

Software for NetFlow Monitoring Adapter

Pavel Čeleda, Milan Kováčik, Radek Krejčí, Jaroslav
Kysela, Petr Špringl

December 20, 2005

1 Abstract

The objective of this technical report is to describe software activities and tools developed during the work on the NetFlow probe. The development process follows hardware software co-design principles. We are using the hardware advantages to speed up time critical parts during IP flows monitoring. The software covers state of the art problems in the field of generation and export of NetFlow v9 datagrams.

2 Introduction

The development started as part of the GN2 project with the objective to create an autonomous network monitoring probe capable of generating data in the NetFlow version 9 format. At the beginning the work was hardware oriented. The COMBO6 hardware accelerator with interface card (COMBO-4MTX or COMBO-4SFP) form the basis of our hardware [CoHW]. The hardware behaves like a programable NIC (*Network Interface Card*). The program for the NIC is written in VHDL (*VHSIC Hardware Description Language*) and we call them firmware. Firmware is software that is embedded in a hardware device. Line of partition between the hardware and software world builds the communication interface (PCI bus) which provides data to upper software layers in PC.

The hardware description of NetFlow probe can be find in [Zad04], [Zad05]. Technical report is focusing on the software development done during 2005 which joined the hardware development. The main aim was to support reading of flow records from the probe and the generation of NetFlow v9 datagrams [RFC3954]. Netflow data are typically acquired and exported by IP routers. Figure 1 1 shows installation overview of the hardware accelerated NetFlow probe which we are using for IP flow monitoring.

The NetFlow probe works as a T-splitter and is fully transparent for network traffic. IP flows are monitored by NIC firmware, preprocessed and forwarded to

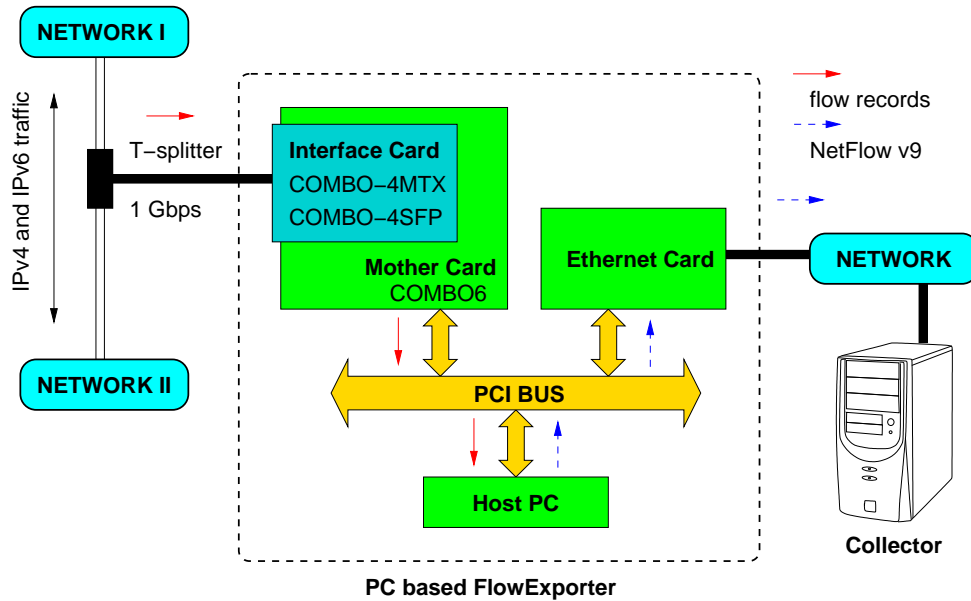


Figure 1: NetFlow probe overview

upper software layers. User space application *flowexporter(1)* reads data from hardware, produces UDP NetFlow v9 datagrams and resends them to a collector.

3 NetFlow Linux driver for COMBO6

The NetFlow Linux driver for COMBO6 supports latest 2.4 and 2.6 Linux kernels. The configuration of the driver sources is standalone, but the configured linux kernel source tree must be present to proceed a successful compilation. The configuration script is available for the user to make the driver configuration easy.

The NetFlow driver defines special *ioctl()* syscalls which are managing the device enumeration, subscription and the ring buffer management. The ring buffer contains the netflow records and queues for all connected applications. The contents of netflow records is shared among all applications (with the read-only access) and mapped into the user space using the *mmap()* syscall. The record descriptors (containing status and pointer to contents) are also available using the *mmap()* syscall. All these data are read-only.

The *mmap()* access for data is used to eliminate extra copy rather than the standard *read()* syscall. This method is called zero-copy, because applications have direct access to data from driver. Because modern CPU have MMU (*Memory Management Unit*), these memory pages are protected (read-only access for applications). The driver only maintains the queues of waiting netflow records

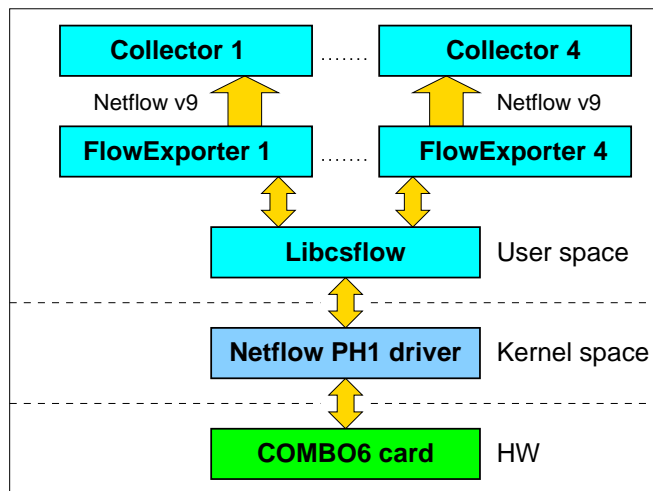


Figure 2: NetFlow probe software layers

for applications. Also, putting the data arbiter code into kernel makes the data flow robust, because if one application crashes, the other applications are not affected.

The ring buffer is allocated when the driver is loaded. The ring buffer size can be specified via kernel module parameter or command line parameter to kernel at boot. We can eventually use another external way to modify the ring buffer size at runtime (use the proc filesystem etc.).

3.1 Data flow control (from the application view)

An application will start/stop capturing of data using an dedicated ioctl. Note that other applications might be running after stop, thus packets can be still overwritten in the ring buffer even if the application stops the capturing. The real hardware stop occurs when all applications are disconnected from the driver. To determine the first packet in or get a next packet from the FIFO (*First In First Out*) list maintained for the given application, a lock-next ioctl is used. At the end of communication, the remaining lock should be unlocked with an unlock ioctl (note that the lock-next ioctl will unlock the current packet automatically, so there is no need to call always a lock-next -> unlock sequence; the standard use is: open, subscribe, lock-next, lock-next, . . . , unlock, close). The access to unlocked areas is permitted, but not very useful, because driver or hardware might overwrite data at any time.

3.2 Library libcsflow

The libcsflow library was designed as a middle layer between netflow applications and driver. It hides specific syscalls and provides the standard interface

in C language. The further improvements will be to move more common code like the netflow record parsing to this library from netflow applications.

3.2.1 The C interface

```
/*
 * Device handler management
 */

int    csflow_open(csflow_device_t **dev, const int card, const int interface);
int    csflow_close(csflow_device_t **dev);
int    csflow_fd(csflow_device_t *dev, int *fd, short *events);

/*
 * Start / stop functions
 */

int    csflow_start(csflow_device_t *dev);
int    csflow_stop(csflow_device_t *dev);

/*
 * Flow record / timestamp management
 */

int    csflow_bufinfo(csflow_device_t *dev, u_int32_t *entries, u_int32_t *entry_size);
int    csflow_lock(csflow_device_t *dev, u_int32_t count, u_int32_t *locked);
int    csflow_getptr(csflow_device_t *dev, u_int32_t idx, void **ptr);
u_int64_t csflow_tstamp32_uptime(csflow_device_t *dev, u_int32_t tstamp32);
u_int64_t csflow_get_init_uptime(csflow_device_t *dev);
u_int64_t csflow_get_current_uptime(csflow_device_t *dev);
```

4 Hardware start-up

It is necessary to load and initialize the NetFlow probe firmware into the COMBO cards (mother and interface card) after kernel drivers are loaded. It means boot *.mcs (*Intel PROM format*) files which are result of VHDL compilation into FPGAs (*Field-Programmable Gate Array*). *csboot(1)* is used for it.

Now the firmware is loaded into COMBO cards but it doesn't run. It must be initialized and started up. Firmware initialization contains several activities:

- UHDRV initialization,
- HGEN initialization,
- HFE program loading.

This operations can be done by *netflowctl(1)* tool.

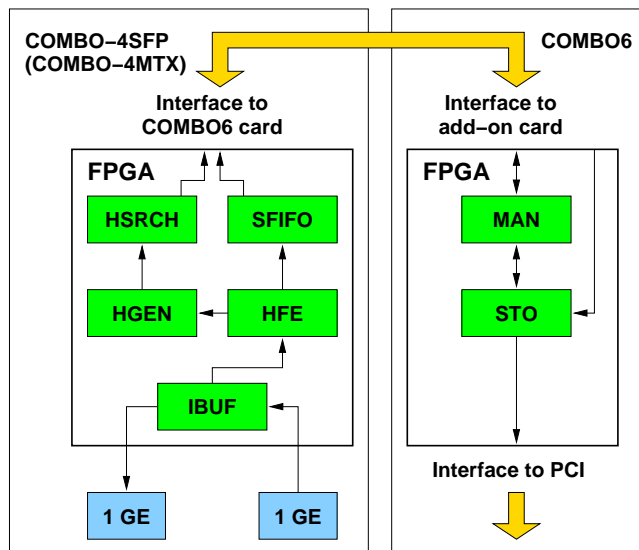


Figure 3: Combo6 mother card with Combo-4SFP

4.1 UHDRV initialization

UHDRV (*Unified Header Driver*) is a hardware unit which is an interface between UH_FIFO and HGEN_FIFO and STAT_FIFO. UHDRV decides whether incoming record from UH_FIFO is sent for further processing - records with bad CRC (*Cyclic Redundancy Check*) or non IP (*Internet Protocol*) does not continue further. UHDRV chooses which array of incoming records will be masked and it is used for kind of agregation.

It is necessary to write some program to UHDRV which tells it what kind of arrays has to be masked.

4.2 HGEN initialization

HGEN (*Hash Generator*) is a hardware unit for generating hash of flow. The hash function has to be initialized by random 64-bit long value for better usage.

4.3 HFE program loading

HFE (*Header Field Extractor*) is a hardware unit which is intended for analyzing of input packets. It is a processor based on RISC (*Reduced Instruction Set Computer*) architecture controlled by specific instruction set. HFE reads packets data from input buffer, analyses control information in its headers and produces specific data structures.

Before using HFE it is necessary to load program into processor memory. Program controls the generation of specific data structures.

4.4 Configuration of NetFlow probe

The *netflowctl(1)* is used for configuration of NetFlow probe. There are several main properties to set and configure:

- active timeout,
- inactive timeout,
- sample and hold,
- ...

Active timeout: Active timeout is used for releasing flows which last for longer time than the specified timeout.

Inactive timeout: Inactive timeout is used for determining how long the flow should be kept in memory even if the device has not seen any packet of that flow.

Sample and hold: Sampling of incoming packets is the easiest way how to guarantee the measured bandwidth. It helps also to decrease number of new flows during attacks when every incoming packets belongs to new flow.

Sample and hold method is quite similar to input sampling but with following twist. As with ordinary sampling, each packet is sampled with a probability. If a packet is chosen and the flow it belongs to is not in the flow memory, a new item is created. However, after an item is created for a flow, unlike in sampled NetFlow, every subsequent packet belonging to the flow updates the item.

4.5 How to use *netflowctl(1)* for configuration of NetFlow probe

Description of user interface of *netflowctl(1)*.

Firmware initialization.

```
-c init                initialization of UHDRV, loads program for HFE  
                        and initialize HGEN.
```

If you don't want to use default values you can specify it with following parameters.

```

-u udrv_file          file for initialization of UDRV, udrv_file can
                      be generated by gen_mem tool
-g init_value_high   32 high bits of initialization value for HGEN
-i init_value_low    32 low bits of initialization value for HGEN
-e hfe_prog          file with binary program for HFE

```

Inactive timeout can be set to value in seconds by parameter

```
-c inact_timeout -v value -b
```

Active timeout can be set to value in seconds by parameter

```
-c act_timeout -v value -b
```

Sample and hold can be set by parameters

```

-c sample_hold -v "threshold" -s "sampling_rate" -t "type"
  "type" is 0 - for constant sampling,
             1 - for variable sampling

```

Reset the whole design can be done by parameter

```
-c reset
```

and new initialization values can be set by parameters -u, -g, -i, -e else the default values will be used.

4.6 How to configure NetFlow probe

A set of shell scripts can be used to do all necessary operations for configuring and starting up NetFlow probe. Detailed description is available here [NetHowTo]

- *netflow_ph1_modules* - loads kernel modules,
- *netflow_ph1* - hw booting, configuration and *flowexporter(1)* start,
- *netflow_ph1_log* - hw monitoring and logging.

netflow_ph1(1) detects which COMBO cards are installed in computer and according that it will load right firmware into FPGAs (see Figure 5 4, step 2 and 3). The script calls *netflowctl(1)* and initialize firmware and starts it up (step 4). Then active and inactive timeouts are set (step 5) according parameters of the script.

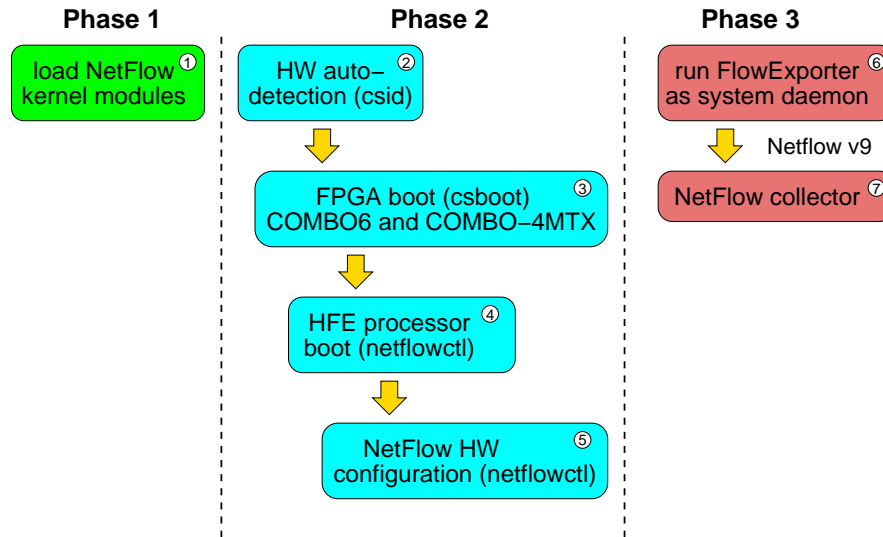


Figure 4: NetFlow probe initialization

```

Usage: netflow_ph1 [options]
-a active timeout in seconds
-c collector
-h print this help
-i inactive timeout in seconds
-p port (0,1)
-r set timeouts and run flowexporter (skips card boot)
  
```

Parameter '-c' is mandatory.

```

Example: netflow_ph1 -p 0 -a 30 -i 10 -c collector.liberouter.org:60000
         netflow_ph1 -p 0 -a 15 -i 5 -c collector.liberouter.org:60000 -r
  
```

Flowexporter is now activated and exporting data to collector.

5 Flowexporter

Flowexporter is a basic NetFlow v9 data exporting tool. It is written in C to support the COMBO6 card with NetFlow extension. It communicates with a collector via the IPv4 protocol using UDP connection.

Command line options:

```

Usage: flowexporter [-dh] [-i card:interface -n host:port -t num]
-d run as a daemon
-h display this help message
  
```

```

-i card:interface    specify netflow card and interface number
-n host:port         send packets to host on port
-t num              set the count of data packets between two template sendings
                   (collector refresh timeout)
-v                  program version

```

5.1 Structure and functions

Flowexporter consists of three basic modules and one supervising module. The basic modules create templates (*flowexporter_configure()*), read records from the COMBO6 card (*flowexporter_system()*), connect to a collector and assemble NetFlow v9 packets (*flowexporter_exporter()*). The supervising module parses command line options, calls certain functions from basic modules and handles some events (errors, system signals).

The algorithm of flowexporter is very simple. Having all modules configured, flowexporter comes into an infinite loop:

```

loop_begin
  if <refreshing timeout> then
    refresh the collector;

    read a netflow record;

  if <exporting packet not too big> then
    copy record into packet;
  else
    begin
      send the packet to the collector;
      create a new exporting packet;
      copy the record into the new packet;
    end
    if <any error occurred or signal cough>
      exit
  fi
loop_end

```

To be a little more exact, we may have a look at the pending flow record inside the flowexporter.

After the *poll()* function succeeded a flow record is picked from the driver's ring buffer. The record is overcasted into a structure describing certain record members so the flowexporter could deal with the flow record.

Due to the differences between members of certain records, eight basic types of records can be distinguished:

- ipv4 : tcp, udp, icmp, other
- ipv6 : tcp, udp, icmp, other

A netflow template is related to each of these types.

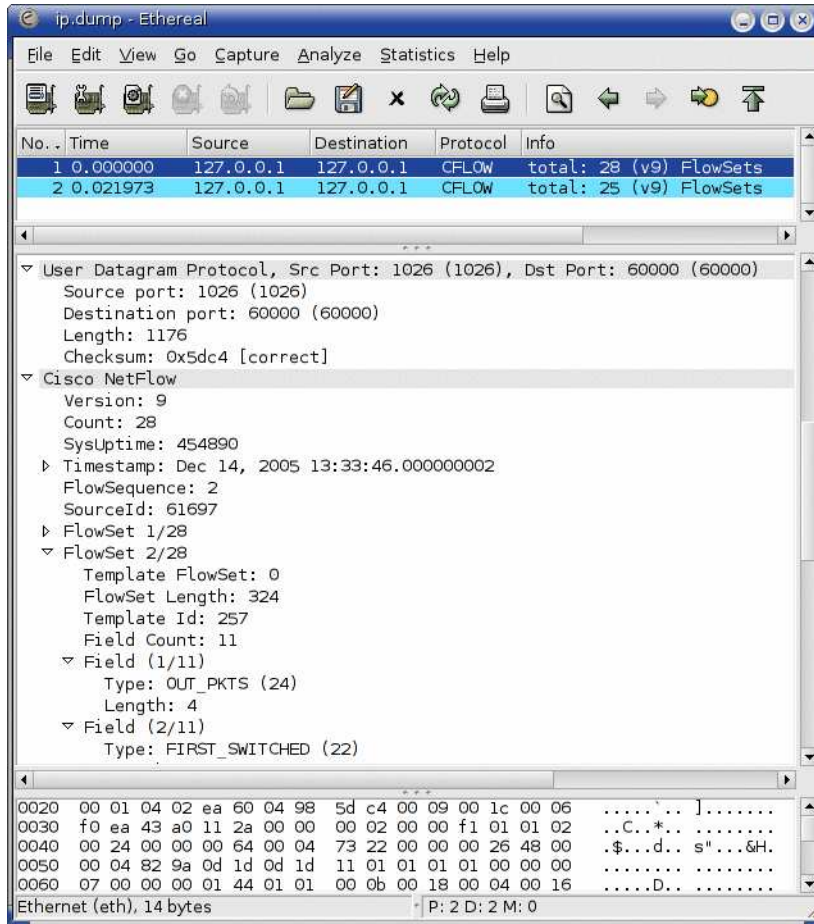


Figure 5: Exported NetFlow v9 UDP datagram

Having been overcasted, the flow record is classified as a member of a certain type. The classification allows the flowexporter to prepare the record for sending to the collector. This is done by removing unused structure members according to the netflow template and putting the record into the right flowset. Flowset is a part of export packet storing records of the same type.

Here, very basic operation is done: if the record belongs to the flowset currently being assembled, it is just added. If not, the current flowset is terminated and a new one is created to be able to store the record.

From here on, one could name the record as a flowset record.

A record gets to the collector when the packet currently being built is too big. Flowexporter packets are 1200 bytes long. So, if the record adding exceeds the packet size, the packet is sent and a new one is created to be able to send the exceeding record later on.

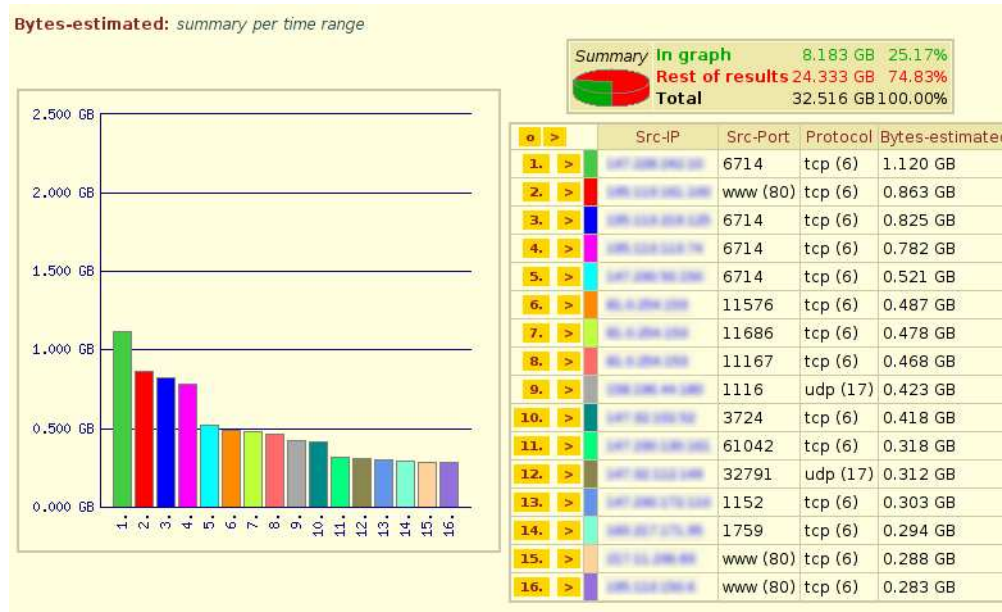


Figure 6: NetFlow data processed by FTAS collector[Kos14][Kos15]

6 Package system

Package system is a tool for easier preparation of distribution packages. These packages contain necessary files to start work with COMBO cards. The external testers can simply plug the card, proceed installation of the package and start to use the card. The first such package was created in the summer, but it was created completely manually. After experiences from this package we decided to develop a tool, which would be able to build packages automatically.

Package system is a shell script that copies necessary files from CVS (*Concurrent Versions System*) tree and from project machines. Package system is universal for all current Liberouter projects and it can be easily adapted to any new project. This universality warrants unified form of all distribution packages. Contents of the specific package is dependent on the project database file (each project has own database file). Database file contains information for the package system script about files which should be copied from CVS and alternatively what changes have to be performed (*cvs update* or removing of some subdirectories). The contents of database file is not limited to paths in the CVS tree, e.g. MCS

files are stored separately on the project machines. This is the reason why the script can be run only on the project machines. However, some files stored in the CVS tree can contain data (usually paths) which are all right in the CVS, but in the package structure they are incorrect (e.g. package doesn't contain all directories). This problem is solved by modified files which are stored separately in the CVS tree. The script copies these files at the end and rewrites all incorrect files. This is a way how to add files which shouldn't be stored in the CVS into the package (typically main README or release notes for the package). The last operation is generation of distribution package (archive file - .tgz).

6.1 Package contents

Package must contain all necessary files for card initialization (*csboot(1)*, *csbus(1)*, *csid(1)*, etc.). The most important files are MCS files. These files contain programs for the FPGAs used on the COMBO cards. Kernel drivers are next vital part of the package. Drivers provide means for communication between COMBO cards and operating system (applications running there). There is range of user space applications which can be part of package but it depends on the given project. All these tools (and drivers) are distributed as source code files. Package installation is covered by the build system. This system allows users simple process of installation. Build system perform automatic compilation of all necessary tools and eventually their installation into the system. Drivers are compiled separately, but the compilation isn't complicated (it is very similar to build system) and it's well described in the README file. In this moment user has prepared all necessary tools on his machine but he still can't use the card. This step is covered by scripts which are specific for the concrete project. Generally these scripts have to boot firmware programs into the COMBO cards, initialize design and attach the driver. Next activity is dependent on the concrete project (interfaces can be configured or some application can be run).

6.2 Preparation of new package

The preparation of new package begins with modification of the database file for the appropriate project. It means that lines with information about files that will contain the package must be added. These lines must contain information about package version and alternatively about changes which have to be performed (*CVS update* or removing subdirectory). When the database contains information about whole directory structure in the new package we have to prepare files which aren't in the CVS (or they are incorrect). It includes preparation of the main README file and files with description about changes since last package version (release notes - RELNOTES file). At this moment it's time to "tell" script that new package exists. It means adding general information about package in the list of packages. The file with this list is stored near the package script and it

contains names of all known packages, path to database files and the firmware version. Firmware version is more detailed described in the database file (path to the mcs files on the project machine). Package name is usually compound of the project name and the version of the package.

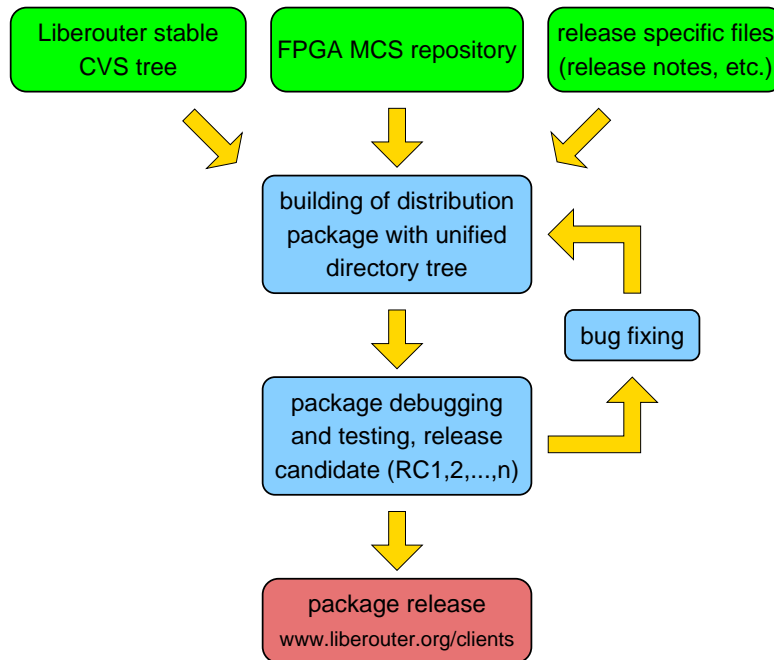


Figure 7: NetFlow package system

Preparation of new package version is more simple because the most of files are prepared in the previous version and only changes must be defined. Our policy says that new package must be released at least every three months. In the case that no new features were added during last three months, it's necessary to announce it at client's web. But usually there are some changes in tools which are part of the package. In this case we simply use package system to create last version of the package with up-to-date versions of changed tools (stored in the CVS).

Before releasing the package we test packages on the computers with the basic installation of supported operating systems. This testing is used for checking function of the package, i.e. proper function of the build system and scripts for booting COMBO card.

After finishing testing (testing period is usually one week), the new package will be released. Distribution to our customers (external testers) is provided through our website. For this purpose we have client's web (at

<http://www.liberouter.org/clients¹>). After simple registration, anybody can download required version of the package.

7 Conclusion

We succeeded in reading flow records from NetFlow probe hardware and generating NetFlow v9 datagrams. The probe has been successfully tested during last three months. IP flows from real university backbone have been acquired and NetFlow v9 datagrams sent to the FTAS (*Flow-Based Traffic Analysis System*) collector. NetFlow package (hw and sw) was delivered to foreign testers in Holland.

Realized software solution is based on several basic entities wired together with shell scripts. The development of software was closely connected with progress in hardware implementation. We have developed several tools to support different hardware requirements and configurations. To improve this situation we will focus in the future releases on the following topics:

- integration of NetFlow framework tools and creation of NetFlow probe hardware manager (system daemon), which will be responsible for common configuration (FPGA and HFE booting, hardware setup, etc.) and providing shared configuration interface to other user space applications.
- enhancement of supported features by the *flowexporter(1)*
 - support for Cisco NetFlow v5,
 - software flow filtering based on IP ranges,
 - support for Netopeer configuration utility (XML configuration),
 - IPFIX (*Internet Protocol Flow Information Export*),
 - flow anonymization,
 - ...
- support for new hardware features and platforms which will be added.

References

[CoHW] Liberouter Project. *Description of COMBO cards*.
<http://www.liberouter.org/hardware.php>

¹<http://www.liberouter.org/clients>

- [NetHowTo] Liberouter Project. *NetFlow probe HOWTO*.
<http://www.liberouter.org/netflow/userdoc.php>
- [Kos14] Košňar, T. *Notes to Flow-Based Traffic Analysis System Design*. CESNET Technical Report 14/2004.
- [Kos15] Košňar, T. *Flow-Based Traffic Analysis System - Architecture Overview*. CESNET Technical Report 15/2004.
- [RFC3954] Network Working Group. *RFC3954 - Cisco Systems NetFlow Services Export Version 9*.
- [Zad04] Žádník, M. *Overview of NetFlow Monitoring Adapter*. CESNET Technical Report 8/2004.
- [Zad05] Žádník, M. *NetFlow probe firmware design*.
<http://www.liberouter.org/netflow/design.php>