Network Monitoring with FlowScan and JKFlow

Jurgen Kobierczynski

Situation

Measuring network traffic, is a common task in network management. The reason behind network measuring can be various: It's easier to fix a malfunctioning network that's been monitored, redesigning a network requires a good insight of the running processes over the network, a virus outbreak or a network attack over the network must be detected, firewall rules or network accounting over a running network has to be implemented, and other reasons.
Classic well-known tools like MRTG, RRDTool are good for most basic network interface traffic measurement, but for some these tools are not enough.

Here I describe a Netflow network monitoring framework using the open source tools Flow-Tools, FlowScan and JKFlow. These tools provides a network traffic measurement infrastructure with detailed online graphing and scoreboard capabilites.

FlowScan

FlowScan is a collection of Perl scripts programmed by Dave Plonka of University of Wisconsin, Madison. In using Netflow the role of data collection is done by the router sending netflow datagrams (short called 'flows') to a netflow collector. The collector saves these flows in 'flowfiles'. Usually these files will consume a lot of diskspace over time and the needed information has to be extracted. Using a 'report module' FlowScan processes every flowfile and creates the reports needed .

FlowScan supports with the Cflow module multiple flow formats like argus, cflowd, flow-tools, and lfapd. The Fprobe tool using pcap library makes flow measurement also possible on Unix hosts, so you are not restricted to Cisco routers.

FlowScan depends on these tools & perl modules:

- Korn Shell (only if you use CampusIO/SubnetIO)
- RRDTool + RRDs shared library
- Cflow
- Boulder::Stream
- ConfigReader::Directivestyle
- HTML::Table
- Net::Patricia
- Flow-Tools' flow-capture or cflowd

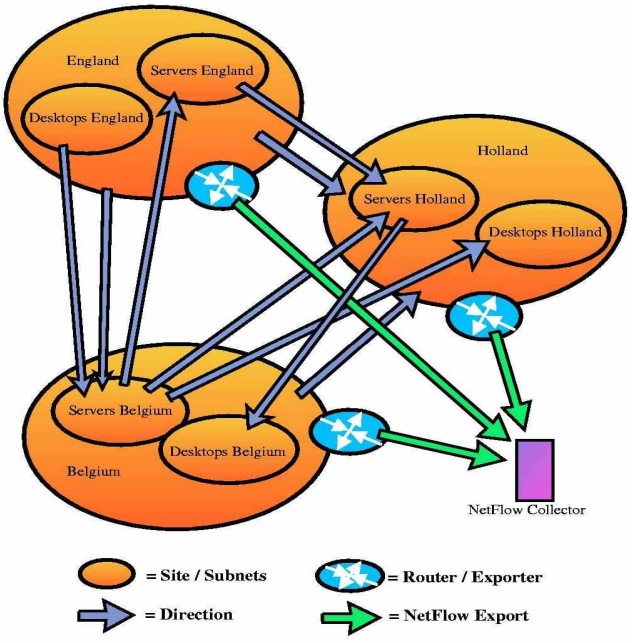
On the FlowScan website the collector cflowd is proposed, but lacking support of gcc 3.x, makes flow-capture from Flow-Tools, a package by Mark Fullmer, now the recommended solution.

Motivation

In my study for ACE graduate Telecom at GroepT Leuven, Belgium, I was facing a FlowScan setup using Netflow on a few core routers on a Corporate European hub site. Because site-to-site reporting of services was needed, I looked both at the included CampusIO report module for reporting on services, and the SubnetIO module for subnet reporting. After some inspection of the code I saw I couldn't combine these modules because they do not work together. Possible solutions to this problem like running multiple FlowScan processes on filtered flows, or using a different report module, were evaluated. In my search for a better report module I've found CUFlow, written by Matt Selsky and Johan Andersen of Columbia University. While this tool provides excellent reporting on services for different routers, and has a user friendly web interface, this still didn't solved my site-to-site reporting problem. To solve my problem I've decided to program JKFlow, not only to solve my problem but also to provide a new generic reporting module to the open source community. Thanks to the feedback from the community JKFlow has improved a lot.

JKFlow report module

JKFlow is a generic report module and very flexible to configure using a XML configuration file. To setup JKFlow just copy JKFlow.pm, JKFlow.xml to the location where you've installed FlowScan, copy JKGrapher.pl to the web-cgi directory of your webserver and edit JKFlow.xml. The next picture shows some of the basic concepts and entities of JKFlow.



JKFlow sites, direction entities

- Sites/Subnets define source/destination subnets and are used in directions, which combine 2 Sites/Subnets pairs.
- Directions selects flows with matching source/destination sites/subnets, inside these directions you define traffic pattern to monitor.
- Outbound traffic matches source address matching with 'from' Subnets/Sites and destination address with 'to' Subnets/Sites. Inbound matches otherwise.
- You can monitor multiple Directions, within each Direction you can specify traffic patterns to monitor like applications, services, protocols, total & scoreboarding, but you can also specify Sets.
- Sets are grouping of traffic patterns to watch. You can reuse Sets over multiple Directions.

Here is a very basic JKFlow.xml file. Most of the configuration file is self explaining:

Example 1 JKFlow.xml file

```
<config>
  <definesets>
    <defineset name="Common Services">
      <application name="web">80/tcp,443/tcp</application>
      <application name="mail">110/tcp,143/tcp</application>
      <application name="dns">53/udp,53/tcp</application>
      <protocols>tcp,udp,icmp</protocols>
      <services>22-23/tcp,25/tcp,102/tcp,119/tcp</services>
      <total/>
    </defineset>
    <defineset name="Scoreboarding">
      <scoreboard>
        <report count="12" hostsbase="AggHost1H" portsbase="AggPort1H"/>
        <report count="72" hostsbase="AggHost6H" portsbase="AggPort6H"/>
      </scoreboard>
    </defineset>
  </definesets>
  <sites>
    <site name="Belgium" subnets="10.10.0.0/16"/>
    <site name="Holland" subnets="10.20.0.0/16"/>
    <site name="England" subnets="10.30.0.0/16"/>
    <site name="Internet" subnets="0.0.0.0/0"/>
  </sites>
  <directions>
    <direction name="Belgium-Int" from="Belgium" to="Internet" noto="Belgium">
      <set name="Common Services"/>
      <set name="Scoreboarding"/>
    </direction>
    <direction name="Belgium-Holland" from="Belgium" to="Holland">
      <set name="Common Services"/>
      <set name="Scoreboarding"/>
    </direction>
    <direction name="Belgium-England" from="Belgium" to="England">
      <set name="Common Services"/>
      <set name="Scoreboarding"/>
    </direction>
  </directions>
  <rrddir>/var/flows/reports/rrds</rrddir>
  <scoredir>/var/flows/score</scoredir>
  <sampletime>300</sampletime>
</config>
```

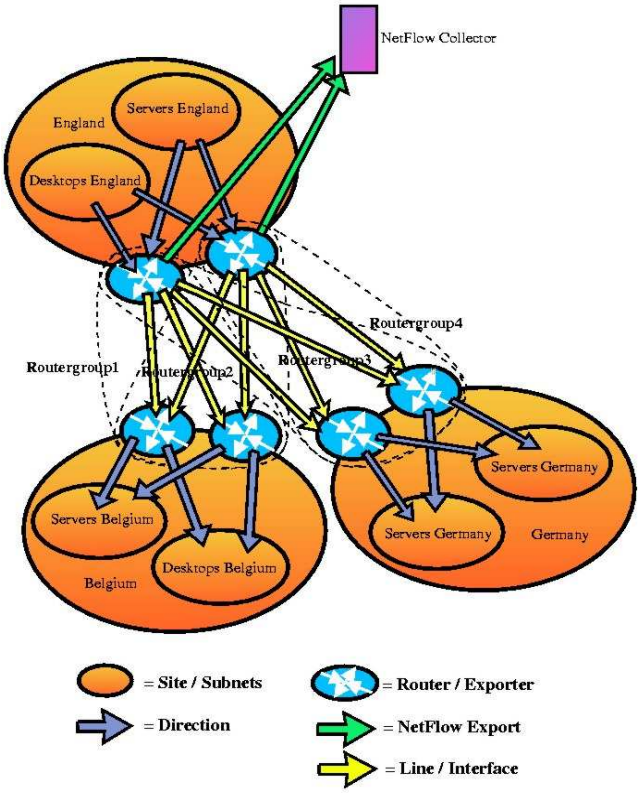
What JKFlow actually does at the startup of FlowScan it will create several directories under rrddir named after the directions specified in JKFlow.xml. In each directory it will automatically create the necessary RRDTool files for the specified traffic patterns. FlowScan processes all flow files, with each it will updates the values in RRDTool files, and remove the processed file. A second set of directions under scoredir is created and inside each directory aggregate scoreboard HTML files are created.

This basic configuration works for scenarios with a single central core router, passing all network traffic. Often this is not the case, and multiple routers are deployed, with each routing some of the subnets/sites network traffic. The first example use 3 routers and all are exporters. Netflow doesn't provide a correlation of the flows between the exporters. How do you prevent over counting correlating flows from different exporters, while still monitoring all networks?

Because users were reporting this problem, this showed me that that direction matching on subnets alone does not work in the real world. To make JKFlow really useful, I've also included an 'edge' monitoring design. With a 'edge' approach, you assume network traffic will not pass multiple links to a network. Take the case of 2 redundant fail-back routers. Network traffic can pass both routers, but they pass a single link to the outside world.

This 'edge' approach is can be done using 'routergroups'. Routergroups groups multiple router interfaces or routers, and a direction with a routergroup selects all network traffic passing any router interface included within the routergroup.

In the following picture is a common scenario involving redundant lines and routers between a central site and 2 hub sites:



JKFlow routergroup entities

You'll have to combine the 4 lines traffic over the redundant routers of a site. This could be easily done using subnet-to-subnet directions, if all network traffic passes a single router of the sites redundant pair only. In the case this is not possible, you will have to use routergroup based directions.

Here below is the JKFlow.xml configuration file:

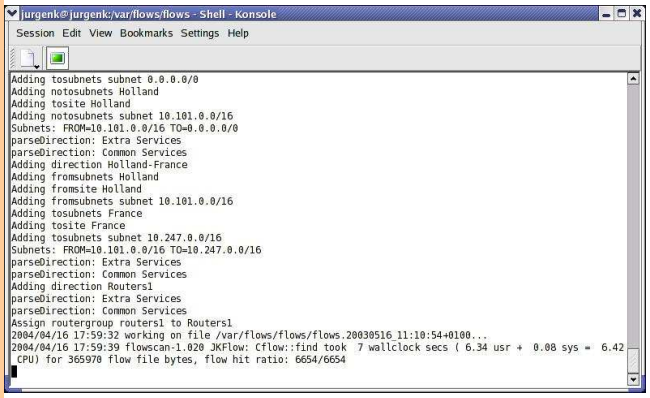
Example 2 JKFlow.xml file

```
<config>
  <definesets>
    <defineset name="Common Services">
      <application name="web">80/tcp,8080/tcp,443/tcp</application>
      <application name="mailreading">110/tcp,143/tcp</application>
      <application name="dns">53/udp,53/tcp</application>
      <services>22-23/tcp,25/tcp,102/tcp,119/tcp</services>
      <protocols>tcp,udp,icmp</protocols>
      <ftp/>
      <multicasts/>
      <tos/>
      <total/>
    </defineset>
  </definesets>
  <routergroups>
    <routergroup name="ROUTERGROUP1">
      <router exporter="10.2.2.2" interface="9"/>
      <router exporter="10.2.2.3" interface="9"/>
      <router exporter="10.2.2.2" interface="12"/>
      <router exporter="10.2.2.3" interface="12"/>
      <router exporter="10.2.2.2" interface="13"/>
      <router exporter="10.2.2.3" interface="13"/>
    </routergroup>
    <routergroup name="ROUTERGROUP1-LINE1">
      <router exporter="10.2.2.2" interface="9"/>
      <router exporter="10.2.2.3" interface="9"/>
    </routergroup>
    <routergroup name="ROUTERGROUP1-LINE2">
      <router exporter="10.2.2.2" interface="12"/>
      <router exporter="10.2.2.3" interface="12"/>
    </routergroup>
    <routergroup name="ROUTERGROUP1-LINE3">
      <router exporter="10.2.2.2" interface="13"/>
      <router exporter="10.2.2.3" interface="13"/>
    </routergroup>
  </routergroups>
  <directions>
    <direction name="ROUTERS1" routergroup="ROUTERGROUP1">
      <set name="Common Services"/>
    </direction>
    <direction name="ROUTERS1-LINE1" routergroup="ROUTERGROUP1-LINE1">
      <set name="Common Services"/>
    </direction>
    <direction name="ROUTERS1-LINE2" routergroup="ROUTERGROUP1-LINE2">
      <set name="Common Services"/>
    </direction>
    <direction name="ROUTERS1-LINE3" routergroup="ROUTERGROUP1-LINE3">
      <set name="Common Services"/>
    </direction>
  </directions>
  <rrddir>/var/flows/report/rrds</rrddir>
  <scoredir>/var/flows/score</scoredir>
  <sampletime>300</sampletime>
</config>
```

In network monitoring architectures where multiple exporters creates redundancies in flow reporting, like 2 stub routers reporting both flows of the same traffic of between the stub sites, in defining an 'edge' consisting of a routergroup of all routerinterfaces of the routers on the 'edge', this can solve the redundancy problem. In assuming that no network traffic passes the 'edge' more than once you can group routers and avoid double counting of flows. You can combine both from/to sites/subnets and routergroups.

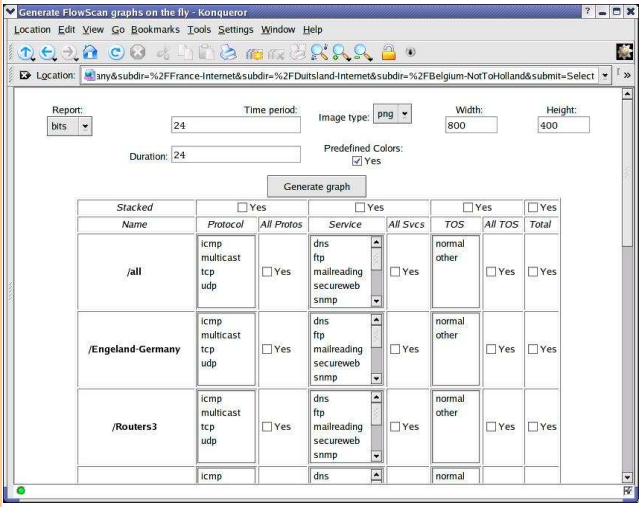
Running FlowScan and JKFlow

When you run FlowScan for the first time it will find the flow files already created by flow-capture in the /var/flows/flows directory and starts processing them. When no files are found it will wait 30 seconds and tries again. On startup several initialization messages are dumped by JKFlow, which can help in troubleshooting. When a flow file is processed it will dump the values in directories containing RRDTool files (and creates those when they aren't created yet.)



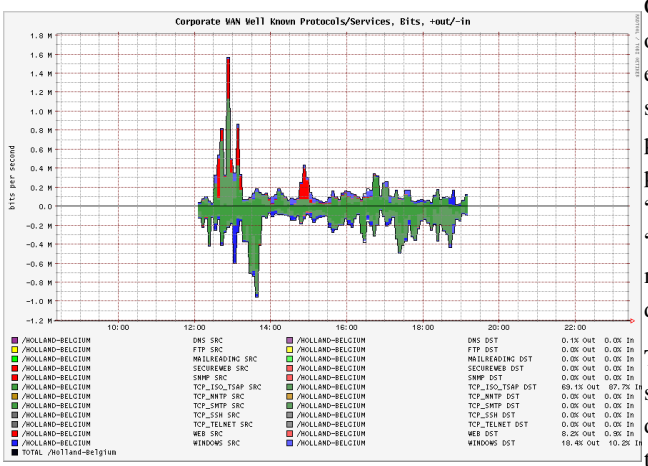
FlowScan and JKFlow.pm running

The results are displayed with a CGI-script called JKGrapher.pl on a webserver, like Apache. You select first the directions on which you want to report, and select the services on the directions. Here is a screenshot displaying the JKGrapher.pl CGI-frontend.



JKGrapher.pl CGI-Frontend

After pressing “Generate graph” a RRD graph appear. This graph is created real-time and the URL used to obtain this graph can be copied into a HTML web page for easy retrieval. Because the URL represents a single image, you can link these in as images.



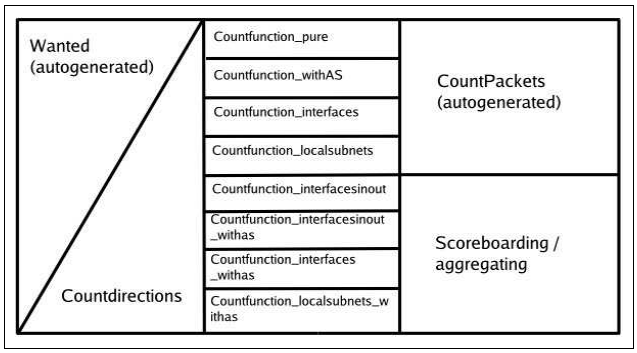
Example JKFlow Graph

Under the hood of JKFlow

FlowScan calls the JKFlow report module during 3 moments:

- at startup the module parses JKFlow.xml and initialize the counter data structures.
- for every flow the 'wanted' function updates the counters of the matching directions
- after processing the flowfile, all RRDTool files in the directories are updated using the counters of corresponding directions.

JKFlow is not only very flexible but also efficient in flow processing. The wanted function, responsible for processing every flow, is critical in this aspect. The following picture outlines the flow processing structure of the wanted function of Jkflow:



This picture outlines the internal working of the JKFlow flow processing and gives you an insight to it's configuration. A mayor part in understanding lies in to understanding the difference between the logic to split traffic to directions, and the logic measuring services, protocols, tos, total, ftp, etc.. and scoreboarding inside the directions. This reflects directly in the configuration rules where services, protocols, tos, total, ftp, etc... and other elements are always defined inside directions, possible with aid of definsets and sets. Sets are templates of services/protocols/tos/total/ftp/etc...

To lookup these directions as fast as possible, I have programmed Countdirections which calls the countPackets on basis of source/destination IP-address to subnets to directions Net::Patricia matching. This works as described for source/destination subnets. Combining this functionality with the matching of flows on basis of routers, interfaces, autonomous systems too is however not easy. Introducing such matching code into source/destination subnets matching code would lay a heavy burden on the performance of the code, so another way is implemented. In the picture below there is a second layer between the “Countdirections” and the “CountPackets”, consisting out of “Countfunction_*” functions. These functions will process every matched direction on further constraints defined within every direction.

This is not actually not complete: what to do when no source/destination subnets/or sites where defined within a direction? In the wanted function is this solved by evaluating these directions using these Countfunction_* functions direction outside Countdirections. So the whole picture consists of Wanted calling Countdirections calling Countfunctions calling Countpackets, Wanted calling Countfunctions calling Countpackets, Wanted calling Countdirections calling CountPackets and at last Wanted calling CountPackets. The whole structure on what has to call what is defined at startup using autogeneration of the wanted code, and registrate the references of the correct Countfunction on the directions. The Countfunctions at last calls the Countpackets which are also autogenerated on startup for every direction, so asymmetric configurations with directions with just </all> and others with lots of services, protocols, tos, etc... are processed as fast as possible.

These measures explains why JKFlow, while so flexible, is still very efficient in processing.

Conclusion

While JKFlow in size isn't comparable with other tools, it adds lots of flexibility to FlowScan. This tool provides a cheap and flexible NetFlow monitoring solution.

Links

JKFlow: http://jkflow.sourceforge.net
JKFlow Demo: http://jkflow.sourceforge.net/jkflowdemo.html
JKFlow Manual: http://jkflow.sourceforge.net/eindwerk.pdf
FlowScan: http://www.caida.org/tools/utilities/FlowScan/
RRDTool: http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/
Software: http://www.switch.ch/tf-tant/floma/software.html

Written by: Jurgen Kobierczynski
Email: jurgen.kobierczynski@telenet.be