Design and Implementation of NetBSD Base System Package Distribution Service

Ken'ichi Fukamachi
Chitose Institute of Science and Technology
k-fukama@photon.chitose.ac.jp

Yuuki Enomoto
Cybertrust Japan Co., Ltd.
yuki.enomoto@cybertrust.co.jp

Abstract

We consider that Unix operating system should be built on fine granular small parts (packages) to improve the system maintenance. It is expected that it enables speedy security update, system update tracking in detail, easy replacement and rollback of specific parts.

We have implemented and run a new service to distribute modular base system userland for NetBSD. We generate the least amount of modular base packages by using `basepkg.sh`. It splits NetBSD daily binaries into 1000 over packages based on `syspkgs` meta-data and ident comparison within the binaries. This scheme drastically reduces the processing time to realize operations within practical time.

Our system have shown that granular update system and service can be implemented and operational under breakdown approach. NetBSD users can maintain NetBSD base system in more granular way with fine update history and build an arbitrary system from the NetBSD minimal installation.

1 Introduction

Historically, before the use of Internet leased lines was popular in 1990s, operating system (OS) had been managed on one source tree and the source tree set has been distributed. The typical example is BSD Unix. It has been developed in its own source tree including kernel, general commands, configuration files, and manuals. BSD Unix distinguishes between the official distribution and 3rd party software.

Another example is Linux distribution. It does not distinct its own base system from third-party software. It assembles a lot of small packages which are created and maintained by many different authors. To manage the whole system, it is inevitable to develop software such as `apt` for Debian GNU/Linux and `yum` (`dnf` in the future) for Red Hat Enterprise Linux.

Aside from the origin of development styles, OS built on fine granular small parts must be preferable to improve the system maintenance. It is expected that it enables speedy security update, system update tracking in detail, easy replacement and rollback of specific parts.

To reconstitute NetBSD base system to be comprised of small parts, we have implemented software (Chapter 4) to dispose the base system to 1000 over parts and run a service (Chapter 5) to distribute them with our experimental client (Chapter 6). In this paper we call our strategy `breakdown` approach in contrast to the bottom up one of Linux distribution.

The rest of this paper is organized as follows. We define terms in Chapter 2. We introduce components of the whole service in Chapter 3. The details of each component are described in Chapter 4, 5 and 6. We discuss several remaining issues in Chapter 7.

2 Terms

The term "package" implies both 3rd party software management and a kind of a container. The usage differs from OS to OS. We need to clarify the terms "base system" and "package". In this paper, we use

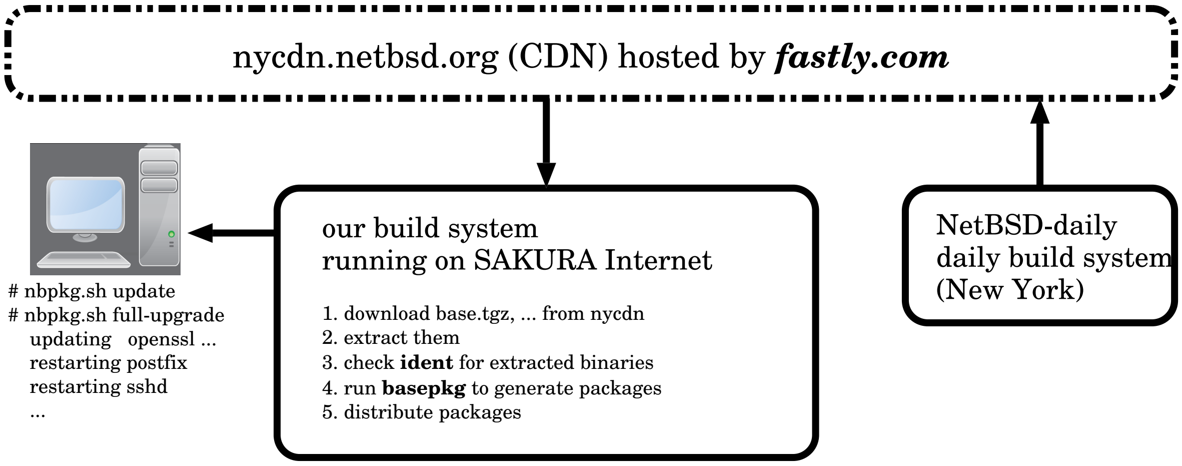

Figure 1: Overview of NetBSD base system package distribution service. It generates base packages by using `basepkg` and distributes them. `nbpkg.sh` client demonstrates updating and restarting.

the term “package” as a container by default.

Linux distributions consider the whole system consists of packages but BSD Unix(s) distinguish between the base system and 3rd party software. BSD Unix(s) consider that the whole system consists of the base system and 3rd party software.

“base system” implies a set of programs officially maintained and distributed by the project. In almost cases, the OS base system distribution is divided by roles to a set of tarballs (which extension is known as ”.tgz”) such as ”base.tgz” (mandatory for the operating system), ”comp.tgz” (compiler tools), ”man.tgz” (manuals) and so on. BSD Unix base system is composed of a set of 10 or more tarballs.

In the BSD Unix, we manage each 3rd party software as a “package”. “package” itself implies a container which consists of software, documentation, configuration files and this package’s meta data required to operate in installation and de-installation. We also call the 3rd party software system “package”. Each BSD Unix project provides the package system such as `pkgsrc` (NetBSD), `ports` (FreeBSD and OpenBSD) and so on. Users can easily handle the package by using the management system.

3 Components of NetBSD Base System Package Distribution Service

We have implemented and been running a new service to distribute modular base system userland for NetBSD (Figure 1). This distribution system consists of three components: (1) `basepkg`[1, 2] (2) `nbpkg-build.sh`[3] (3) `nbpkg.sh`[3].

`basepkg` splits NetBSD base system into 1000 over packages (we call them `base packages`). `basepkg` is a simple almost POSIX compliant shell script built on `pkgsrc`[4] framework and `syspkgs`[5] meta-data. Hence the naming convention of `base package` is same as `syspkgs` one such as `base-crypto-shlib` (shared libraries for cryptography, classified as a mandatory system).

`nbpkg-build.sh` is the top level dispatcher to run `basepkg` for NetBSD binaries downloaded from `nycdn.netbsd.org`. We generate base packages which changes are detected based on ident (RCS Id) comparison. Though community based development does not have powerful computer resources, those measures reduce the work, as a result, our build sys-

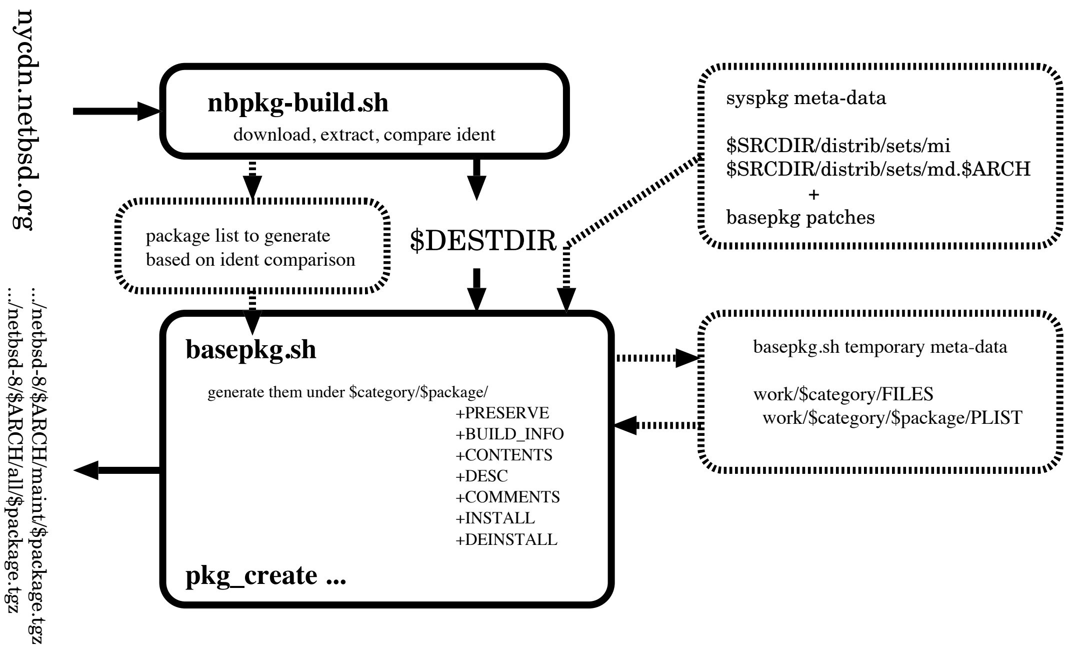
nycdn.netbsd.org
nbpkg-build.sh
download, extract, compare ident
syspkg meta-data
$SRCDIR/distrib/sets/mi
$SRCDIR/distrib/sets/md.$ARCH
+
basepkg patches
package list to generate
based on ident comparison
$DESTDIR
basepkg.sh
generate them under $category/$package/
+PRESERVE
+BUILD_INFO
+CONTENTS
+DESC
+COMMENTS
+INSTALL
+DEINSTALL
pkg_create ...
basepkg.sh temporary meta-data
work/$category/FILES
work/$category/$package/PLIST
.../netbsd-8/$ARCH/maint/$package.tgz
.../netbsd-8/$ARCH/all/$package.tgz

Figure 2: `basepkg` internals

tem, running on low spec VPS[1] , works out daily to provide base packages for NetBSD 8.0 stable branch (62 targets).

`nbpkg.sh` is an experimental client to show our operation model. `basepkg` is built on `pkgsrc` framework so that we can use `pkgsrc` functions as could as possible. `nbpkg.sh` is an extension to `pkgin`[6] (`pkgsrc/pkgtools/pkgin`) which provides `apt/yum/dnf` like functions to maintain the base system more systematically.

4 Basepkg

`basepkg` is a 1200 lines Bourne shell script to split NetBSD base system into 1000 over packages. It consists of meta-data and package build system. The `basepkg` processing (Figure 2) is briefly described below. See the reference[1] for more details.

The meta-data is derived from NetBSD source tree but modified and enhanced by us. The `basepkg` meta-data is based on `syspkgs` one, files in `/usr/src/distrib/sets/lists/`. Each line of the meta-data file contains a set of information (path, syspkgs package name, comments) such as

```
./bin/ls       base-util-root
./bin/rcorder  base-obsolete  obsolete
./bin/rump.dd  base-util-root rump
./usr/bin/cpp  base-util-bin  gcccmds
```

It has been maintained but is inconsistent and contains several bugs. `basepkg` imports the `syspkgs` meta-data and modifies it to fix several bugs and enhances it to support X11.

The actual build process of `basepkg` is running `pkg_*` utilities (`pkgsrc/pkgtools/pkg_install`) to split the base system according to the meta-data.

[1]`bytebench`(`pkgsrc/benchmarks/bytebench`) shows our VPS power is considered to be almost same as a popular home server such as NEC S70 (its CPU is Intel Pentium G6950) on sale in 2011.

```
#   list_arch = amd64 evbarm evbmips evbppc hpcarm i386 sparc64 xen
#        arch = amd64
#      branch = netbsd-8
#    url_base = http://nycdn.netbsd.org/pub/NetBSD-daily/netbsd-8/
#  build_nyid = 201811180430Z
#  build_date = 20181118
#  build_url  = $url_base/$build_nyid/$arch/binary/sets/
for arch in $list_arch
do
    nbdist_download $arch $build_url
    nbdist_extract  $arch
    nbdist_check_ident_changes ...
    if "X$is_ident_changes_found" = "Xyes"; then
        nbpkg_build_gen_basepkg_conf    $arch $branch ...

        # (1) maint mode
        nbpkg_build_run_basepkg         $arch $branch "maint"
        nbpkg_release_basepkg_packages  $arch $branch "maint"

        # (2) all mode
        nbpkg_build_run_basepkg         $arch $branch   "all"
        nbpkg_release_basepkg_packages  $arch $branch   "all"
    fi
done
```

Figure 3: Concept of the `nbpkg-build.sh` main loop

`basepkg` is built on `pkgsrc` framework. It is good to avoid `reinventing the wheel`. `basepkg`-generated package format is same as pkgsrc one, so that we can use features `pkgsrc` provides. For example, we can use `pkg_add/pkg_delete` to add/remove base packages. Moreover, we can use more powerful utility such as `pkgin` which provides `apt/yum/dnf` like functions for pkgsrc. By using `pkg_summary(5)`, `pkgin` resolves associated dependencies among base packages and provides smart operations for installation, removal and upgrade of base packages.

5 Base Package Generation and Distribution System

5.1 Overview

`basepkg` is just a script to split NetBSD base system to 1000 over packages. To provide the NetBSD base system upgrade service, we need to design, implement a package generation system (`nbpkg-build.sh`) described in this section and run the web service at `http://basepkg.netbsd.fml.org`. `nbpkg-build.sh` is the top level dispatcher to run `basepkg` to split NetBSD binaries. Both `nbpkg-build.sh` and the web runs on **SAKURA**

```
# tar -C $DEST_DIR -zxpf $DIST_DIR/*.tgz
# find $DEST_DIR -exec /usr/bin/ident {} \;        > $IDENT_NEW
# diff $IDENT_OLD $IDENT_NEW                        |
  sh $CONVERT_TO_PACKAGE_NAME_VIA_SYSPKGS_DATA      > $TARGETS
```

Figure 4: Brief description of logic on how to compare ident information: NetBSD binaries e.g. base.tgz, etc.tgz, ... are downloaded to $DIST_DIR and are extracted at $DEST_DIR. The final output $TARGETS file contains only `syspkgs` names e.g. base-sysutil-bin. $TARGETS file is passed to `basepkg.sh` as an optional argument.

Internet VPS(v3)[7][2].

`nbpkg-build.sh` (1) downloads binaries from `nycdn.netbsd.org` (fastly CDN) (2) extracts them (3) checks ident within the extracted NetBSD base system binaries and (4) runs `basepkg` to generate base packages.

5.2 Download and Extract NetBSD Binaries

Firstly we need to prepare NetBSD base system binaries to split.

It is preferable to build the binary from the source code to avoid versioning problem (see the Section 7) but it requires a lot of machine resources (both CPU power and storage). To process them within practical time, we decided not to build NetBSD from the source code but download NetBSD binaries provided as NetBSD-daily (what we call **daily build**). The daily build system runs at Columbia University in New York[8] but we can download the binaries via `nycdn.netbsd.org` hosted on `fastly.com` CDN (contents delivery network). Also, our build machine (`basepkg.netbsd.fml.org` running on SAKURA Internet VPS[7]) has enough bandwidth to Internet. Hence we can download binaries enough fast.

It is observed that the time to download and extract them requires not more than 300 seconds (the average is 200 seconds for downloading and 70 seconds for extraction) per one distribution e.g. amd64 on netbsd-8 branch. If we build NetBSD from the source on this host, it requires at least 1200 seconds even in the case of running “build.sh -u release” (18000 seconds in the case of running “build.sh release”). This process contributes to the processing time reduction and does not consume the storage temporarily required for build process.

[2]3 CORES, 3GB MEMORY, 200GB HDD

5.3 Base Package Generation Strategy

We have implemented the following two plans to consider which is proper in operating NetBSD upgrade service using base packages.

The plan A proposed in AsiaBSDCon2018[1] is that we generate all available base packages daily and determine which base package should be installed according to a given configuration based on NetBSD security advisory. In this case, the implementation of the package generation system is simple but we need huge machine resources. To make matters worse, it is not automatic since somebody needs to edit the configuration.

The plan B is opposite to the plan A. We have only to generate the least amount of base packages having the change after the major release. We can find changes by tracing the ident information in binaries, To implement it, we need to modify both `nbpkg-build.sh` and `basepkg` but this modification can drastically reduce the amount of work in generation base packages. In addition it is important that this mechanism runs automatically.

Hence we decided to use and run the service based on the plan B. We define the basis of the ident comparison is the most recent major release, NetBSD 8.0 release now.

5.4 Ident Based Comparison

The ident based comparison is straightforward (Figure 4). We check all files on the extracted NetBSD base system and compare the ident information with the previous one to determine which base packages are changed and so should be re-generated. The list of base packages which should be re-generated is passed to the `basepkg` as the configuration.

In this case, `basepkg` has only to build a minimum of packages. Both the saving of targets and downloading via CDN (Section 5.2) drastically reduces the processing time. Currently our processing time per `arch` per `branch` is about 1300 seconds on average. It includes downloading, extraction and `basepkg` which runs two times to generate packages for two modes described below (Section 5.5). It is estimated that we can process 62 targets of netbsd-8 branch within one day. We try to process Tier 1[9] targets twice a day to keep the base package up-to-date as could as possible, but Tier 2 ones once a day.

5.5 Package Dependency Problems

Our breakdown approach introduces a new kind of difficulty on that the basis of comparison is arbitrary. Consider the following cases: (1) keep the system up-to-date mainly for security update (2) build up an arbitrary system from the minimum one. They are similar but the dependencies among base packages differ in essential. In the case 1, we need only packages having the change after the major release. In the case 2, we need to prepare both all base packages for the major release and packages having the change after the major release. The package dependencies in two cases differ since packages in the case 2 demands major release packages if needed. Since the coexistence of different dependency within one package is difficult, our system distributes two kinds of base packages with different dependency at different URLs.

```
[at .../netbsd-8/$ARCH/maint/]
SHA512
base-cron-bin-8.0.20181129.tgz
base-ext2fs-root-8.0.20181129.tgz
base-ipf-bin-8.0.20190119.tgz
...

[at .../netbsd-8/$ARCH/all/]
SHA512
base-adosfs-root-8.0.20180717.tgz
base-amd-bin-8.0.20180717.tgz
base-amd-examples-8.0.20180717.tgz
base-amd-shlib-8.0.20180717.tgz
...
base-cron-bin-8.0.20180717.tgz
base-cron-bin-8.0.20181129.tgz
base-cron-root-8.0.20180717.tgz
...
```

Figure 5: Examples of packaegs for both maint and all modes. The path of "all" mode contains base packages for 8.0 release (suffix .20180717) and packages (e.g. suffix .20181129 and .20190119) which changed after 8.0 release. Howeve the path of "maint" mode contains only changed packages.

In the case 1, we call it `maint` mode, we assume a scenario that users have full-installed NetBSD (e.g. NetBSD 8.0) initially and keep it as the latest NetBSD 8.0 stable (netbsd-8 branch). In this case, we have only to install (overwrite) all base packages having the change after the major release. This update process is able to work automatically. It is possible that each user can install the specific package manually if needed.

In the case 2, we call it `all` mode, it is necessary to think about the possibility that we need to install packages which do not exist on the system. Consider the minimal NetBSD which consists of only base.tgz and etc.tgz. If we newly want to install a C compiler (e.g. `/usr/bin/cc`), which does not exist now, we only have to run ``‘‘pkg_add comp-c-bin’’``. If `comp-c-bin` has no difference from 8.0 release, there is no `comp-c-bin` package in the `maint` mode (case

1). In the case 2, we need to obtain and install 8.0 release `comp-c-bin` base package which name is `comp-c-bin-8.0.20180717.tgz`. Hence in this case, for the possibility building any system from a scratch, we need to prepare both all base packages for the major release and packages having the change after the major release.

The details of package dependency are as follows. For example, let autopsy the dependency of `base-sysutil-bin-8.0.20190119.tgz` tarball. `+CONTENTS` file (`pkgsrc` meta-data) in the tarball in `maint` mode is defined as

```
@pkgdep base-sys-shlib>=8.0.20181129
```

but `+CONTENTS` in `all` mode contains

```
@pkgdep base-sys-root>=8.0.20180717
@pkgdep base-sys-shlib>=8.0.20181129
@pkgdep base-sys-usr>=8.0.20180717
```

where the suffix `8.0.20180717` implies the NetBSD 8.0 release we define artificially since NetBSD 8.0 was released on July 17, 2018. Due to this artificial dependency

```
@pkgdep $package>=8.0.20180717
```

a package which does not exist can be installed automatically.

6 Experimental Client

6.1 Overview

We provide an experimental client `nbpkg.sh` to show our operation model. As mentioned above, `basepkg`-generated package format is same as pkgsrc one, so that we can use full features `pkgsrc` provides. We design our client as a wrapper of `pkgin` to provide integrated service like apt/yum/dnf on Linux distribution.

Our current experimental client uses `/var/db/pkg/` directory for package registration which `pkgsrc` uses originally. Hence `/var/db/pkg/` holds installed package data for both `pkgsrc` and `basepkg`. It is considered to be easy to distinguish `basepkg` packages from `pkgsrc` ones by names, since the naming convention for `pkgsrc` and `basepkg` are very different.

6.2 Usage

`nbpkg.sh` usage is similar to `apt` command. See a demonstration running `update` and `full-upgrade` commands in Figure 6.

At the first time, `nbpkg.sh` checks the environment and install mandatory package management utilities (`pkgsrc/pkgtools/`) such as `pkg_install` (`pkgsrc/pkgtools/pkg_install`) and `pkgin` by default if they are not found.

` ``nbpkg.sh update'' ` updates the database (`pkg_summary(5)`) from a remote repository defined by the environmental variable `PKG_PATH` which is hard-coded in `nbpkg.sh`.

` ``nbpkg.sh upgrade'' ` is reserved, not recommended currently.

We can use ` ``nbpkg.sh full-upgrade'' ` to keep the system up-to-date, the latest one on stable branch, automatically. By default we assume `maint` mode operation described above (Section 5.5). ` ``nbpkg.sh full-upgrade'' ` upgrades packages specified by a file `pkg_list2upgrade` in `PKG_PATH` e.g. `http://basepkg.netbsd.fml.org/pub/NetBSD/basepkg/netbsd-8/$ARCH/maint/pkg_list2upgrade`. `pkg_list2upgrade` describes a list of all packages having the change after the major release. It is prepared and updated by our system `nbpkg-build.sh`.

We can install or remove arbitrary packages manually. We assume `all` mode for such operation. For example, to install a C compiler `/usr/bin/cc`, we run ` ``nbpkg.sh -a install comp-c-bin'' ` where `-a` option implies all mode (Section 5.5). It resolves the dependecies so that it installs all packages required to use a C compiler. The packages to install are the latest stable one if exists but the last major release (8.0 release now) ones if the package has no change after release.

6.3 Extension

Our client `nbpkg.sh` is not just a wrapper of `pkgin`. It is extended to support (1) fool-proof function not

```
# nbpkg.sh full-upgrade

Running install with PRE-INSTALL for pkg_install-20180425.
man/man1/pkg_add.1
   ...
Package pkg_install-20180425 registered in /var/db/pkg/pkg_install-20180425
   ...

Running install with PRE-INSTALL for pkgin-0.11.6.
bin/pkgin
man/man1/pkgin.1
   ...
Package pkgin-0.11.6 registered in /var/db/pkg/pkgin-0.11.6
   ...
Requesting http://basepkg.netbsd.fml.org/.../maint/pkg_list2upgrade
100% |***********************************| 435 967.66 KiB/s 00:00 ETA
435 bytes retrieved in 00:00 (608.60 KiB/s)
pkgin import /var/db/nbpkg/pkg_list2upgrade
reading local summary...
processing local summary...
processing remote summary (http://basepkg.netbsd.fml.org/.../maint)
  ... snip ...
downloading pkg_summary.gz: ...
calculating dependencies...done.

29 packages to install:
  base-cron-bin-8.0.20181123 base-ext2fs-root-8.0.20181123
  ... snip ...
  xetc-sys-etc-8.0.20181123

0 to refresh, 0 to upgrade, 29 to install
62M to download, 221M to install

proceed ? [Y/n] y
downloading base-cron-bin-8.0.20181123.tgz ...
    ...
installing base-cron-bin-8.0.20181123...
    ...
pkg_install warnings: 0, errors: 0
reading local summary...
processing local summary...
    ...
marking xetc-sys-etc-8.0.20181123 as non auto-removable
```

Figure 6: Example of upgrading the system

to overwrite /etc by accident (2) alias function for us to handle user friendly package names.

To implement the fool proof, `etc-*` packages was removed from `pkg_list2upgrade`. Hence automatic upgrade for `etc-*` packages does not work but you can use explicitly running `nbpkg.sh install etc-*` to update files under `/etc/` (We trust each user for such critical actions).

We support `nbpkg.sh` alias function since `syspkgs` naming convention is too far from the usual convension. For example, `syspkgs` name `base-crypto-shlib` contains `libcrypto.a` and `libssl.a` but generally we call them `openssl shared libraries`. `base-crypto-bin` includes `openssl` command. By our alias support, we can use ``` ``nbpgk.sh upgrade openssl'' ``` to update both openssl libraries and commands. Currently the following aliases are defined.

```
alias           syspkgs-package-name
------------------------------------
libcrypto.so    base-crypto-shlib
libssl.so       base-crypto-shlib
openssl         base-crypto-shlib
openssl         base-crypto-bin
openssh         base-secsh-bin
named           base-bind-bin
bind            base-bind-bin
postfix         base-postfix-bin
```

We can install `openssh` by running ``` ``nbpkg.sh install openssh'' ``` instead of ``` ``nbpkg.sh install base-secsh-bin'' ```. This definition is hard-coded currently, it is an issue to resolve in the future.

Theoretically we can rollback the specific package manually. After checking package registration logs in `/var/db/pkg`, we run `nbpkg.sh` to remove the installed package and enforce the installation of the previous one.

7 Discussion

We have resolved several issues addressed in AsiaBSDCon2018[1]. Especially the use of `nycdn.netbsd.org` and ident based comparison contributes to the huge reduction of processing time. However it remains a few difficult issues such as base package versioning and dependency discussed in the Section 5.5. We do not discuss `syspkgs` meta-data maintenance problem such as validation of the granularity since these topics are beyond our 3rd-party development scheme.

The versioning problem implies we distinguish base packages by the date suffix not semantic versioning[10] x.y.z e.g. 1.0.0, 1.0.1 and so on.

In the case of the bottom up approach such as Linux distribution, OS assembles a lot of small packages which are created and maintained by many different authors. Each package has each author and versioning e.g. etc-passwd-1.0.1, bin-ls-2.0 and timezone-20190101. The versioning are inconsistent but meaningful in some sense.

In the case of BSD Unix, the whole system is maintained uniformly so that the version is **NetBSD 8.0** for not only the whole system but also all parts in it. Paradoxically we cannot determine the precise version for each granular base package since the author or maintainer is ambiguous for each small parts. For example, consider `etc-sys-etc` package which contains password related files such as `/etc/passwd`, `/etc/master.passwd` and so on. We can assign etc-sys-etc-8.0.0 for NetBSD 8.0 release, but we cannot automatically assign etc-sys-etc-8.0.1 when a part of the `etc-sys-etc` package e.g. `/etc/passwd.conf` is updated on netbsd-8 branch. In addition, NetBSD daily build does not consider the update details and generates the whole NetBSD system on a daily basis. Hence, especially in the case of our system, it is practical to assign etc-sys-etc-8.0.YYYYMMDD e.g. etc-sys-etc-8.0.20180717 not etc-sys-etc-8.0.0.

The date based naming convention may introduce the following problem. If the base package generation does not end within one day, package names for the same source changes may be different among architectures e.g.

```
amd64/all/base-sys-shlib-8.0.20190101.tgz
   ...
zaurus/all/base-sys-shlib-8.0.20190102.tgz
```

However it is considered to be enough practical in order to keep the system up-to-date. We always have only to install the latest base packages irrespective of the precise package version name since the versioning is consistent within each branch and architecture. Hence the versioning problem must be trivial from the point of practical or operational view.

8 Conclusion

We reorganize NetBSD userland by breakdown approach and run the base package distribution service for NetBSD users.

Our build scheme is based on the use of NetBSD daily binaries via CDN and ident based comparison to generate the least number of base packages. It enables practical time operations.

Our client is useful to maintain the base system in more granular way and build an arbitrary NetBSD system from the minimum. Most importantly, we can upgrade NetBSD with detailed history data stored under `/var/db/pkg/`. We can trace the update details on a daily basis so that we can rollback if needed.

Our approach introduces another package dependency problem but it is trivial from the point of practical view. Our system must be beneficial for NetBSD users.

References

[1] Yuuki Enomoto and Ken'ichi Fukamachi. Design, Implementation and Operation of NetBSD Base System Packaging. *AsiaBSDCon2018 Proceedings*, pp. 21–31, 2018.

[2] Yuuki Enomoto. `basepkg`. `https://github.com/user340/basepkg`. (accessed 2018-12-31).

[3] Ken'ichi Fukamachi. NetBSD modular userland. `https://github.com/fmlorg/netbsd-modular-userland`. (accessed 2018-12-31).

[4] The NetBSD Project. pkgsrc. `https://www.pkgsrc.org`, 1998. (accessed 2019-02-22).

[5] The NetBSD Project. syspkgs. `http://wiki.netbsd.org/projects/project/syspkgs`, 2015. (accessed 2018-12-31).

[6] Emile 'iMil' Heitor. pkgin, a binary package manager for pkgsrc. `https://pkgin.net`. (accessed 2019-02-22).

[7] Sakura Intenet. Sakura Internet VPS Service. `https://vps.sakura.ad.jp/`. (accessed 2019-02-22).

[8] The NetBSD Project. admin. `https://wiki.netbsd.org/users/spz/admin/`. (accessed 2019-02-22).

[9] The NetBSD Project. Platforms Supported by NetBSD. `https://www.netbsd.org/ports/`. (accessed 2019-02-22).

[10] Tom Preston-Werner. Semantic Versioning 2.0.0. `https://semver.org/`. (accessed 2019-02-22).