

DSC Manual

Duane Wessels
The Measurement Factory, Inc.

<http://dns.measurement-factory.com/tools/dsc/>

November 29, 2006

Contents

1	Introduction	3
1.1	Components	3
1.1.1	The Collector	3
1.1.2	XML Data Transfer	4
1.1.3	The Extractor	4
1.1.4	The Grapher	4
1.2	Architecture	5
2	Installing the Presenter	6
2.1	Install Perl Dependencies	6
2.2	Install Ploticus	7
2.3	Install DSC Software	7
2.4	CGI Symbolic Links	7
2.5	/usr/local/dsc/data	8
2.5.1	X.509 method	8
2.5.2	rsync Method	8
2.6	/usr/local/dsc/var/log	9
2.7	/usr/local/dsc/cache	9
2.8	Cron Jobs	9
2.9	Data URIs	10
3	Configuring the DSC Presenter	11
3.1	Generating X.509 Certificates	11
3.1.1	Certificate Authority	11
3.1.2	Server Certificate	12
3.2	Client Certificates	13
3.3	Apache Configuration	15
4	Collector Installation	16
4.1	Prerequisites	16
4.2	Installation	16
4.3	Uploading XML Files	17
4.3.1	upload-x509.sh	18
4.3.2	upload-rsync.sh	18

5	Configuring and Running the DSC Collector	20
5.1	dsc.conf	20
5.2	A Complete Sample dsc.conf	21
5.3	Running dsc	22
6	Viewing DSC Graphs	23
7	DSC Datasets	26
7.1	Dataset Name	26
7.2	Protocol	27
7.3	Indexers	27
7.3.1	IP Indexers	27
7.3.2	IP Filters	28
7.3.3	DNS Indexers	28
7.3.4	DNS Filters	30
7.3.5	Parameters	31
8	Data Storage	33
8.1	XML Structure	33
8.1.1	XML File Naming Conventions	35
8.2	Archived Data Format	35
8.2.1	Format 1	35
8.2.2	Format 2	36
8.2.3	Format 3	36
8.2.4	Format 4	36
9	Bugs	37

Chapter 1

Introduction

DSC is a system for collecting and presenting statistics from a busy DNS server.

1.1 Components

DSC consists of the following components:

- A data collector
- A data presenter, where data is archived and rendered
- A method for securely transferring data from the collector to the presenter
- Utilities and scripts that parse XML and archive files from the collector
- Utilities and scripts that generate graphs and HTML pages

1.1.1 The Collector

The collector is a binary program, named `dsc`, which snoops on DNS messages. It is written in C and uses *libpcap* for packet capture.

`dsc` uses a relatively simple configuration file called *dsc.conf* to define certain parameters and options. The configuration file also determines the *datasets* that `dsc` collects.

A Dataset is a 2-D array of counters of IP/DNS message properties. You can define each dimension of the array independently. For example you might define a dataset categorized by DNS query type along one dimension and TLD along the other. `dsc` dumps the datasets from memory to XML files every 60 seconds.

1.1.2 XML Data Transfer

You may run the DSC collector on a remote machine. That is, the collector may run on a different machine than where the data is archived and displayed. DSC includes some Perl and `/bin/sh` scripts to move XML files from collector to presenter. One technique uses X.509 certificates and a secure HTTP server. The other uses *rsync*, presumably over *ssh*.

X.509/SSL

To make this work, `Apache/mod_ssl` should run on the machine where data is archived and presented. Data transfer is authenticated via SSL X.509 certificates. A Perl CGI script handles all PUT requests on the server. If the client certificate is allowed, XML files are stored in the appropriate directory.

A shell script runs on the collector to upload the XML files. It uses `curl`¹ to establish an HTTPS connection. XML files are bundled together with `tar` before transfer to eliminate per-connection delays. You could use `scp` or `rsync` instead of `curl` if you like.

`put-file.pl` is the script that accepts PUT requests on the HTTP server. The HTTP server validates the client's X.509 certificate. If the certificate is invalid, the PUT request is denied. This script reads environment variables to get X.509 parameters. The uploaded-data is stored in a directory based on the X.509 Organizational Unit (server) and Common Name fields (node).

rsync/ssh

This technique uses the *rsync* utility to transfer files. You'll probably want to use *ssh* as the underlying transport, although you can still use the less-secure *rsh* or native rsync server transports if you like.

If you use *ssh* then you'll need to create passphrase-less SSH keys so that the transfer can occur automatically. You may want to create special *dsc* userids on both ends as well.

1.1.3 The Extractor

The XML extractor is a Perl script that reads the XML files from `dsc`. The extractor essentially converts the XML-structured data to a format that is easier (faster) for the graphing tools to parse. Currently the extracted data files are line-based ASCII text files. Support for SQL databases is planned for the future.

1.1.4 The Grapher

DSC uses *Ploticus*² as the graphing engine. A Perl module and CGI script read extracted data files and generate Ploticus scriptfiles to generate plots. Plots are always generated on demand via the CGI application.

¹<http://curl.haxx.se>

²<http://ploticus.sourceforge.net/>

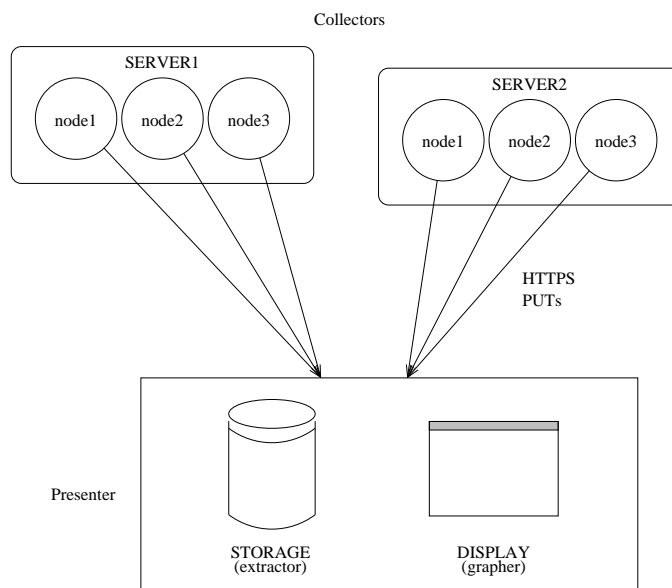


Figure 1.1: The DSC architecture.

`dsc-grapher.pl` is the script that displays graphs from the archived data.

1.2 Architecture

Figure 1.1 shows the DSC architecture.

Note that DSC utilizes the concept of *servers* and *nodes*. A server is generally a logical service, which may actually consist of multiple nodes. Figure 1.1 shows six collectors (the circles) and two servers (the rounded rectangles). For a real-world example, consider a DNS root server. IP Anycast allows a DNS root server to have geographically distributed nodes that share a single IP address. We call each instance a *node* and all nodes sharing the single IP address belong to the same *server*.

The DSC collector program runs on or near³ the remote nodes. Its XML output is transferred to the presentation machine via HTTPS PUTs (or something simpler if you prefer).

The presentation machine includes an HTTP(S) server. The extractor looks for XML files PUT there by the collectors. A CGI script also runs on the HTTP server to display graphs and other information.

³by “near” we mean that packets may be sniffed remotely via Ethernet taps, switch port mirroring, or a SPAN port.

Chapter 2

Installing the Presenter

You'll probably want to get the Presenter working before the Collector. If you're using the secure XML data transfer, you'll need to generate both client- and server-side X.509 certificates.

Installing the Presenter involves the following steps:

- Install Perl dependencies
- Install DSC software
- Create X.509 certificates (optional)
- Set up a secure HTTP server (e.g., Apache and `mod_ssl`)
- Add some cron jobs

2.1 Install Perl Dependencies

DSC uses Perl for the extractor and grapher components. Chances are that you'll need Perl-5.8, or maybe only Perl-5.6. You'll also need these readily available third-party Perl modules, which you can find via CPAN:

- CGI-Untaint (CGI::Untaint)
- CGI.pm (CGI)
- Digest-MD5 (Digest::MD5)
- File-Flock (File::Flock)
- File-Spec (File::Spec)
- File-Temp (File::Temp)
- Geography-Countries (Geography::Countries)
- Hash-Merge (Hash::Merge)
- IP-Country (IP::Country)
- MIME-Base64 (MIME::Base64)

- Math-Calc-Units (Math::Calc::Units)
- Scalar-List-Utils (List::Util)
- Text-Template (Text::Template)
- URI (URI::Escape)
- XML-Simple (XML::Simple)

Also note that XML::Simple requires XML::Parser, which in turn requires the *expat* package.

2.2 Install Ploticus

DSC uses Ploticus to generate plots and graphs. You can find this software at <http://ploticus.sourceforge.net>. The *Download* page has links to some pre-compiled binaries and packages. FreeBSD and NetBSD users can find Ploticus in the ports/packages collection.

2.3 Install DSC Software

All of the extractor and grapher tools are Perl or `/bin/sh` scripts, so there is no need to compile anything. Still, you should run `make` first:

```
% cd presenter
% make
```

If you see errors about missing Perl prerequisites, you may want to correct those before continuing.

The next step is to install the files. Recall that `/usr/local/dsc` is the hard-coded installation prefix. You must create it manually:

```
% mkdir /usr/local/dsc
% make install
```

Note that DSC's Perl modules are installed in the "site_perl" directory. You'll probably need *root* privileges to install files there.

2.4 CGI Symbolic Links

DSC has a couple of CGI scripts that are installed into `/usr/local/dsc/libexec`. You should add symbolic links from your HTTP server's `cgi-bin` directory to these scripts.

Both of these scripts have been designed to be `mod_perl`-friendly.

```
% cd /usr/local/apache/cgi-bin
% ln -s /usr/local/dsc/libexec/put-file.pl
% ln -s /usr/local/dsc/libexec/dsc-grapher.pl
```

You can skip the `put-file.pl` link if you plan to use *rsync* to transfer XML files. If you cannot create symbolic links, you'll need to manually copy the scripts to the appropriate directory.

2.5 /usr/local/dsc/data

2.5.1 X.509 method

This directory is where `put-file.pl` writes incoming XML files. It should have been created when you ran *make install* earlier. XML files are actually placed in *server* and *node* subdirectories based on the authorized client X.509 certificate parameters. If you want `put-file.pl` to automatically create the subdirectories, the `data` directory must be writable by the process owner:

```
% chgrp nobody /usr/local/dsc/data/
% chmod 2775 /usr/local/dsc/data/
```

Alternatively, you can create *server* and *node* directories in advance and make those writable.

```
% mkdir /usr/local/dsc/data/x-root/
% mkdir /usr/local/dsc/data/x-root/blah/
% chgrp nobody /usr/local/dsc/data/x-root/blah/
% chmod 2775 /usr/local/dsc/data/x-root/blah/
```

Make sure that `/usr/local/dsc/data/` is on a large partition with plenty of free space. You can make it a symbolic link to another partition if necessary. Note that a typical DSC installation for a large DNS root server requires about 4GB to hold a year's worth of data.

2.5.2 rsync Method

The directory structure is the same as above (for X.509). The only differences are that:

- The *server* subdirectories must be made in advance.
- The directories should be writable by the userid associated with the *rsync/ssh* connection. You may want to create a dedicated *dsc* userid for this.

2.6 /usr/local/dsc/var/log

The `put-file.pl` script logs its activity to `put-file.log` in this directory. It should have been created when you ran *make install* earlier. The directory should be writable by the HTTP server userid (usually *nobody* or *www*). Unfortunately the installation isn't fancy enough to determine that userid yet, so you must change the ownership manually:

```
% chgrp nobody /usr/local/dsc/var/log/
```

Furthermore, you probably want to make sure the log file does not grow indefinitely. For example, on FreeBSD we add this line to `/etc/newsyslog.conf`:

```
/usr/local/dsc/var/log/put-file.log nobody:wheel      644  10    *    @T00  BN
```

You need not worry about this directory if you are using the *rsync* upload method.

2.7 /usr/local/dsc/cache

This directory, also created by *make install* above, holds cached plot images. It also must be writable by the HTTP userid:

```
% chgrp nobody /usr/local/dsc/cache/
```

2.8 Cron Jobs

DSC requires two cron jobs on the Presenter. The first is the one that processes incoming XML files. It is called `refile-and-grok.sh`. We recommend running it every minute. You also may want to run the jobs at a lowerer priority with *nice*. Here is the cron job that we use:

```
* * * * * /usr/bin/nice -10 /usr/local/dsc/libexec/refile-and-grok.sh
```

The other useful cron script is `remove-xmles.pl`. It removes XML files older than a specified number of days. Since most of the information in the XML files is archived into easier-to-parse data files, you can remove the XML files after a few days. This is the job that we use:

```
@midnight find /usr/local/dsc/data/ | /usr/local/dsc/libexec/remove-xmles.pl 7
```

2.9 Data URIs

DSC uses “Data URIs” by default. This is a URI where the content is base-64 encoded into the URI string. It allows us to include images directly in HTML output, such that the browser does not have to make additional HTTP requests for the images. Data URIs may not work with some browsers.

To disable Data URIs, edit *presenter/perl/lib/DSC/grapher.pm* and change this line:

```
$use_data_uri = 1;
```

to

```
$use_data_uri = 0;
```

Also make this symbolic link from your HTTP servers “htdocs” directory:

```
# cd htdocs
# ln -s /usr/local/dsc/share/html dsc
```

Chapter 3

Configuring the DSC Presenter

This chapter describes how to create X.509 certificates and configure Apache/mod_ssl. If you plan on using a different upload technique (such as scp or rsync) you can skip these instructions.

3.1 Generating X.509 Certificates

We use X.509 certificates to authenticate both sides of an SSL connection when uploading XML data files from the collector to the presenter.

Certificate generation is a tricky thing. We use three different types of certificates:

1. A self-signed root CA certificate
2. A server certificate
3. Client certificates for each collector node

In the client certificates we use X.509 fields to store the collector's server and node name. The Organizational Unit Name (OU) becomes the server name and the Common Name (CN) becomes the node name.

The DSC source code distribution includes some shell scripts that we have used to create X.509 certificates. You can find them in the `presenter/certs` directory. Note these are not installed into `/usr/local/dsc`. You should edit `openssl.conf` and enter the relevant information for your organization.

3.1.1 Certificate Authority

You may need to create a self-signed certificate authority if you don't already have one. The CA signs client and server certificates. You will need to distribute

the CA and client certificates to collector sites. Here is how to use our `create-ca-cert.sh` script:

```
% sh create-ca-cert.sh
CREATING CA CERT
Generating a 2048 bit RSA private key
.....
.....+++
.....+++
writing new private key to './private/cakey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
```

3.1.2 Server Certificate

The server certificate is used by the HTTP server (Apache/mod_ssl). The clients will have a copy of the CA certificate so they can validate the server's certificate when uploading XML files. Use the `create-srv-cert.sh` script to create a server certificate:

```
% sh create-srv-cert.sh
CREATING SERVER REQUEST
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to 'server/server.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----

Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Colorado
Locality Name (eg, city) []:Boulder
Organization Name (eg, company) [Internet Widgits Pty Ltd]:The Measurement Factory, Inc
Organizational Unit Name (eg, section) []:DNS
Common Name (eg, YOUR name) []:dns.measurement-factory.com
Email Address []:wessels@measurement-factory.com

Please enter the following 'extra' attributes
to be sent with your certificate request
```

```

A challenge password []:
An optional company name []:
Enter pass phrase for server/server.key:
writing RSA key
CREATING SERVER CERT
Using configuration from ./openssl.conf
Enter pass phrase for ./private/cakey.pem:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName             :PRINTABLE:'US'
stateOrProvinceName     :PRINTABLE:'Colorado'
localityName            :PRINTABLE:'Boulder'
organizationName        :PRINTABLE:'The Measurement Factory, Inc'
organizationalUnitName  :PRINTABLE:'DNS'
commonName              :PRINTABLE:'dns.measurement-factory.com'
emailAddress            :IA5STRING:'wessels@measurement-factory.com'
Certificate is to be certified until Jun  3 20:06:17 2013 GMT (3000 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

```

Note that the Common Name must match the hostname of the HTTP server that receives XML files.

Note that the `create-srv-cert.sh` script rewrites the server key file without the RSA password. This allows your HTTP server to start automatically without prompting for the password.

The script leaves the server certificate and key in the `server` directory. You'll need to copy these over to the HTTP server config directory as described later in this chapter.

3.2 Client Certificates

Generating client certificates is similar. Remember that the Organizational Unit Name and Common Name correspond to the collector's *server* and *node* names. For example:

```

% sh create-clt-cert.sh
CREATING CLIENT REQUEST
Generating a 1024 bit RSA private key
.....++++++
.....++++++
writing new private key to 'client/client.key'
Enter PEM pass phrase:

```

```

Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Los Angeles
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Some DNS Server
Organizational Unit Name (eg, section) []:x-root
Common Name (eg, YOUR name) []:LAX
Email Address []:noc@example.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
CREATING CLIENT CERT
Using configuration from ./openssl.conf
Enter pass phrase for ./private/cakey.pem:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName             :PRINTABLE:'US'
stateOrProvinceName     :PRINTABLE:'California'
localityName            :PRINTABLE:'Los Angeles'
organizationName        :PRINTABLE:'Some DNS Server'
organizationalUnitName  :PRINTABLE:'x-root'
commonName              :PRINTABLE:'LAX'
emailAddress            :IA5STRING:'noc@example.com'
Certificate is to be certified until Jun  3 20:17:24 2013 GMT (3000 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
Enter pass phrase for client/client.key:
writing RSA key
writing RSA key

```

The client's key and certificate will be placed in a directory based on the server and node names. For example:

```
% ls -l client/x-root/LAX
total 10
-rw-r--r-- 1 wessels wessels 3311 Mar 17 13:17 client.crt
-rw-r--r-- 1 wessels wessels 712 Mar 17 13:17 client.csr
-r----- 1 wessels wessels 887 Mar 17 13:17 client.key
-rw-r--r-- 1 wessels wessels 1953 Mar 17 13:17 client.pem
```

The `client.pem` (and `cacert.pem`) files should be copied to the collector machine.

3.3 Apache Configuration

You need to configure Apache for SSL. Here is what our configuration looks like:

```
SSLRandomSeed startup builtin
SSLRandomSeed startup file:/dev/random
SSLRandomSeed startup file:/dev/urandom 1024
SSLRandomSeed connect builtin
SSLRandomSeed connect file:/dev/random
SSLRandomSeed connect file:/dev/urandom 1024

<VirtualHost _default_:443>
DocumentRoot "/httpd/htdocs-ssl"
SSLEngine on
SSLCertificateFile /httpd/conf/SSL/server/server.crt
SSLCertificateKeyFile /httpd/conf/SSL/server/server.key
SSLCertificateChainFile /httpd/conf/SSL/cacert.pem

# For client-validation
SSLCACertificateFile /httpd/conf/SSL/cacert.pem
SSLVerifyClient require

SSLOptions +CompatEnvVars
Script PUT /cgi-bin/put-file.pl
</VirtualHost>
```

Note the last line of the configuration specifies the CGI script that accepts PUT requests. The *SSLOptions* line is necessary so that the CGI script receives certain HTTP headers as environment variables. Those headers/variables convey the X.509 information to the script so it knows where to store received XML files.

Chapter 4

Collector Installation

A collector machine needs only the *dsc* binary, a configuration file, and a couple of cron job scripts.

At this point, DSC lacks certain niceties such as a `./configure` script. The installation prefix, `/usr/local/dsc` is currently hard-coded.

4.1 Prerequisites

You'll need a C/C++ compiler to compile the `dsc` source code.

If the collector and archiver are different systems, you'll need a way to transfer data files. We recommend that you use the `curl` HTTP/SSL client. You may use another technique, such as `scp` or `rsync` if you prefer.

4.2 Installation

You can compile `dsc` from the `collector` directory:

```
% cd collector
% make
```

Assuming there are no errors or problems during compilation, install the `dsc` binary and other scripts with:

```
% make install
```

This installs five files:

```
/usr/local/dsc/bin/dsc
/usr/local/dsc/etc/dsc.conf.sample
/usr/local/dsc/libexec/upload-prep.sh
```

```
/usr/local/dsc/libexec/upload-rsync.sh
/usr/local/dsc/libexec/upload-x509.sh
```

Of course, if you don't want to use the default installation prefix, you can manually copy these files to a location of your choosing. If you do that, you'll also need to edit the cron scripts to match your choice of pathnames, etc.

4.3 Uploading XML Files

This section describes how XML files are transferred from the collector to one or more Presenter systems.

As we'll see in the next chapter, each `dsc` process has its own *run directory*. This is the directory where `dsc` leaves its XML files. It usually has a name like `/usr/local/dsc/run/NODENAME`. XML files are removed after they are successfully transferred. If the Presenter is unreachable, XML files accumulate here until they can be transferred. Make sure that you have enough disk space to queue a lot of XML files in the event of an outage.

In general we want to be able to upload XML files to multiple presenters. This is the reason behind the `upload-prep.sh` script. This script runs every 60 seconds from cron:

```
* * * * * /usr/local/dsc/libexec/upload-prep.sh
```

`upload-prep.sh` looks for `dsc.conf` files in `/usr/local/dsc/etc` by default. For each config file found, it `cd`'s to the *run_dir* and links¹ XML files to one or more upload directories. The upload directories are named `upload/dest1`, `upload/dest2`, and so on.

In order for all this to work, you must create the directories in advance. For example, if you are collecting stats on your nameserver named *ns0*, and want to send the XML files to two presenters (named *oarc* and *archive*), the directory structure might look like:

```
% set prefix=/usr/local/dsc
% mkdir $prefix/run
% mkdir $prefix/run/ns0
% mkdir $prefix/run/ns0/upload
% mkdir $prefix/run/ns0/upload/oarc
% mkdir $prefix/run/ns0/upload/archive
```

With that directory structure, the `upload-prep.sh` script moves XML files from the *ns0* directory to the two upload directories, *oarc* and *archive*.

To actually transfer files to the presenter, use either `upload-x509.sh` or `upload-rsync.sh`.

¹as in "hard link" made with `/bin/ln`.

4.3.1 upload-x509.sh

This cron script is responsible for actually transferring XML files from the upload directories to the remote server. It creates a *tar* archive of XML files and then uploads it to the remote server with *curl*. The script takes three commandline arguments:

```
% upload-x509.sh NODE DEST URI
```

NODE must match the name of a directory under */usr/local/dsc/run*. Similarly, *DEST* must match the name of a directory under */usr/local/dsc/run/NODE/upload*. *URI* is the URL/URI that the data is uploaded to. Usually it is just an HTTPS URL with the name of the destination server. We also recommend running this from cron every 60 seconds. For example:

```
* * * * * /usr/local/dsc/libexec/upload-x509.sh ns0 oarc \  
             https://collect.oarc.isc.org/  
* * * * * /usr/local/dsc/libexec/upload-x509.sh ns0 archive \  
             https://archive.example.com/
```

upload-x509.sh looks for X.509 certificates in */usr/local/dsc/certs*. The client certificate should be named */usr/local/dsc/certs/DEST/NODE.pem* and the CA certificate should be named */usr/local/dsc/certs/DEST/cacert.pem*. Note that *DEST* and *NODE* must match the *upload-x509.sh* command line arguments.

4.3.2 upload-rsync.sh

This script can be used to transfer XML files from the upload directories to the remote server. It uses *rsync* and assumes that *rsync* will use *ssh* for transport. This script also takes three arguments:

```
% upload-rsync.sh NODE DEST RSYNC-DEST
```

Note that *DEST* is the name of the local “upload” directory and *em RSYNC-DEST* is an *rsync* destination (i.e., hostname and remote directory). Here is how you might use it in a crontab:

```
* * * * * /usr/local/dsc/libexec/upload-rsync.sh ns0 oarc \  
             dsc@collect.oarc.isc.org:/usr/local/dsc/data/Server/ns0  
* * * * * /usr/local/dsc/libexec/upload-rsync.sh ns0 archive \  
             dsc@archive.oarc.isc.org:/usr/local/dsc/data/Server/ns0
```

rsync over *ssh* requires you to use RSA or DSA public keys that do not have a passphrase. If you do not want to use one of *ssh*'s default identity files, you can create one specifically for this script. It should be named **dsc_uploader_id** (and **dsc_uploader_id.pub**) in the `$HOME/.ssh` directory of the user that will be running the script. For example, you can create it with this command:

```
% ssh-keygen -t dsa -C dsc-uploader -f $HOME/.ssh/dsc_uploader_id
```

Then add **dsc_uploader_id.pub** to the **authorized_keys** file of the receiving userid on the presenter system.

Chapter 5

Configuring and Running the DSC Collector

5.1 dsc.conf

Before running **dsc** you need to create a configuration file. Note that configuration directive lines are terminated with a semi-colon. The configuration file currently understands the following directives:

local_address Specifies the DNS server's local IP address. It is used to determine the "direction" of an IP packet: sending, receiving, or other. You may specify multiple local addresses (separated by whitespace) if necessary.

Example: `local_address 172.16.0.1;`

run_dir A directory that should become **dsc**'s current directory after it starts. XML files will be written here, as will any core dumps.

Example: `run_dir "/var/run/dsc";`

bpf_program A Berkeley Packet Filter program string. Normally you should leave this unset. You may use this to further restrict the traffic seen by **dsc**. Note that **dsc** currently has one indexer that looks at all IP packets. If you specify something like *udp port 53* that indexer will not work.

However, if you want to monitor multiple DNS servers with separate DSC instances on one collector box, then you may need to use *bfp_program* to make sure that each **dsc** process sees only the traffic it should see.

Note that this directive must go before the *interface* directive because **dsc** makes only one pass through the configuration file and the BFP filter is set when the interface is initialized.

Example: `bpf_program "dst host 192.168.1.1";`

interface The interface name to sniff packets from. You may specify multiple interfaces.

Example: `interface fxp0;`

bpf_vlan_tag_byte_order `dsc` knows about VLAN tags. Some operating systems (FreeBSD-4.x) have a bug whereby the VLAN tag id is byte-swapped. Valid values for this directive are `host` and `net` (the default). Set this to `host` if you suspect your operating system has the VLAN tag byte order bug.

Example: `bpf_vlan_tag_byte_order host;`

match_vlan A list of VLAN identifiers (integers). If set, only the packets belonging to these VLANs are counted.

Example: `match_vlan 101 102;`

qname_filter This directive allows you to define custom filters to match query names in DNS messages. Please see Section 7.3.4 for more information.

dataset This directive is the heart of `DSC`. However, it is also the most complex. To save time we recommend that you copy interesting-looking dataset definitions from `dsc.conf.sample`. Comment out any that you feel are irrelevant or uninteresting. Later, as you become more familiar with `DSC`, you may want to read the next chapter and add your own custom datasets.

5.2 A Complete Sample `dsc.conf`

Here's how your entire `dsc.conf` file might look:

```
#bpf_program
interface em0;

local_address 192.5.5.241;

run_dir "/usr/local/dsc/run/foo";

dataset qtype dns All:null Qtype:qtype queries-only;
dataset rcode dns All:null Rcode:rcode replies-only;
dataset opcode dns All:null Opcode:opcode queries-only;
dataset rcode_vs_replylen dns Rcode:rcode ReplyLen:msglen replies-only;
dataset client_subnet dns All:null ClientSubnet:cip4_net queries-only
    max-cells=200;
dataset qtype_vs_qnamelen dns Qtype:qtype QnameLen:qnamelen queries-only;
dataset qtype_vs_tld dns Qtype:qtype TLD:tld queries-only,popular-qtypes
    max-cells=200;
dataset certain_qnames_vs_qtype dns CertainQnames:certain_qnames
    Qtype:qtype queries-only;
dataset client_subnet2 dns Class:query_classification
```

```

ClientSubnet:cip4_net queries-only max-cells=200;
dataset client_addr_vs_rcode dns Rcode:rcode ClientAddr:client
replies-only max-cells=50;
dataset chaos_types_and_names dns Qtype:qtype Qname:qname
chaos-class,queries-only;
dataset idn_qname dns All:null IDNQname:idn_qname queries-only;
dataset edns_version dns All:null EDNSVersion:edns_version queries-only;
dataset do_bit dns All:null DO:do_bit queries-only;
dataset rd_bit dns All:null RD:rd_bit queries-only;
dataset idn_vs_tld dns All:null TLD:tld queries-only,idn-only;
dataset ipv6_rsn_abusers dns All:null ClientAddr:client
queries-only,aaaa-or-a6-only,root-servers-n et-only max-cells=50;

dataset direction_vs_ipproto ip Direction:ip_direction IPProto:ip_proto
any;

```

5.3 Running dsc

dsc accepts a single command line argument, which is the name of the configuration file. For example:

```

% cd /usr/local/dsc
% bin/dsc etc/foo.conf

```

If you run ps when dsc is running, you'll see two processes:

```

60494 ?? S      0:00.36 bin/dsc etc/foo.conf
69453 ?? Ss    0:10.65 bin/dsc etc/foo.conf

```

The first process simply forks off child processes every 60 seconds. The child processes do the work of analyzing and tabulating DNS messages.

Please use NTP or another technique to keep the collector's clock synchronized to the correct time.

Chapter 6

Viewing DSC Graphs

To view DSC data in a web browser, simply enter the URL to the `dsc-grapher.pl` CGI. But before you do that, you'll need to create a grapher configuration file.

`dsc-grapher.pl` uses a simple configuration file to set certain menu options. This configuration file is `/usr/local/dsc/etc/dsc-grapher.cfg`. You should find a sample version in the same directory. For example:

```
server f-root pao1 sfo2
server isc senna+piquet
server tmf hq sc lgh
trace_windows 1hour 4hour 1day 1week 1month
accum_windows 1day 2days 3days 1week
```

Refer to Figure 6.1 to see how the directives affect the visual display. The configuration file has three directives:

server This directive tells `dsc-grapher.pl` to list the given server and its associated nodes in the “Servers/Nodes” section of its navigation menu. You can repeat this directive for each server that the Presenter has.

trace_windows Specifies the “Time Scale” menu options for trace-based plots.

accum_windows Specifies the “Time Scale” menu options for “cumulative” plots, such as the Classification plot.

timezone @@write me

domain_list @@write me

valid_domains @@write me

embargo @@write me

Servers/Nodes	
tmf	
› hq	
› sc	
› lgh	
isc	
f-root	
Plots	
Qtypes	
› DNSSEC Qtypes	
Rcodes	
Classification	
Client Geography	
TLDs	
Rcodes by Client Address	
Popular Names	
IPv6 root abusers	
Opcodes	
Query Attributes	
CHAOS	
IP Protocols	
Qname Length	
Reply Lengths	
Time Scale	
1hour	
4hour	
1day	
1week	
1month	
Y-Axis	
Query Rate (q/s)	
Percent of Queries	

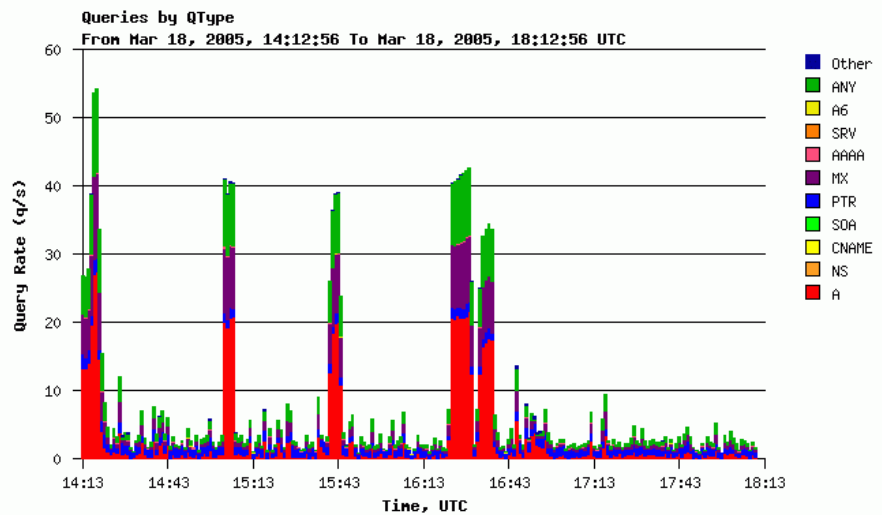


Figure 6.1: A sample graph

anonymize_ip @@write me

Note that the **dsc-grapher.cfg** only affects what may appear in the navigation window. It does NOT prevent users from entering other values in the URL parameters. For example, if you have data for a server/node in your **/usr/local/dsc/data/** directory that is not listed in **dsc-grapher.cfg**, a user may still be able to view that data by manually setting the URL query parameters.

The first few times you try **dsc-grapher.pl**, be sure to run **tail -f** on the HTTP server error.log file.

Chapter 7

DSC Datasets

A *dataset* is a 2-D array of counters. For example, you might have a dataset with “Query Type” along one dimension and “Query Name Length” on the other. The result is a table that shows the distribution of query name lengths for each query type. For example:

Len	A	AAAA	A6	PTR	NS	SOA
...						
11	14	8	7	11	2	0
12	19	2	3	19	4	1
...						
255	0	0	0	0	0	0

A dataset is defined by the following parameters:

- A name
- A protocol layer (IP or DNS)
- An indexer for the first dimension
- An indexer for the second dimension
- One or more filters
- Zero or more options and parameters

The *dataset* definition syntax in `dsc.conf` is:

```
dataset name protocol Label1:Indexer1 Label2:Indexer2 filter [parameters];
```

7.1 Dataset Name

The dataset name is used in the filename for `dsc`’s XML files. Although this is an opaque string in theory, the Presenter’s XML extractor routines must recognize

Indexer	Label	Description
ip_direction	Direction	one of sent, recv, or other
ip_proto	IPProto	IP protocol (icmp, tcp, udp)

Table 7.1: IP packet indexers

the dataset name to properly parse it. The source code file `presenter/perl1lib/DSC/extractor/config.pm` contains an entry for each known dataset name.

7.2 Protocol

DSC currently knows about two protocol layers: IP and DNS. On the `dataset` line they are written as `ip` and `dns`.

7.3 Indexers

An *indexer* is simply a function that transforms the attributes of an IP/DNS message into an array index. For some attributes the transformation is straightforward. For example, the “Query Type” indexer simply extracts the query type value from a DNS message and uses this 16-bit value as the array index.

Other attributes are slightly more complicated. For example, the “TLD” indexer extracts the TLD of the QNAME field of a DNS message and maps it to an integer. The indexer maintains a simple internal table of TLD-to-integer mappings. The actual integer values are unimportant because the TLD strings, not the integers, appear in the resulting XML data.

When you specify an indexer on a `dataset` line, you must provide both the name of the indexer and a label. The Label appears as an attribute in the XML output. For example, Figure 7.1 shows the XML corresponding to this *dataset* line:

```
dataset the_dataset dns Foo:foo Bar:bar queries-only;
```

In theory you are free to choose any label that you like, however, the XML extractors look for specific labels. Please use the labels given for the indexers in Tables 7.2 and 7.1.

7.3.1 IP Indexers

DSC includes only minimal support for collecting IP-layer stats. Mostly we are interested in finding out the mix of IP protocols recieved by the DNS server. It can also show us if/when the DNS server is the subject of denial-of-service attack. Table 7.1 shows the indexers for IP packets. Here are their longer descriptions:

```

<array name="the_dataset" dimensions="2" start_time="1091663940" ...
  <dimension number="1" type="Foo"/>
  <dimension number="2" type="Bar"/>
  <data>
    <Foo val="1">
      <Bar val="0" count="4"/>
      ...
      <Bar val="100" count="41"/>
    </Foo>
    <Foo val="2">
      ...
    </Foo>
  </data>
</array>

```

Figure 7.1: Sample XML output

ip_direction One of three values: sent, recv, or else. Direction is determined based on the setting for *local_address* in the configuration file.

ip_proto The IP protocol type, e.g.: tcp, udp, icmp. Note that the *bpf_program* setting affects all traffic seen by DSC. If the program contains the word “udp” then you won’t see any counts for non-UDP traffic.

7.3.2 IP Filters

Currently there is only one IP protocol filter: **any**. It includes all received packets.

7.3.3 DNS Indexers

Table 7.2 shows the currently-defined indexers for DNS messages, and here are their descriptions:

certain_qnames This indexer isolates the two most popular query names seen by DNS root servers: *localhost* and *[a-m].root-servers.net*.

cip4_net Groups DNS messages together by the /24 subnet of the client’s IPv4 address. We use this to make datasets with large, diverse client populations more manageable and to provide a small amount of privacy and anonymization.

cip4_addr The IPv4 address of the DNS client.

do_bit This indexer has only two values: 0 or 1. It indicates whether or not the “DO” bit is set in a DNS query. According to RFC 2335: *Setting the*

Indexer	Label	Description
certain_qnames	CertainQnames	Popular query names seen at roots
cip4_net	ClientSubnet	The client's IPv4 /24 subnet
client	ClientAddr	The client's IPv4 address
do_bit	DO	Whether the DO bit is on
edns_version	EDNSVersion	The EDNS version number
idn_qname	IDNQname	If the QNAME is in IDN format
msglen	MsgLen	The DNS message length
null	All	A “no-op” indexer
opcode	Opcode	DNS message opcode
qclass	-	Query class
qnamelen	QnameLen	Length of the query name
qtype	Qtype	DNS query type
query_classification	Class	A classification for bogus queries
rcode	Rcode	DNS reply code
rd_bit	RD	Check if Recursion Desired bit set
tld	TLD	TLD of the query name

Table 7.2: DNS message indexers

DO bit to one in a query indicates to the server that the resolver is able to accept DNSSEC security RRs.

edns_version The EDNS version number, if any, in a DNS query. EDNS Version 0 is documented in RFC 2671.

idn_qname This indexer has only two values: 0 or 1. It returns 1 when the first QNAME in the DNS message question section is an internationalized domain name (i.e., containing non-ASCII characters). Such QNAMEs begin with the string `xn--`. This convention is documented in RFC 3490.

msglen The overall length (size) of the DNS message.

null A “no-op” indexer that always returns the same value. This can be used to effectively turn the 2-D table into a 1-D array.

opcode The DNS message opcode is a four-bit field. QUERY is the most common opcode. Additional currently defined opcodes include: IQUERY, STATUS, NOTIFY, and UPDATE.

qclass The DNS message query class (QCLASS) is a 16-bit value. IN is the most common query class. Additional currently defined query class values include: CHAOS, HS, NONE, and ANY.

qname The full QNAME string from the first (and usually only) QNAME in the question section of a DNS message.

qnamelen The length of the first (and usually only) QNAME in a DNS message question section. Note this is the “expanded” length if the message happens to take advantage of DNS message “compression.”

qtype The query type (QTYPE) for the first QNAME in the DNS message question section. Well-known query types include: A, AAAA, A6, CNAME, PTR, MX, NS, SOA, and ANY.

query_classification A stateless classification of “bogus” queries:

- non-auth-tld: when the TLD is not one of the IANA-approved TLDs.
- root-servers.net: a query for a root server IP address.
- localhost: a query for the localhost IP address.
- a-for-root: an A query for the DNS root (.).
- a-for-a: an A query for an IPv4 address.
- rfc1918-ptr: a PTR query for an RFC 1918 address.
- funny-class: a query with an unknown/undefined query class.
- funny-qtype: a query with an unknown/undefined query type.
- src-port-zero: when the UDP message’s source port equals zero.
- malformed: a malformed DNS message that could not be entirely parsed.

rcode The RCODE value in a DNS reply. The most common response codes are 0 (NO ERROR) and 3 (NXDOMAIN).

rd_bit This indexer returns 1 if the RD (recursion desired) bit is set in the query. Usually only stub resolvers set the RD bit. Usually authoritative servers do not offer recursion to their clients.

tld the TLD of the first QNAME in a DNS message’s question section.

7.3.4 DNS Filters

You must specify one or more of the following filters (separated by commas) on the `dataset` line:

any The no-op filter, counts all messages.

queries-only Count only DNS query messages. A query is a DNS message where the QR bit is set to 0.

replies-only Count only DNS reply messages. A query is a DNS message where the QR bit is set to 1.

popular-qtypes Count only DNS messages where the query type is one of: A, NS, CNAME, SOA, PTR, MX, AAAA, A6, ANY.

idn-only Count only DNS messages where the query name is in the internationalized domain name format.

aaaa-or-a6-only Count only DNS Messages where the query type is AAAA or A6.

root-servers-net-only Count only DNS messages where the query name is within the *root-servers.net* domain.

chaos-class Counts only DNS messages where QCLASS is equal to CHAOS (3). The CHAOS class is generally used for only the special *hostname.bind* and *version.bind* queries.

Note that multiple filters are ANDed together. That is, they narrow the input stream, rather than broaden it.

In addition to these pre-defined filters, you can add your own custom filters.

qname_filter

The *qname_filter* directive defines a new filter that uses regular expression matching on the QNAME field of a DNS message. This may be useful if you have a server that is authoritative for a number of zones, but you want to limit your measurements to a small subset. The *qname_filter* directive takes two arguments: a name for the filter and a regular expression. For example:

```
qname_filter MyFilterName example\.(com|net|org)$ ;
```

This filter matches queries (and responses) for names ending with *example.com*, *example.net*, and *example.org*. You can reference the named filter in the filters part of a *dataset* line. For example:

```
dataset qtype dns All:null Qtype:qtype queries-only,MyFilterName;
```

7.3.5 Parameters

dsc currently supports the following optional parameters:

min-count=NN Cells with counts less than *NN* are not included in the output. Instead, they are aggregated into the special values **-:SKIPPED:-** and **-:SKIPPED_SUM:-**. This helps reduce the size of datasets with a large number of small counts.

max-cells=NN A different, perhaps better, way of limiting the size of a dataset. Instead of trying to determine an appropriate *min-count* value in advance, *max-cells* allows you put a limit on the number of cells to include for the second dataset dimension. If the dataset has 9 possible

first-dimension values, and you specify a *max-cell* count of 100, then the dataset will not have more than 900 total values. The cell values are sorted and the top *max-cell* values are output. Values that fall below the limit are aggregated into the special `-:SKIPPED:-` and `-:SKIPPED_SUM:-` entries.

Chapter 8

Data Storage

8.1 XML Structure

A dataset XML file has the following structure:

```
<array name="dataset-name" dimensions="2" start_time="unix-seconds"
      stop_time="unix-seconds">
  <dimension number="1" type="Label1"/>
  <dimension number="2" type="Label2"/>
  <data>
    <Label1 val="D1-V1">
      <Label2 val="D2-V1" count="N1"/>
      <Label2 val="D2-V2" count="N2"/>
      <Label2 val="D2-V3" count="N3"/>
    </Label1>
    <Label1 val="D1-V2">
      <Label2 val="D2-V1" count="N1"/>
      <Label2 val="D2-V2" count="N2"/>
      <Label2 val="D2-V3" count="N3"/>
    </Label1>
  </data>
</array>
```

dataset-name, *Label1*, and *Label2* come from the dataset definition in *dsc.conf*.

The *start_time* and *stop_time* attributes are given in Unix seconds. They are normally 60-seconds apart. *dsc* usually starts a new measurement interval on 60 second boundaries. That is:

$$stop_time \bmod 60 == 0 \quad (8.1)$$

The LABEL1 VAL attributes (*D1-V1*, *D1-V2*, etc) are values for the first dimension indexer. Similarly, the LABEL2 VAL attributes (*D2-V1*, *D2-V2*, *D2-V3*) are values for the second dimension indexer. For some indexers these

values are numeric, for others they are strings. If the value contains certain non-printable characters, the string is base64-encoded and the optional `BASE64` attribute is set to 1.

There are two special VALs that help keep large datasets down to a reasonable size: `-:SKIPPED:-` and `-:SKIPPED_SUM:-`. These may be present on datasets that use the *min-count* and *max-cells* parameters (see Section 7.3.5). `-:SKIPPED:-` is the number of cells that were not included in the XML output. `-:SKIPPED_SUM:-`, on the other hand, is the sum of the counts for all the skipped cells.

Note that “one-dimensional datasets” still use two dimensions in the XML file. The first dimension type and value will be “All”, as shown in the example below.

The *count* values are always integers. If the count for a particular tuple is zero, it should not be included in the XML file.

Note that the contents of the XML file do not indicate where it came from. In particular, the server and node that it came from are not present. Instead, DSC relies on the presenter to store XML files in a directory hierarchy with the server and node as directory names.

Here is a short sample XML file with real content:

```
<array name="rcode" dimensions="2" start_time="1154649600"
      stop_time="1154649660">
  <dimension number="1" type="All"/>
  <dimension number="2" type="Rcode"/>
  <data>
    <All val="ALL">
      <Rcode val="0" count="70945"/>
      <Rcode val="3" count="50586"/>
      <Rcode val="4" count="121"/>
      <Rcode val="1" count="56"/>
      <Rcode val="5" count="44"/>
    </All>
  </data>
</array>
```

Please see <http://dns.measurement-factory.com/tools/dsc/sample-xml/> for more sample XML files.

The XML is not very strict and might cause XML purists to cringe. `dsc` writes the XML files the old-fashioned way (with `printf()`) and reads them with Perl’s `XML::Simple` module. Here is a possibly-valid DTD for the dataset XML format. Note, however, that the *LABEL1* and *LABEL2* strings are different for each dataset:

```
<!DOCTYPE ARRAY [
<!--
  <!-- ELEMENT ARRAY (DIMENSION+, DATA)-->
-->
```

```
1093219980 root-servers.net 122 rfc1918-ptr 112 a-for-a 926 funny-qclass 16
1093220040 root-servers.net 121 rfc1918-ptr 104 a-for-a 905 funny-qclass 15
1093220100 root-servers.net 137 rfc1918-ptr 116 a-for-a 871 funny-qclass 12
```

8.2.2 Format 2

```
time j1 k1:Nj1,k1:k2:Nj1,k2:... j2 k1:Nj2,k1:k2:Nj2,k2:... ...
```

This is a two-dimensional time-series format. In the above, j represents the first dimension indexer and k represents the second. Key-value pairs for the second dimension are separated by colons, rather than space. For example:

```
1093220160 recv icmp:2397:udp:136712:tcp:428 sent icmp:819:udp:119191:tcp:323
1093220220 recv icmp:2229:udp:124708:tcp:495 sent icmp:716:udp:107652:tcp:350
1093220280 recv udp:138212:icmp:2342:tcp:499 sent udp:120788:icmp:819:tcp:364
1093220340 recv icmp:2285:udp:137107:tcp:468 sent icmp:733:udp:118522:tcp:341
```

8.2.3 Format 3

k N_k

This format is used for one-dimensional datasets where the key space is (potentially) very large. That is, putting all the key-value pairs on a single line would result in a very long line in the datafile. Furthermore, for these larger datasets, it is prohibitive to store the data as a time series. Instead the counters are incremented over time. For example:

```
10.0.160.0 3024
10.0.20.0 92
10.0.244.0 5934
```

8.2.4 Format 4

j k $N_{j,k}$

This format is used for two-dimensional datasets where one or both key spaces are very large. Again, counters are incremented over time, rather than storing the data as a time series. For example:

```
10.0.0.0 non-auth-tld 105
10.0.0.0 ok 37383
10.0.0.0 rfc1918-ptr 5941
10.0.0.0 root-servers.net 1872
10.0.1.0 a-for-a 6
10.0.1.0 non-auth-tld 363
10.0.1.0 ok 144
```

Chapter 9

Bugs

- Seems too confusing to have an opaque name for indexers in `dsc.conf` dataset line. The names are pre-determined anyway since they must match what the XML extractors look for.
- Also stupid to have indexer names and a separate “Label” for the XML file.
- DSC perl modules are installed in the “site_perl” directory but they should probably be installed under `/usr/local/dsc`.
- DSC collector does not grok DNS/TCP transactions yet.
- DSC collector silently drops UDP frags