NI Linux Real-TIme

contributions

openembedded

yocto PROJECT

OPKG

jupyterhub

SALTSTACK

Agenda

History

Architecture

Solvers

Future Work

Questions

History

Ipkg (2001)

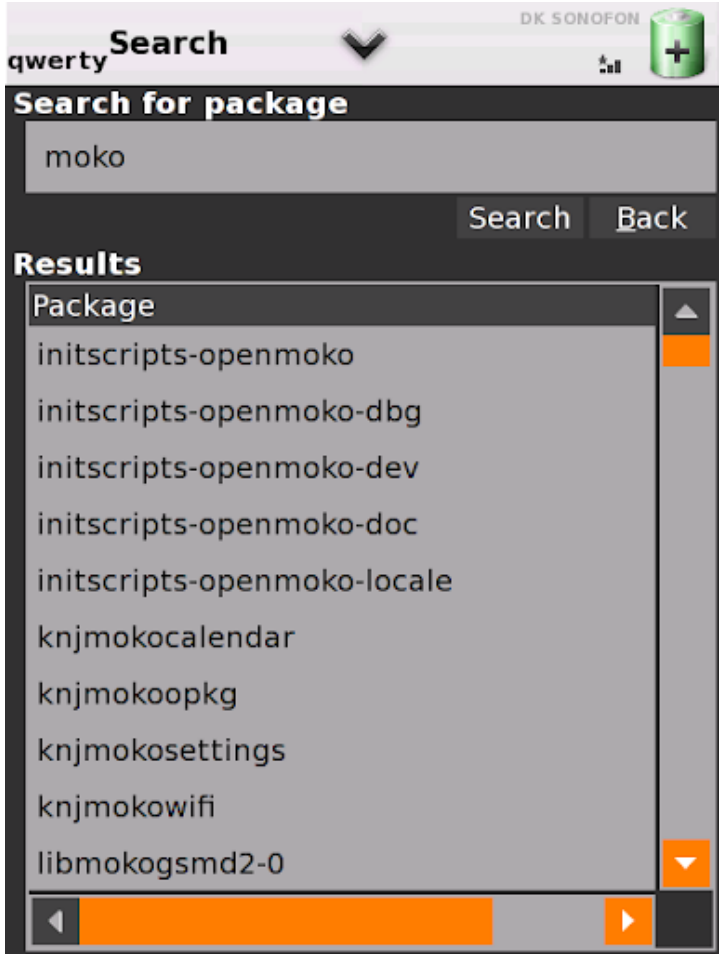
- Itsy Bitsy Package Manager
- Started as a shell script, rewritten in C
- Originally written by Carl Worth (Cairo)
- Last known commit: June 2007
- LinkSyS NSLRU 2 (Network Storage Link for USB 2.0 Disk Drives)

http://www.flickr.com

Opkg (2008)

Fork of ipkg-0.99.163 for openmoko

- IPKG no longer actively maintained
- IPKG had trademark
- Adopted by OpenEmbedded (Marcin Juszkiewicz)
- Helper scripts on opkg-utils repo (build feeds, packages, extract metadata, etc)
- Android became the de-facto standard, opkg active development mainly driven by OpenEmbedded
- Previous maintainers Thomas Wood, Tick Chen, Graham Gower, Paul Barker



http://wiki.openmoko.com

Opkg (2020)

- Under Yocto project umbrella (IT, git hosting, bugzilla)
  - https://git.yoctoproject.org/cgit/cgit.cgi/opkg
  - https://bugzilla.yoctoproject.org/buglist.cgi?product=opkg
- Mailing list on google groups
  - https://groups.google.com/forum/#!forum/opkg-devel
- Actively maintained
  - 2 releases per year (June & December)
- Mature
  - Robust solver backend
  - Basic package manager features done

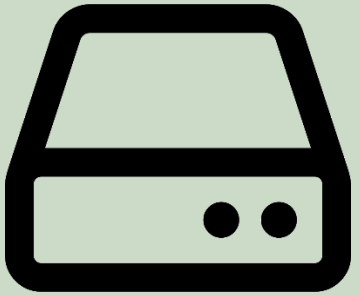
Architecture

Development Philosopy

IPKG tries to be like Debian, as much as possible. In fact, the philosophy in the development has been something like, **"Do it like Debian unless there is a strong reason not to"**. The majority of the packages in the current repository come more or less straight out of Debian/arm unstable, (with documentation removed).

Original IPKG FAQ

Why yet another Package Manager?



MACHINE = quemux86_64

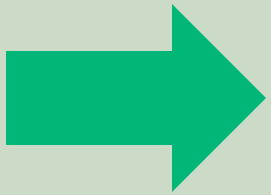
EXTRA_IMAGE_FEATURES ?= “package-management”

IMAGE = core-image-minimal

	package_ipk	package_deb	package_rpm
Package Manager	opkg	dpkg	dnf
Language	C	Perl	Python
core-image-minimal	4.6 MB	37 MB	245 MB

Ipk Structure

|-- package.ipk (ar)
  |-- debian-binary
  |-- **data.tar.{gz|xz|lz4|bz2}**
    |-- <rootdir>
  |-- **control.tar.gz**
    |-- **control**
    |-- preinst
    |-- postinst
    |-- prerm
    |-- postrm
    |-- conffiles
    |-- md5sums



Package: busybox
Version: 1.31.1-r0
Description: Tiny versions of many common UNIX utilities in a single small executable
Section: base
Priority: optional
Maintainer: OE-Core Developers
License: GPLv2 & bzip2-1.0.6
Architecture: core2-64
OE: busybox
Homepage: https://www.busybox.net
Depends: libc6 (>= 2.31+git0+1094741224), update-alternatives-opkg
Recommends: busybox-udhcpc
Source: busybox_1.31.1.bb

Differences from dpkg (Architecture field)

- **opkg**: weighted architectural structure (/etc/opkg/*.conf)

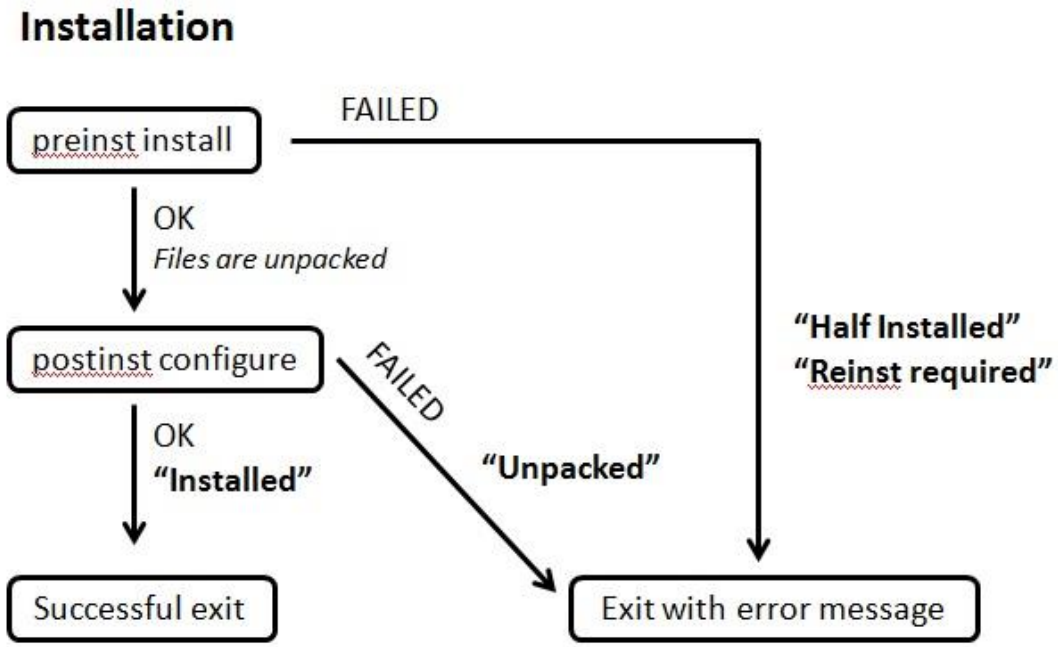
```
arch all          1
arch x86_64      16
arch core2-64    21
arch x64         26
src/gz  uri-all http://repo-server/alll
src/gz  uri-core2-64 http://repo-server/core2-64
src/gz  uri-x86_64  http://repo-server/x86_64

opkg install busybox  => busybox_core2-64.ipk
```

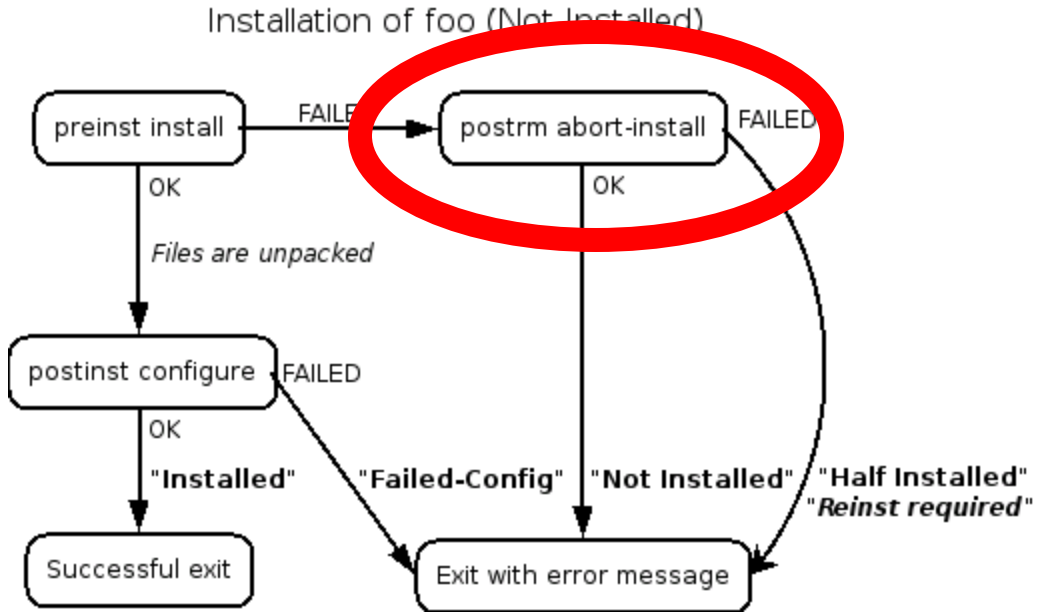
- **dpkg**: the package Architecture is matched with the target machine unique architecture (all, any and wildcards are acceptable)

State Diagram - Install

opkg

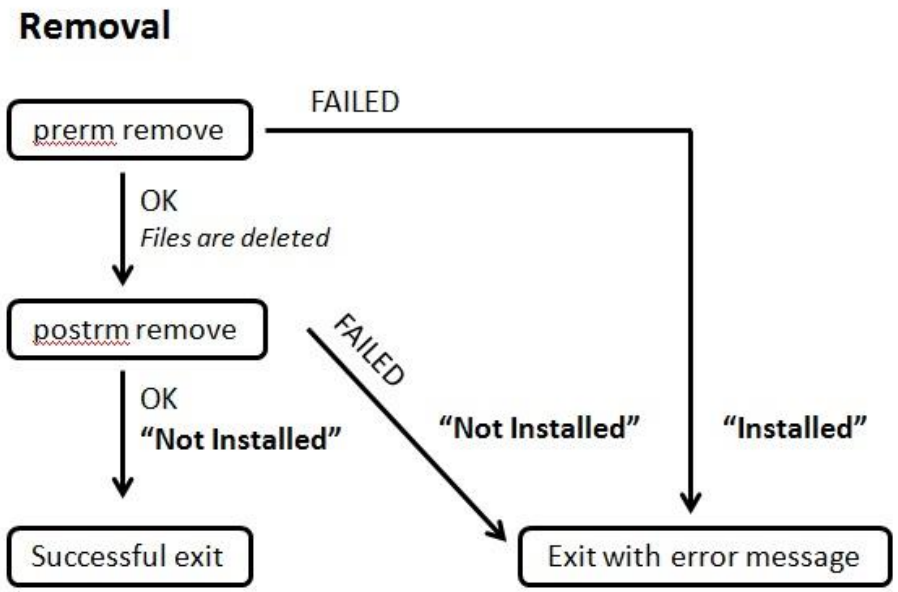


dpkg

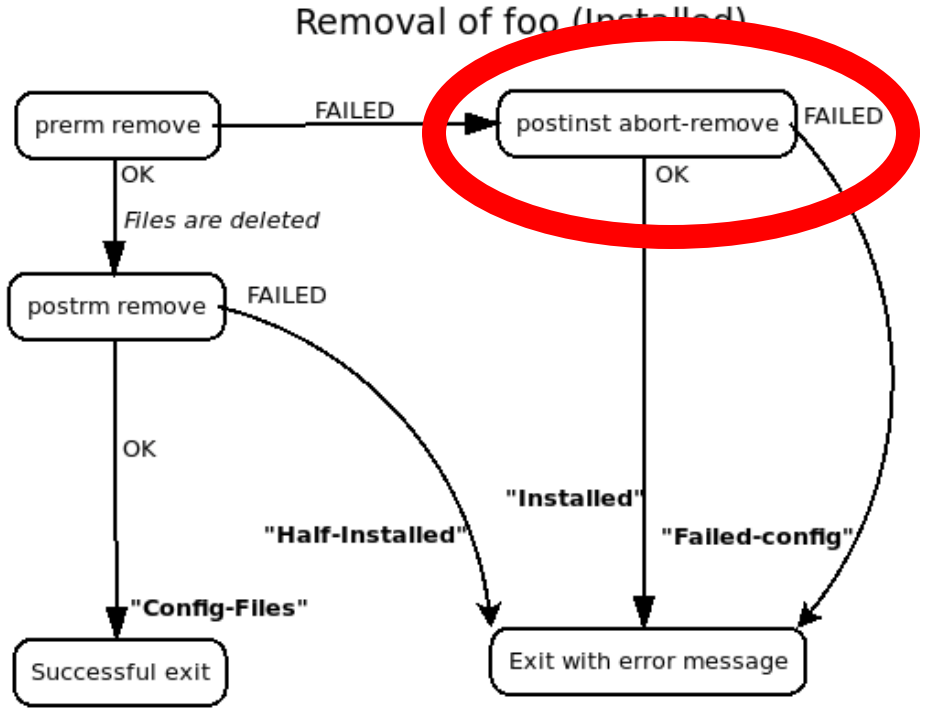


State Diagram - Remove

opkg

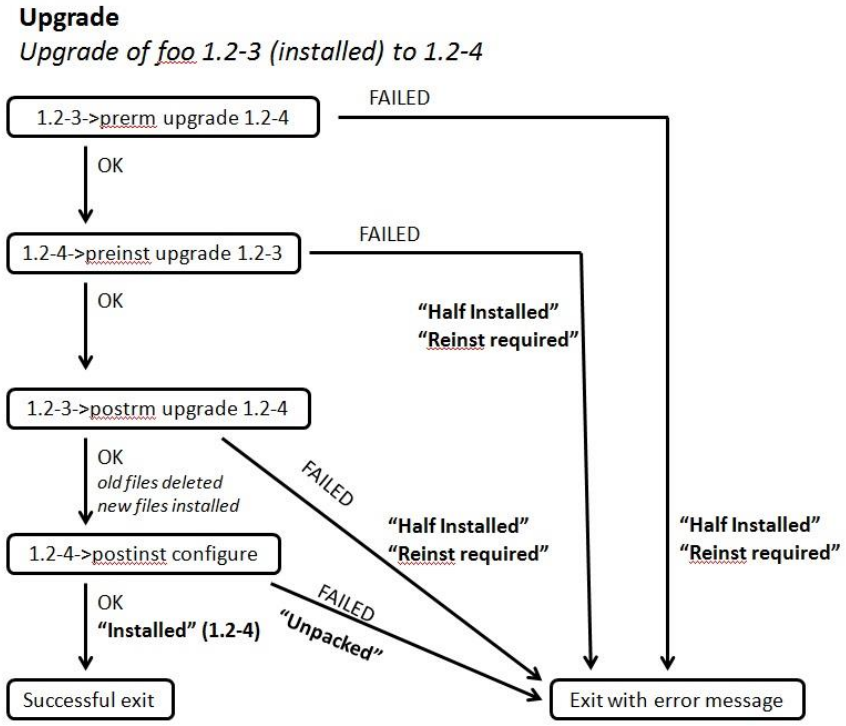


dpkg

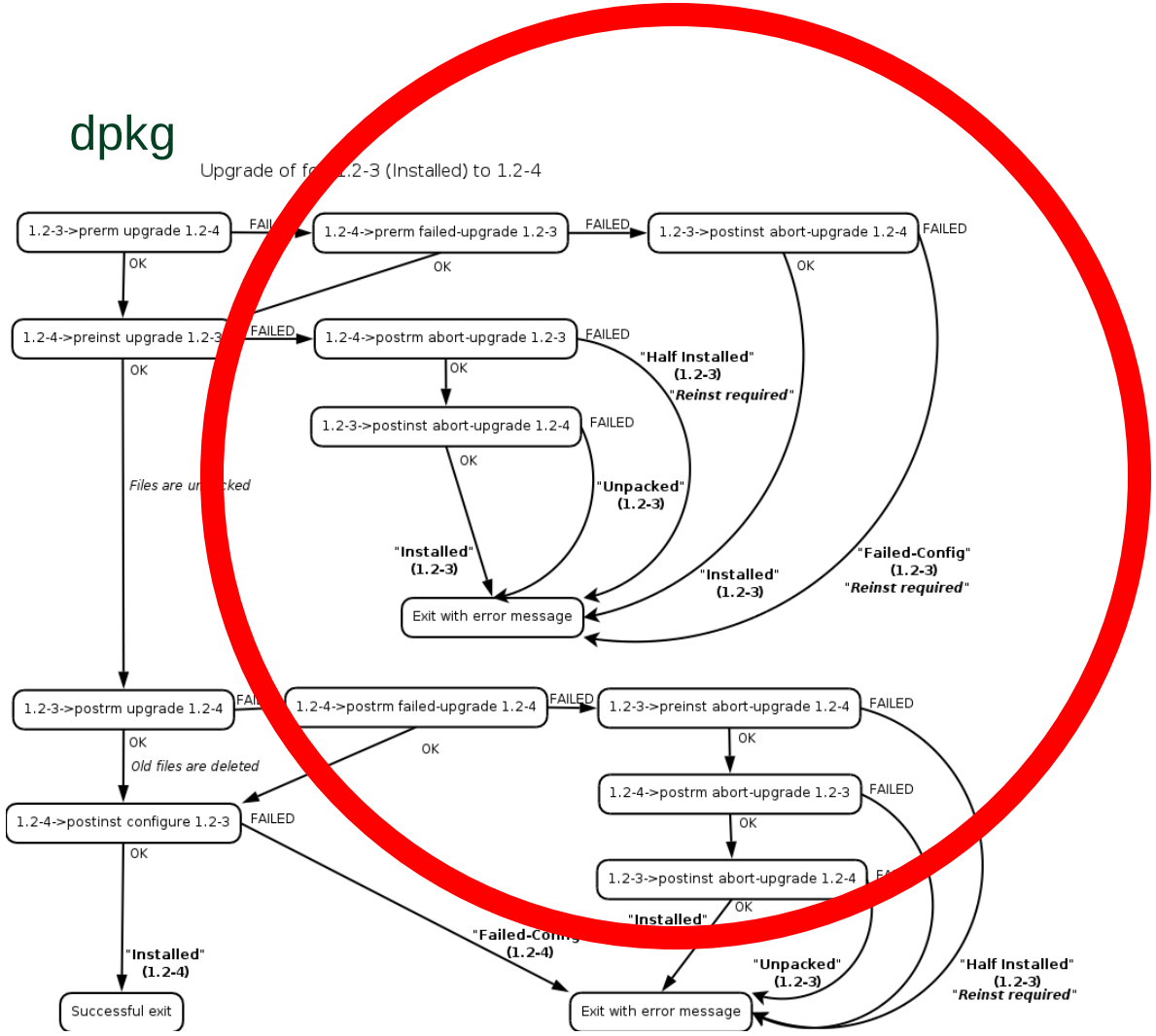


State Diagram - Upgrade

opkg



dpkg



Opkg_0.4.2.bb

DEPENDS = "libarchive“

PACKAGECONFIG ??= "libsolv"

PACKAGECONFIG[gpg] = "--enable-gpg,--disable-gpg”
PACKAGECONFIG[curl] = "--enable-curl,--disable-curl,curl"
PACKAGECONFIG[ssl-curl] = "--enable-ssl-curl,--disable-ssl-curl,curl openssl"
PACKAGECONFIG[openssl] = "--enable-openssl,--disable-openssl,openssl"
PACKAGECONFIG[sha256] = "--enable-sha256,--disable-sha256"
PACKAGECONFIG[libsolv] = "--with-libsolv,--without-libsolv,libsolv“

make check

```
#! /usr/bin/env python3
# SPDX-License-Identifier: GPL-2.0-only
#
# Install a package 'a' which depends on a second package 'b'. Check that both
# are installed
#

import os
import opk, cfg, opkgcl

opk.regress_init()

o = opk.OpkGroup()
o.add(Package="a", Depends="b")
o.add(Package="b")
o.write_opk()
o.write_list()

opkgcl.update(

opkgcl.install("a")
if not opkgcl.is_installed("a"):
   opk.fail("Package 'a' installed but does not report as installed.")
if not opkgcl.is_installed("b"):
   opk.fail("Package 'b' should be installed as a dependency of 'a' but does not report as installed.")
```

Solvers

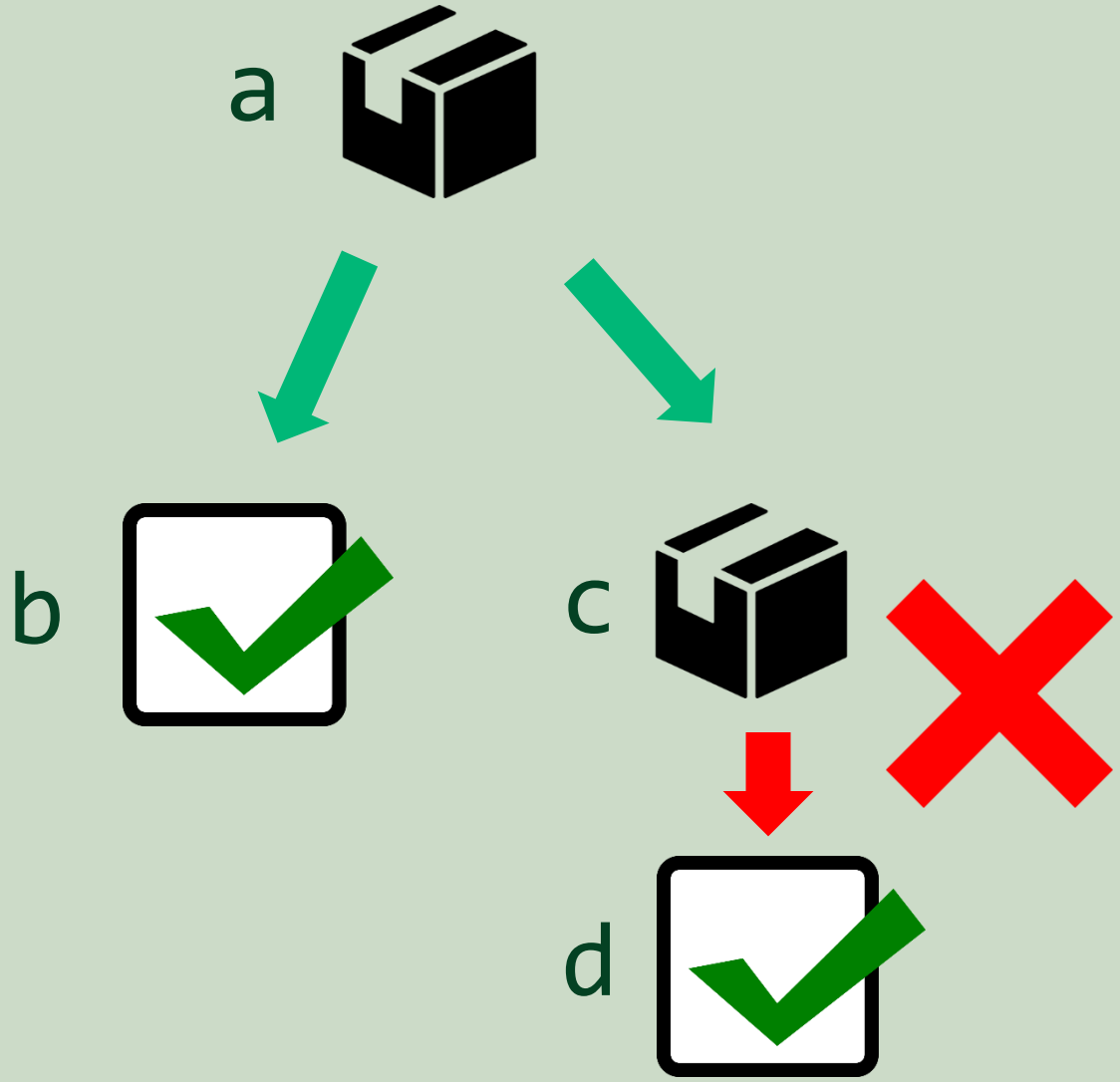
Opkg < 0.3.2 – Scenario 1

```
import os
import opk, cfg, opkgcl

opk.regress_init()

o = opk.OpkGroup()
o.add(Package="a", Depends="b,c")
o.add(Package="b")
o.add(Package="c", Conflicts="d")
o.add(Package="d")
o.write_opk()
o.write_list()

opkgcl.update()
opkgcl.install("d")
opkgcl.install("a")
```

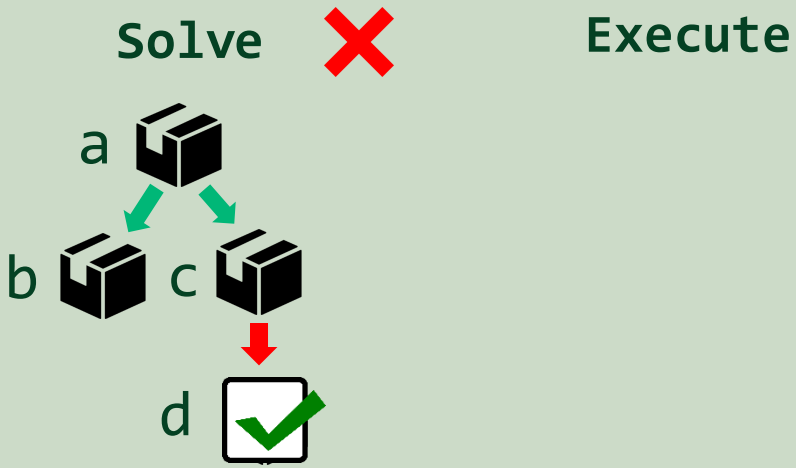


Opkg >= 0.3.2 – Scenario 1

Split solving operation in solve/execute:

```
o = opk.OpkGroup()
o.add(Package="a", Depends="b,c")
o.add(Package="b")
o.add(Package="c", Conflicts="d")
o.add(Package="d")
o.write_opk()
o.write_list()

opkgcl.update()
opkgcl.install("d")
opkgcl.install("a")
```

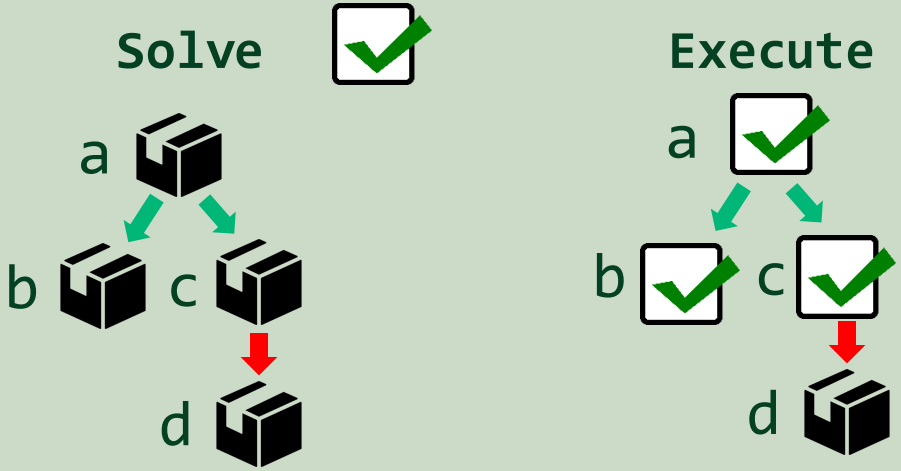


Opkg >= 0.3.2 – Scenario 2

Split solving operation in solve/execute:

```
o = opk.OpkGroup()
o.add(Package="a", Depends="b,c")
o.add(Package="b")
o.add(Package="c", Conflicts="d")
o.add(Package="d")
o.write_opk()
o.write_list()

opkgcl.update()
opkgcl.install("a")
```



However…

- Ad-hoc solver remain main cause of bugs
- Very hard to implement new features (dist-upgrade)
- Most complicated area of the codebase (dependency management is hard!)
- Dependency management is a well researched topic

author	Eric Yu <eric.yu@ni.com>	2015-08-14 11:27:54 -0500
committer	Alejandro del Castillo <alejandro.delcastillo@ni.com>	2015-08-20 16:21:55 -0500
commit	5ab9b754f9d3dc7d65c8f41303f49952f88a7e46 (patch)	
tree	4e8e87ee3668dfle7f47e50ca38lbf9c82fl6cc6	
parent	0500lecflc43078c0b07c914d9f83c5402f9c489 (diff)	
download	opkg-5ab9b754f9d3dc7d65c8f41303f49952f88a7e46.tar.gz opkg-5ab9b754f9d3dc7d65c8f41303f49952f88a7e46.tar.bz2 opkg-5ab9b754f9d3dc7d65c8f41303f49952f88a7e46.zip	

opkg_solver_libsolv.c: Add libsolv support

```
Adds opkg_solver_libsolv.c and opkg_solver_libsolv.h, and
makes changes to configure.ac, libopkg/Makefile.am, opkg_solver.c,
and pkg_hash.c to enable opkg to use libsolv as a package dependency
solver if specified when configure is run with the flag
--enable-solver=libsolv.

Signed-off-by: Eric Yu <eric.yu@ni.com>
Signed-off-by: Alejandro del Castillo <alejandro.delcastillo@ni.com>
```

Diffstat

-rw-r--r--	TODO	19
-rw-r--r--	configure.ac	16
-rw-r--r--	libopkg/Makefile.am	4
-rw-r--r--	libopkg/opkg_solver.c	20
-rw-r--r--	libopkg/opkg_solver_libsolv.c	662
-rw-r--r--	libopkg/opkg_solver_libsolv.h	70
-rw-r--r--	libopkg/pkg_hash.c	8
-rwxr-xr-x	tests/core/18_upgrade_recommends.py	2
-rwxr-xr-x	tests/misc/update_loses_autoinstalled_flag.py	2
-rwxr-xr-x	tests/regress/issue124.py	2
-rwxr-xr-x	tests/regress/issue124a.py	2
-rwxr-xr-x	tests/regress/issue50.py	2

12 files changed, 802 insertions, 7 deletions

Libsolv

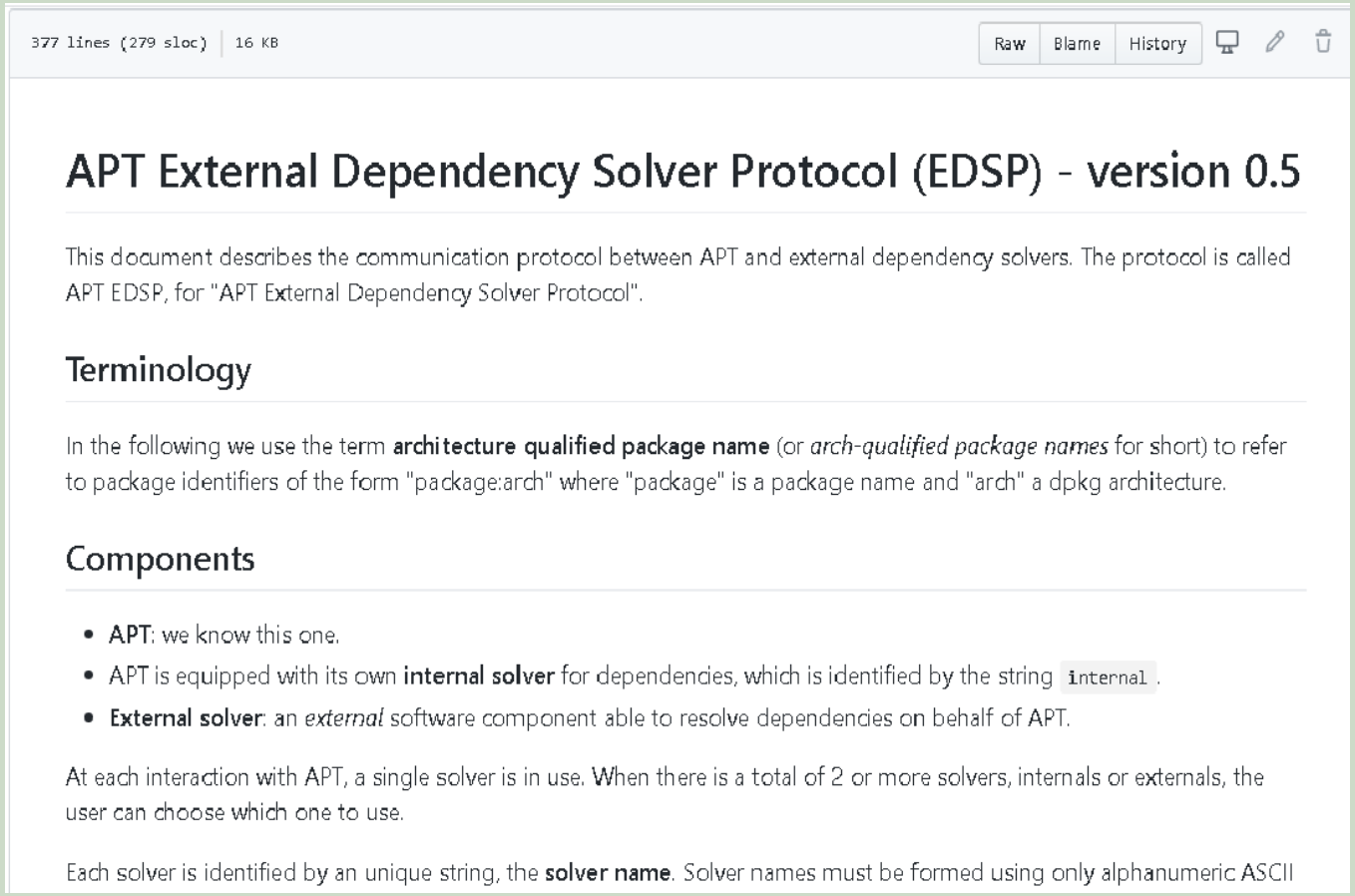
Package dependency solver, using minisat with package management specifics:

- Choose packages with highest versions during unit resolution
- Keep as many installed packages as possible
- Record decision chain for solution introspections

Created by Michael Schroeder (Suse) during a hack week on June 2007. Now is a mature library being used in Zypper (Suse) and DNF (RedHat), among others.

Delegate package dependency to the domain experts, focus on non dependency management features

What about Debian?



377 lines (279 sloc) | 16 KB

Raw Blame History

APT External Dependency Solver Protocol (EDSP) - version 0.5

This document describes the communication protocol between APT and external dependency solvers. The protocol is called APT EDSP, for "APT External Dependency Solver Protocol".

Terminology

In the following we use the term **architecture qualified package name** (or *arch-qualified package names* for short) to refer to package identifiers of the form "package:arch" where "package" is a package name and "arch" a dpkg architecture.

Components

- **APT**: we know this one.
- APT is equipped with its own **internal solver** for dependencies, which is identified by the string `internal`.
- **External solver**: an *external* software component able to resolve dependencies on behalf of APT.

At each interaction with APT, a single solver is in use. When there is a total of 2 or more solvers, internals or externals, the user can choose which one to use.

Each solver is identified by an unique string, the **solver name**. Solver names must be formed using only alphanumeric ASCII

apt-get install apt-cudf

apt-get -s --solver aspcud install gdb

SAT Solvers

Software engines that use heuristics to solve the Boolean Satisfiability Problem (NP Complete). Widely used in optimization problem (Electronic Design Automation, Dependency Solving, Routing FPGAs, etc) with millions of variables.

Different types of heuristics used (Divide and Conquer, Backtracking) for many types of different SAT solvers

$(\neg a \vee b) \wedge (\neg a \vee c)$ if $a = T$

Satsifiable!

$b = T$

$c = T$

Unit Propagation: If an unsatisfied clause has only one variable unassigned, it must be assigned the value that would make the clause true.

SAT Solvers on Package Managers

1) Translate package relationships into disjunctive Boolean clauses (only ORs).

a depends b, c	(¬a ∨ b) ∧ (¬a ∨ c)
a conflicts b	(¬a ∨ ¬b)
a depends b (b1, b2)	(¬a ∨ b1 ∨ b2) ∧ (¬b1 ∨ ¬b2)

2) Translate jobs into Boolean clauses:

Install a :	a
Remove b:	¬b

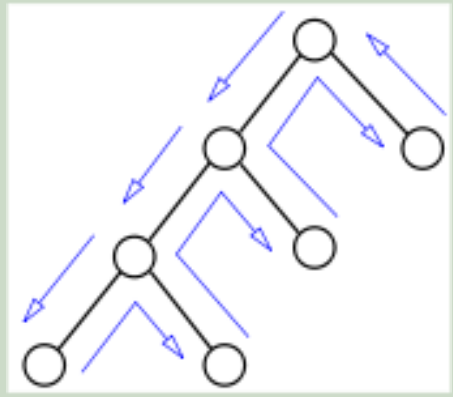
3) Solve

SAT: return transactions to execute

UNSAT: report issues

4) Execute Transactions

Conflict Driven Clause Learning (CDCL)



1. Apply Unit Propagation. If a conflict is found return UNSAT
2. Initialize the decision level to 0
3. While there are unassigned variables
    1. Select an unassigned variable using package manager heuristics and assign it a value. Remember this value
    2. Increment the decision level
    3. Apply Unit Propagation. Keep track of the resulting assignments in an “implication graph”
    4. If a conflict is found:
        1. Find the cut in the “implication graph” that resulted in the conflict
        2. Add a new clause that is the negation of the assignments leading to the conflict
        3. Backtrack to the decision level where the first-assigned variable leading to the conflict was assigned

Example

```
import os
import opk, cfg, opkgcl

opk.regress_init()

o = opk.OpkGroup()
o.add(Package="a", Depends="x")
o.add(Package="b", Conflicts="d", Provides="x")
o.add(Package="c", Provides="x")
o.add(Package="d")
o.write_opk()
o.write_list()
opkgcl.update()


opkgcl.install("d")
opkgcl.install("a")
```

Expected Result:

- Install c
- Install a

Solver notation:

(¬a ∨ b ∨ c) ∧ (¬b ∨ ¬d) ∧ (a)

SAT Expression: (¬a ∨ b ∨ c) ∧ (¬b ∨ ¬d) ∧ (a)

1) Unit Propagation (a=T)
   (F ∨ b ∨ c) ∧ (¬b ∨ ¬d) ∧ (T)

2) Select a variable to set. d=T (prefer already installed packages)
   (F ∨ b ∨ c) ∧ (¬b ∨ F) ∧ (T)

3) Unit Propagation (b=F)
   (F ∨ F ∨ c) ∧ (T ∨ F) ∧ (T)

4) Unit Propagation (c=T)
   (F ∨ F ∨ T) ∧ (T ∨ F) ∧ (T)

Satisfiable!

a=T, b=F, c=T, d=T

Install a
Install c

Upgrades

```
o = opk.OpkGro
o.add(Package=
o.add(Package=
o.write_opk()
o.write_list()
opkgcl.update(
opkgcl.install

o.add(Package=
o.add(Package=
o.add(Package=
o.write_opk()
o.write_list()
opkgcl.update()
opkgcl.upgrade()
```

```
> ./configure –with-libsolv ...
```

2.0)

3.0)

a(2.0), b(1.0)	a(2.0), b(2.0)	a(2.0), b(1.0)

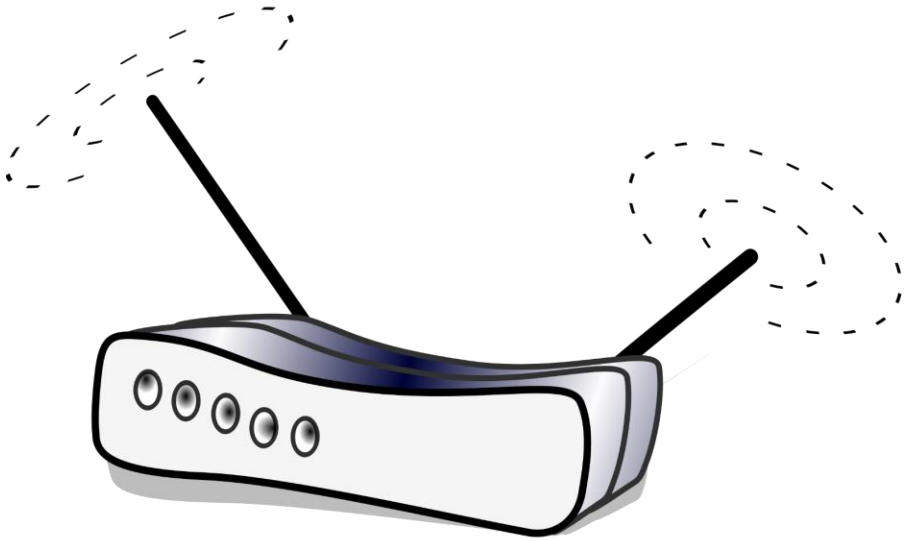
Future Work

- Logo
- Build system
- Error handling & reporting
- Clean up opkg-utils
- Multilib support
- Opkg single source for OE/OpenWRT

OpenWRT

Fork of OE opkg 0.2.7 with many patches on top

- Cherry-pick from OE opkg
- Effort to unify on December 2016 (Florin Gherendi), but was not adopted by OpenWRT
- Duplication of efforts



Future Work

SECURITY WEEK
INTERNET AND ENTERPRISE SECURITY NEWS, INSIGHTS & ANALYSIS
Subscribe | 2019 CISO Forum, Presented by Intel | ICS Cyber Security Conference | Contact

Malware & Threats | Cybercrime | Mobile & Wireless | Risk & Compliance | Security Architecture | Security Strategy | SCADA / ICS | IoT Security
Mobile Security | Wireless Security

Home > Vulnerabilities

Remote Code Execution Vulnerability Patched in OpenWrt

By Ionut Arghire on March 26, 2020

Share | Tweet | Recommend 0 | RSS

A vulnerability that OpenWrt addressed in its opkg fork could have been exploited for the remote execution of arbitrary code.

A free, Linux-based embedded platform, OpenWrt has been specifically tailored for network routers and is used on millions of devices worldwide. Opkg is a package management system forked from ipkg, and is intended for use on embedded devices.

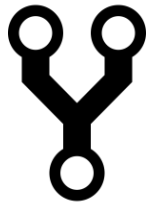
Tracked as CVE-2020-7982, the addressed issue resides in the package list parse logic of opkg, which did not perform the necessary checks on downloaded .ipk artifacts.

"Due to the fact that opkg on OpenWrt runs as root and has write access to the entire

Google™ Custom Search | Search

GET THE DAILY BRIEFING
BRIEFING
Business Email Address | SUBSCRIBE

http://www.securityweek.com



OpenWrt
Wireless Freedom

The End