Network Improvement Through Flow Analysis

or, how to know what your network *really* did without going broke

Michael W. Lucas
mwlucas@blackhelicopters.org

Obligatory Tax-Deducting Content

Based on the forthcoming book

Network Flow Analysis: Assessment, Archaeology, and Awareness

by Michael W Lucas, November 2009

What We'll Cover

Network Awareness Tools
Netflow Basics
Netflow Architecture
Implementation
Flow Records
Pretty Reports
Ugly Reports
Real World Problems

What We Won't Cover

- Expensive Tools
- Argus
- RMON

- Advanced Netflow

What is the Problem?

- The network is more and more important
- The network is invisible
- It's easy to blame the network
- Ignorance of TCP/IP is rampant
- “The T1 is full? Full of what?”
- “What broke?”

“Absence of evidence” != “evidence of absence”

Network Awareness Tools

- MRTG/Cricket/Cacti
- RTG
- Random SNMP Cruft
- Nagios/BigBrother
- CiscoWorks/SolarWinds/OpenView

What Is a Flow?

- “A unidirectional sequence of packets all sharing the same source and destination IP address, source and destination port, and IP protocol.”
- Each TCP connection is two or more flows, at least one in each direction.
- Also known as “session-level data.”

What is Flow Analysis?

- Collecting, managing, and reporting on flow records.
- Flow analysis system components can be mixed-and-matched as desired to fit the environment and equipment.

Flows versus Netflow

- “Flow Export” or “Flow Analysis” – generic term
- “Netflow” – owned by Cisco

What Flows Give You

- Who talked to who?
- How much was said?
- What port did they talk on?
- What protocol did they use?
- What TCP/IP flags did they use?
- All the history you have disk for

What Flows Don't Give You

- The contents of the conversation
- You can see that you transferred 10kB out via port 80, but you cannot verify that the traffic was HTTP
- Alarms (at least, not directly)

Flow System Architecture

- Sensor(s)
- Collector(s)
- Analysis System(s)

These can be on the same system, or all on different systems, depending on available hardware and network architecture.

the sensor

- A program or device that sniffs the network
- Aggregates and condenses network information into flow records
- security-sensitive device
- very few system resources required
- flow records are flung across the network
- sensitive to network loss

the collector

- The software that catches records flung by the collector
- Stores the flows in files on disk
- Managing those flow files is your problem, not the collector software's.
- 5Mb/s uses 2GB disk/month

the reporting system

- Data files are nice, but you need to get information out of them.
- Choice of graphical or text reports
- Must have flexibility to transform and aggregate data in any way you desire.

Supported Operating Systems

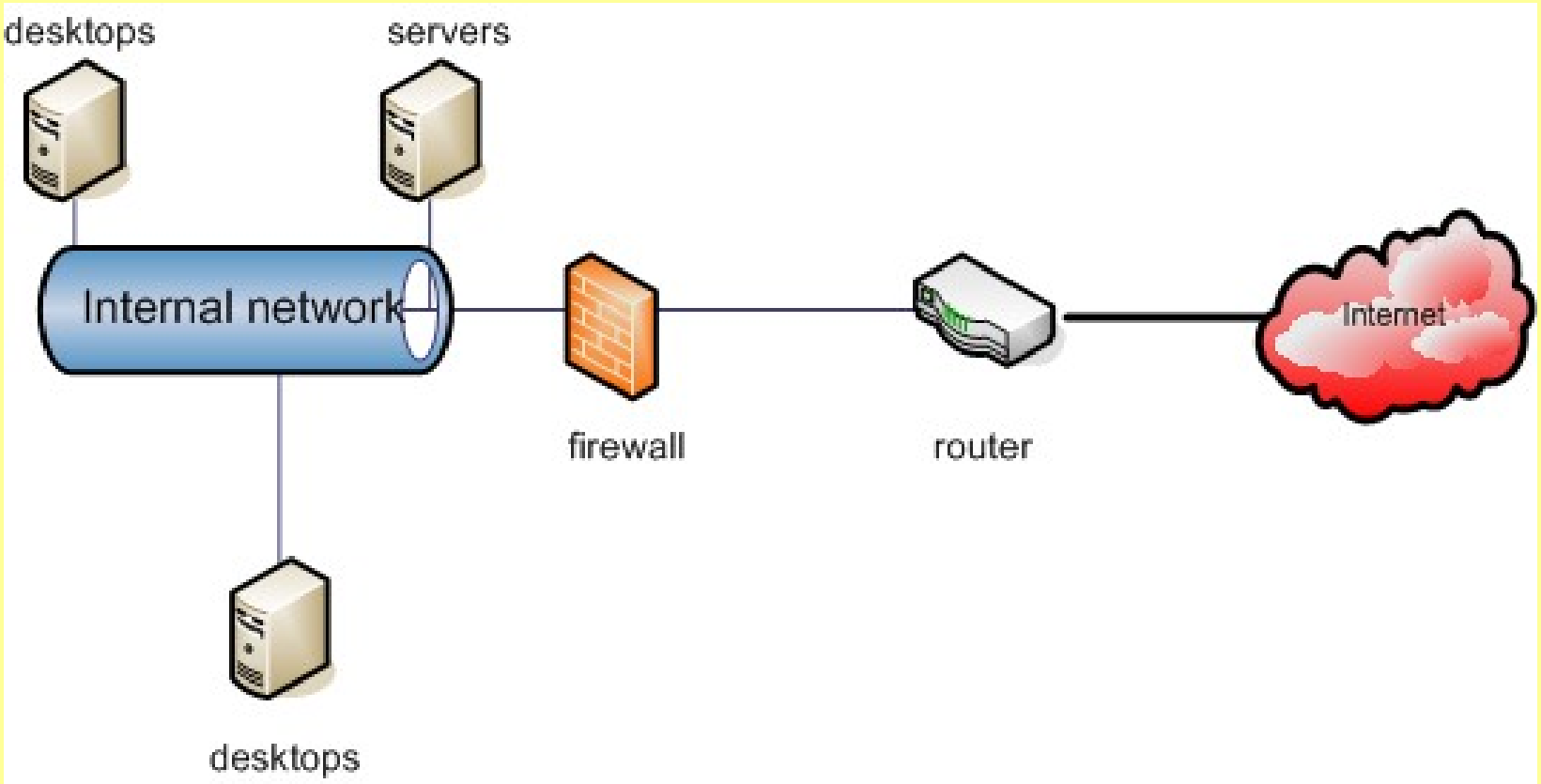
- UNIX-like OS is least expensive
- Windows options are available, but cost $$$
- We're going to focus on free UNIX
- Reference platform is FreeBSD
- 100% NetBSD/OpenBSD compatible
- Also works on lesser Unixen (Linux, Solaris, blah blah blah)

Where to place each component?

- Sensor in a place you want to collect data from
- Collector in a secure environment
- Reporter only needs read access to collector files

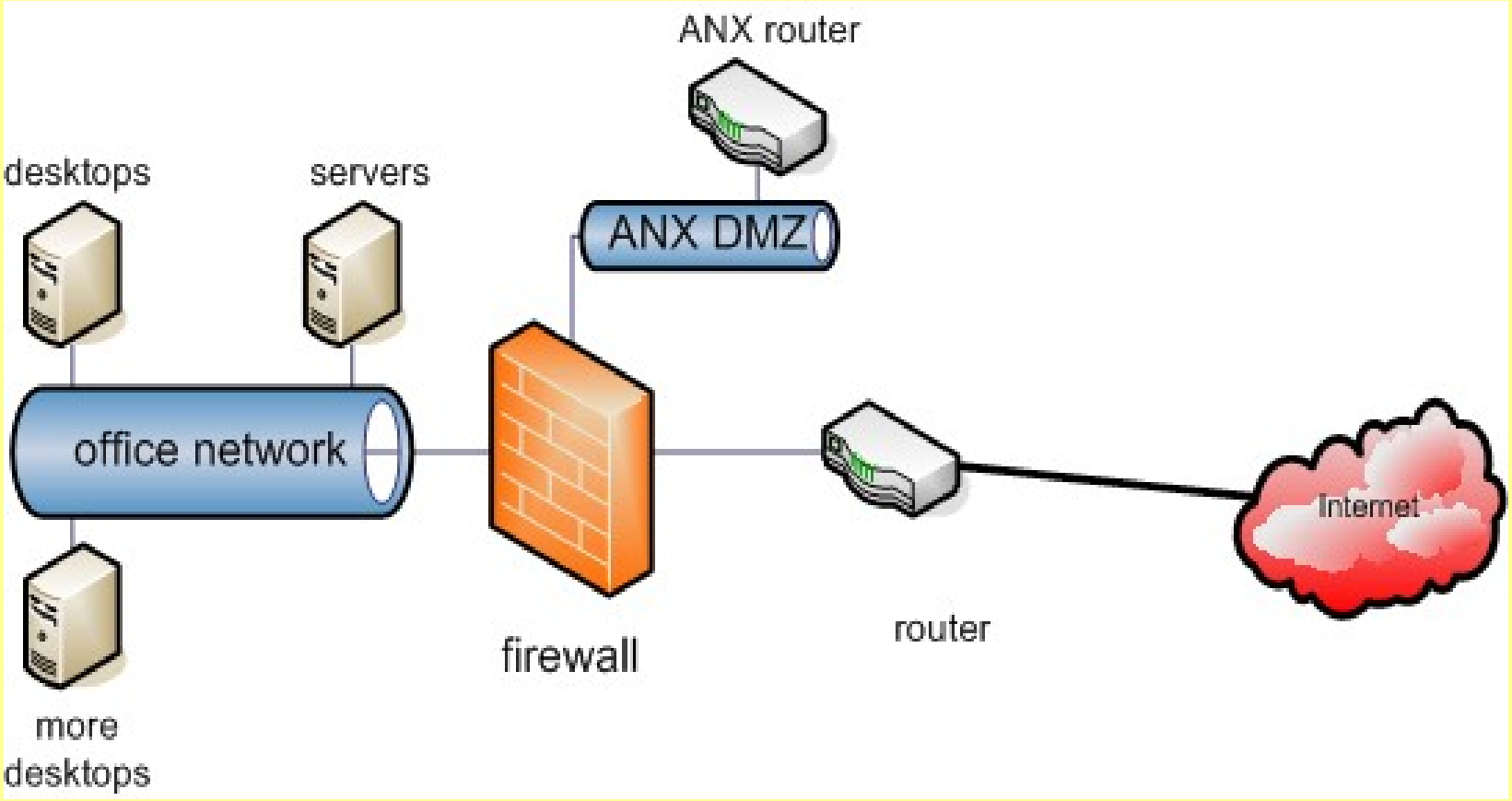
Simple Network

Two sensible collector points: inside & outside

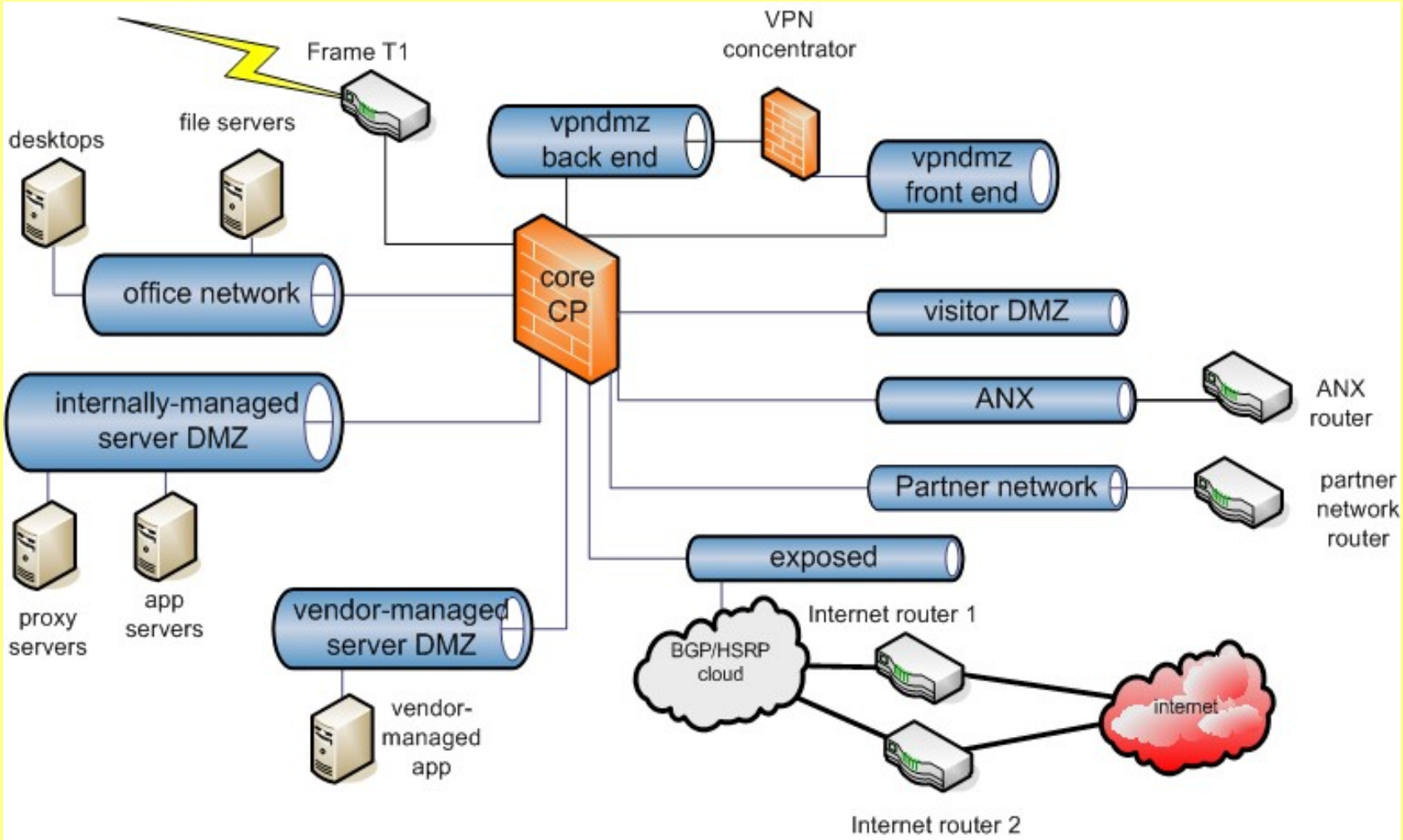


Larger Network

Lots of places for sensors



Scary Network



Implementing the Sensor

- Choose your platform first, based on what you have
- Some Cisco switches speak Netflow natively
- Most Cisco routers speak Netflow natively
- Juniper speaks cflowd, aka Netflow v5
- Implement on UNIX via softflowd

Sampling vs Capture

- Sampling: grabbing a small percentage of all traffic. Examples includes many Cisco devices, SFlow, etc
  - Advantages: easy on system resources
  - Disadvantages: incomplete data, must extrapolate
- Capturing: pull all traffic
  - Advantages: you know exactly what happened
  - Disadvantages: harder on system resources

Taps vs. Monitor

- Tap: Hardware Ethernet signal replicator
  - hardware, lives in-line
- Monitor: sniffer port on switch
  - easy
  - limited by hardware

Available Sensor Software

- FreeBSD: ng_netflow
- OpenBSD: pfflowd
- softflowd: http://www.mindrot.org/softflowd.html

Cisco IOS

- Cisco routers and 4000+ switches can export Netflow.

```
# conf t
# int s0/0
#   ip route-cache flow
#   exit
# ip flow-export collector-ip collector-port
```

- Netflow support varies *widely* by model
- Flow data taken from switching hardware

Cisco IOS #2

- Or, if you're exporting from a switch, try this:

```
# conf t
# ip route-cache flow
# ip flow ingress
# ip flow ingress layer2-switched
# ip flow-export version 5
# ip flow-export destination hostname port
```

softflowd

- Lets you manage flows more easily
- uses cheap server horsepower instead of expensive router horsepower
- author very responsive to bug reports
- BSD license -- bury it in your products
- http://www.mindrot.org/softflowd.html

installing softflowd

download, extract, and run

```
# ./configure
# make
# make install
```

requires libpcap on Linux and Solaris
respects $PREFIX

What Softflowd Installs

- softflowd binary
- softflowctl binary
- softflowd(8)
- softflowctl(8)
- share/doc/softflowd/README

running softflowd

 # softflowd -i *interfacename* -n *host:port*

useful options:

- -t timeout
  suggest using -t maxlife=5m, for reasons to be explained later
- -m max-flows
  defaults to 8192, or 800K
- see softflowd(8) for more

flow timeouts

- In a perfect world, all flows are recorded when last FIN_ACK received.
- The world is not perfect.
- Recommend a 5-minute timeout for RRD-based tools (CUFlow.pm).
- If you're not generating pretty graphs with hacked-together tools, you can let flows run as long as you like.

watching/controlling softflowd

softflowctl(8) is the softflowd management program

softflowctl ***command***

Useful commands include: **shutdown** (graceful), **exit** (not graceful), **dump-flows** (print to stdout), **expire-all**

softflowctl statistics part 1

```
Number of active flows: 931
Packets processed: 331397135
Fragments: 182973
Ignored packets: 7971877 (7971877 non-IP, 0 too short)
Flows expired: 4110389 (0 forced)
Flows exported: 8220778 in 292488 packets (0 failures)
 Packets received by libpcap: 331397135
 Packets dropped by libpcap: 0
 Packets dropped by interface: 0

Expired flow statistics:  minimum        average       maximum
  Flow bytes:                  46          46906    2280684986
  Flow packets:                 1             80       4345644
  Duration:                 0.00s          16.83s     604815.56s
```

softflowctl statistics, part 2

```
Expired flow reasons:
       tcp =     99705   tcp.rst =    366531   tcp.fin =    694735
       udp =   2923234      icmp =     25869   general =       305
   maxlife =         2
  over 2Gb =         8
  maxflows =         0
   flushed =         0
```

softflowctl statistics, part 3

```
Per-protocol statistics:     Octets      Packets   Avg Life    Max Life
          icmp (1):       13516208       167299      84.00s   25940.57s
          igmp (2):           2306           50    3043.94s    7017.85s
           tcp (6):   145305156006    199306901      30.07s  604815.56s
          udp (17):    45176433183    123235886      10.39s  604803.24s
         rsvp (46):          11448           53     224.09s    1592.06s
          esp (50):     2308692158      5139749    5918.22s   51528.69s
```

suspending collection

If you think you need to do this, you're solving the wrong problem

softflowctl stop-gather and *start-gather* will let you suspend softflowd without shutting it down.

viewing cache

softflowctl dump-flows will show everything softflowd is tracking right now, but has not yet exported

Grep for IP addresses or ports of interest; output is ghastly.

flushing cache

- Dump everything into the collector as quickly as possible
- Normally you'd want to shutdown, but presumably you have your reasons

 # softflowctl expire-all

shutting down

- Expire everything and close down with:

softflowctl shutdown

- Throw away all remaining data and die immediately with:

softflowctl exit

implementing the collector

- The sensor throws stuff at the net, the collector catches it and stores it to disk
- Collector and reporting system are linked
- Need a secure system!
- Many flow collectors available:
  - flowd (developed by softflowd developer)
  - cflowd (obsolete)
  - flow-capture (part of flow-tools)

tcpdump or ethereal

- Question #1: “is my sensor data reaching my collector?”
- Check and find out!

- do not need promiscuous mode

 # tpcdump -p -i *interface* port *portnum*

flow-tools history

- For many years, the standard toolkit
- Was neglected, has recently risen from the dead
- Original version is not 64-bit clean, new version is OK
- New version had rough start, but is now stable

Which version?

- Old version: http://www.splintered.net/sw/flow-tools/
- New version: http://code.google.com/p/flow-tools/

install flow-tools

- Both tools install the same way

```
# ./configure
# gmake
# gmake install
```

- MySQL and Postgresql support optional

Cflowd Warning

- Cflowd is the primary predecessor of flow-tools.
- It is obsolete, deprecated, and referred to in many online documents.
- Cflowd rquires gcc 2.9, does not build easily on amd64
- You do not need Cflowd.
- You do need Cflow.pm, a different tool!
- May have to hack package management or build system

Databases and flow-capture

- Seems like a great idea, doesn't it?
- Lots of people thought that, too!
- Drawbacks:
  getting the data out
  write your own reports
  not as great as it seems
  please feel free to get involved and solve the problems, you'd delight lots of people
  Flow-based databases (e.g., Stager) have special programs to feed flow data into the database

run flow-capture

flow-capture -p /var/run/flow-capture.pid -n 287
-w /var/db/flows/ -S 5 127.0.0.1/0/5678

- -p == pidfile
- -n == number of rotations per day
- -w == log directory
- -S == log announcements
- *listening on/accepting from/port*

flow-capture log files

- directories: *$LOGDIR/yyyy/yyyy-mm/yyyy-mm-dd/*
- ft-* files are old, no longer being written to
- tmp-* files are currently being written to
- file format includes netflow version, timestamp file began, and offset from GMT.

{tmp|ft}-v05.2008-09-05.134501-0400

If files have size, flow-capture is working.

collecting from multiple sensors

We've seen how to build a collector for flows from one sensor. What do you do with multiple sensors?

- dump them all in one collector?
- build separate collectors for each?

Much like centralized logging, each has pros and cons. I recommend separate collectors.

Implementing Reporting

- Lots available, but most extensive by far is flow-tools with Cflow.pm
- Cflow.pm is older, but lots of tools built around it
- flow-tools is newer, has legacy glue for cflow
- integrated with flow-capture

Components we'll use

- flow-tools
- flowscan -- built on Cflow.pm

- First challenge: installing Cflow.pm with flow-tools hooks

Cflow.pm Headaches

- Everything before Cflow.pm is easy
- Everything after Cflow.pm is easy...
- ...once Cflow.pm works
- Cflow.pm installation is inconsistent across platforms
- Don't assume Cflow.pm works -- test it *immediately* upon installation!
- Do not proceed until Cflow.pm works.

Cflow.pm Install Method #1

The kiss of death:

```
"Note (probably harmless): No
  library found for -lnsl"
```

Try standard from-source install or local package manager, it might work.

Cflow.pm Install Method #2

- Go to directory where you built flow-tools
- In contrib directory, extract Cflow-*.tar.gz
- Build in this directory, see if it picks up libft.a
- If not, your system has let you down again...

Cflow.pm Install method #3

Search Cflow.pm's Makefile.PL. Find:

```
if (-f '../../lib/libft.a') {
```

Hard-code that path!

```
if (-f '/usr/local/lib/libft.a') {
```

testing with flowdumper

Cflow.pm includes flowdumper(1), a simple flow file query tool. If flowdumper works, you have successfully installed Cflow.pm and may continue

```
# flowdumper -s ft-your-flow-file-here
2005/04/28 19:14:01 172.16.30.247.80 ->
  216.98.200.250.63647 6(SYN|ACK) 3 144
 2005/04/28 19:14:01 216.98.200.250.63647 ->
  172.16.30.247.80 6(SYN) 1 48
...
```

Cflow.pm Programming

```
#!/usr/bin/perl

use Cflow qw(:flowvars find);
find (\&wanted, @ARGV);

sub wanted {
    return unless (($srcport == 500 && $dstport
== 500 ) && $udp == $protocol);

    printf("%s %15.15s.%-5hu %15.15s.%-5hu %2hu
%10u %10u\n",$localtime, $srcip, $srcport,
$dstip,$dstport, $protocol, $pkts, $bytes)

}
```

Once per file with perfile{}

```
#!/usr/bin/perl
use Cflow qw(:flowvars find );
find (\&wanted, \&perfile, @ARGV);

sub wanted {
}

sub perfile {
    print "working on \"$_[0]\"...\n";
}
```

returns matching rows

```
#!/usr/bin/perl
use Cflow qw(find :flowvars );
$hitrate = find (\&wanted, @ARGV);

sub wanted {
    return unless (1 == $protocol);
    $icmp++;
}

print "hitrate is $hitrate\n";
```

flow versions

Flow records come in many different versions:

- v1: better than nothing
- v5: today's lowest common denominator
- v7: switching information
- v9: sings, dances, does the dishes

You don't care which one you have, so long as you have it.

Why you care

Higher versions include more information

Netflow v9 has a whole different architecture built for IPv6, but doesn't work with flow-tools

Why you don't care

They all contain the same basics, however:

- who
- talked to who
- on what port
- how much was said
- TCP/IP flags

Information common to all versions

```
router:          172.16.20.1       The sensor or softflowd server
src IP:          192.168.1.54      Source of the flow
 dst IP:          10.0.8.3         Flow Destination
 input ifIndex:   0                router input interface
 output ifIndex:  0                router output interface
 src port:        61521            TCP/IP source port
 dst port:        443              TCP/IP destination port
 pkts:            10               number of packets
 bytes:           3015             size of this flow
 start time:      Wed Jul  6 10:47:29 2005
 end time:        Wed Jul  6 10:48:32 2005
 protocol:        6                from /etc/protocols
 tos:             0x0              TOS flags set in flow
 src AS:          0                if BGP-speaking router
 dst AS:          0                if BGP-speaking router
 src masklen:     0                if BGP-speaking router
 dst masklen:     0                if BGP-speaking router
 TCP flags:       0x1e (PUSH|SYN|ACK|RST)
```

Pretty Reports with Flowscan

- Management likes pretty pictures
- Good for a general overview, not so good for detailed analysis
- Web-based CGI

http://net.doit.wisc.edu/~plonka/FlowScan/

Flowscan Basics

- Provides overview of network traffic contents suitable for presenting to users
- requires pdksh, rrdtool, Cflow.pm, and assorted Perl modules
- installs easily via package management
- has hooks for report modules

Flowscan weaknesses

- based on RRDTool
  - historical analysis problematic due to compression of older data
- Difficult to add new protocols later
- complex to set up and debug
- default modules are hard to configure

Flowscan modules

- Flowscan is a engine for converting flow files into RRD databases.
- Other people have written modules to display in different formats.
- Original flowscan modules aren't bad, but there are much better available.

Install FlowScan

```
# cd FlowScan-1.006
# ./configure --prefix=/var/db/flows/router1
# make install
```

- creates “bin” and “images” subdirectory
- also create flowscandata and flowscanrrd

```
# chown -R flowscan:flowscan bin
# chmod g+w bin
# chown flowscan:flowscan flowscandata
flowscanrrd
# chmod g+ws flowscandata/ flowscanrrd/
```

Flowscan Setup 1

- Main config files are in $PREFIX/var/db/flows
- Two subdirectories, bin and graphs
- ignore graphs, we'll do better graphs later
- FlowScan.pm in bin directory is *obsolete*. Use FlowScan.pm 1.6, available at `http://net.doit.wisc.edu/~plonka/list/flowscan/archive/att-0848/01-FlowScan.pm`
- copy into flows/bin directory

Flowscan Setup 2

- copy flowscan.cf.sample to flowscan.cf and set:

```
FlowFileGlob /path/to/logs/ft-v*[0-9]
```

regex for log files

```
ReportClasses CUFlow
```

Flowscan modules we're using

Flowscan Setup 3

`WaitSeconds 300`

Leave this alone!

`Verbose 1`

Controls logging chattiness; good for setup

CUFlow: the generic pretty picture Flowscan module

CUFlow is much more configurable than Flowscan's default modules, but not as flexible.

`http://www.columbia.edu/acis/networks/advanced/CUFlow/`

Grab the tarball.

You must install and configure Flowscan first!

Configuring CUFlow 1

- Copy CUFlow.cf and CUFlow.pm to /var/db/flows/bin
- Edit CUFlow.cf to fit your environment

```
Subnet 192.168.2/23, 172.16.8/24
```

This is your sensor's complete IP range

```
OutputDir /var/log/cuflow
```

The place where CUFlow dumps log files

Configuring CUFlow 2

```
Network 192.168.2.3,192.168.2.5 webservers
Network 192.168.2.9,172.16.8.4  mailservers
Network 192.168.2.0/25          infrastructure
Network 192.168.2.128/25        dmz
Network 192.168.3.0/25          finance
Network 192.168.3.128/25        developers
```

Anything we want separate records on is a network

Configuring CUFlow 3

"Top X Talkers" scoreboard:

Top 20 below

old reports in /usr/local/www/flows/topten

current report is 2nd argument

```
Scoreboard 20 /usr/local/www/flows/topten /
  usr/local/www/flows/topten.html
```

Aggregate Over time

```
AggregateScore 20 /var/log/cflow/agg.dat
  /usr/local/www/flows/overall.html
```

Configuring CUFlow 4

Netflow Exporters

```
router 192.168.1.1 router1
router 192.168.1.2 router2
```

Particular services we want to track

```
Service 20-21/tcp ftp
Service 53/udp, 53/tcp dns
Service 80/tcp http
Service 4661-4662/tcp,4665/udp edonkey
```

Configuring CUFlow 5

Tracking per-protocol

```
protocol 1 icmp
protocol 6 tcp
protocol 17 udp
protocol 47 gre
```

ToS

```
TOS 0 normal
TOS 1 special
TOS 2-255 other
```

Configuring CUFlow 6

Traffic to or from a particular AS number
- next hop only

```
ASNumber 1 Level3
ASNumber 14246 GKNDNA
ASNumber 701 UUNet
```

Each of these produces its own set of RRD data

Moving Flow Files

- Must copy flow files flowscandata directory
- Use a rotation script

```
# flow-capture -p /var/run/flow-capture.pid
-R /usr/local/bin/flow-rotate.sh -n 287 -w
/var/db/flows -S 5
10.10.10.10/172.16.16.16/5678
```

- script is called each time flow-capture creates a new log file

Flowscan Rotation Script

- script will get one argument, the full path and name of the old flow file

```
#!/bin/sh

cp $1 /var/db/flows/test/flowscandata/
```

- verify files are copied, then start FlowScan

Running Flowscan

- Startup scripts are included in flowscan source/rc/{linux|solaris}
- Most package management systems install appropriate variant
- Configurable variables at top: log file, flow directory, installation directory
- Start it up, read your log files, see what happens. Should process all of your old files quickly and go to sleep for 300 seconds

Displaying with CUGrapher.pl

Log directory

```
my $rrddir = "/var/log/cuflows";
```

Hours displayed in CGI

```
my $hours = 48;
```

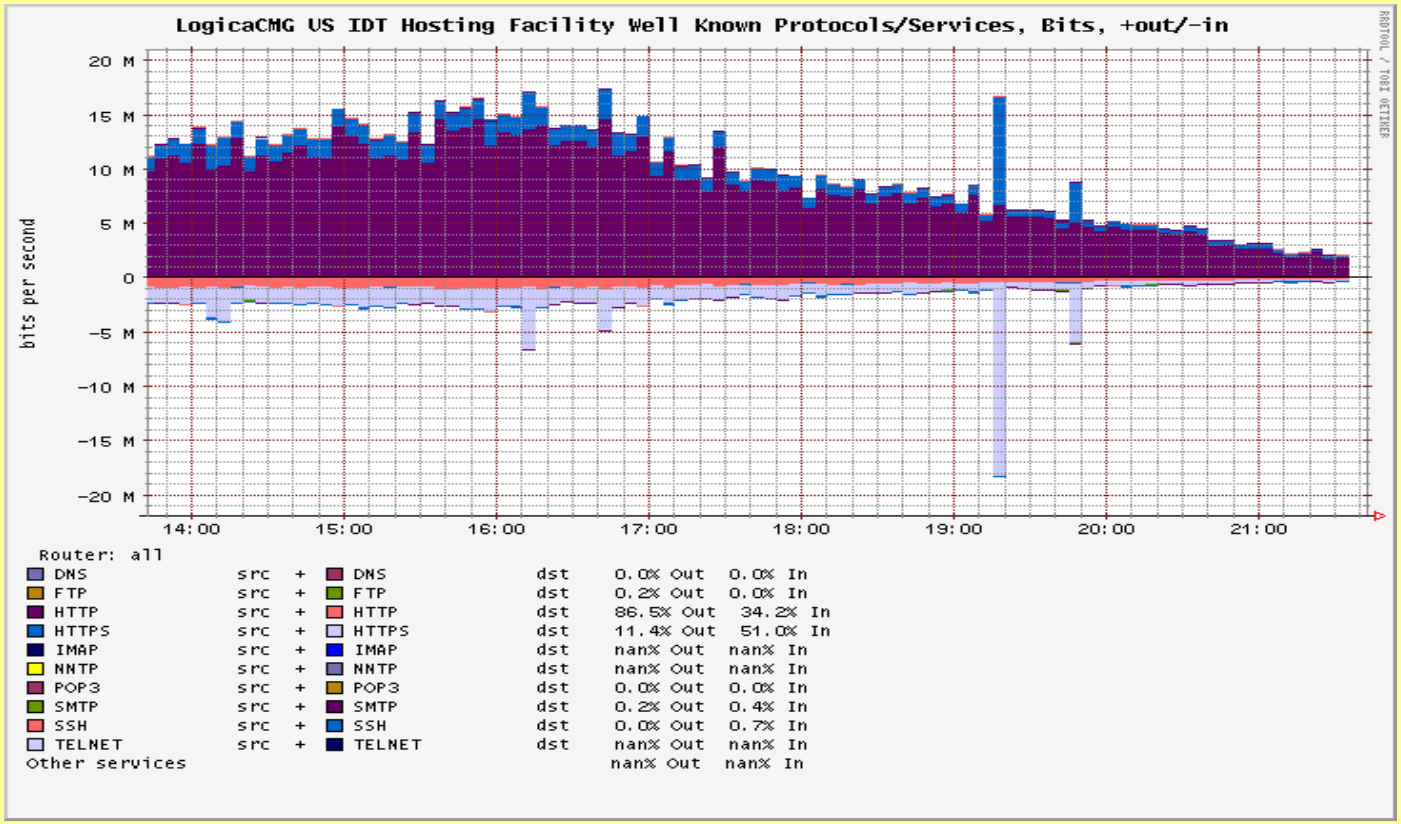
Organization at top of graphs

```
my $organization = "Minions of Lucas";
```

Graph dimensions & type

```
my $width = 640;
my $height = 320;
my $imageType = 'png';
```

Why bother with CUFlow?



It's the quickest route to pretty pictures.

CUGrapher Problems

- RRD versus long flows
  - Some flows last longer than five minutes.
- Identifying protocols by port
- People think that they know what the graphs mean

Ugly Reports with Flow-Tools

- Use flow-cat to open raw (binary) flow files
- flow-print prints them

```
# flow-cat ft-*| flow-print
```

- Default format calls bytes “octets”.

flow-print output

Default Format:

```
srcIP            dstIP            prot  srcPort  dstPort  octets      packets
63.85.32.4       208.109.209.156  6     58943    80       1145        12
208.109.209.156  63.85.32.4       6     80       58943    12081       12
...
```

-f 1 Format:

```
Sif  SrcIPaddress     DIf  DstIPaddress      Pr SrcP DstP  Pkts  Octets
 StartTime          EndTime             Active   B/Pk Ts Fl

0000 63.85.32.4       0000 208.109.209.156   06 e63f 50    12         1145
 1016.12:09:58.042  1016.12:09:58.995      0.953 95  00 1b

0000 208.109.209.156  0000 63.85.32.4        06 50   e63f  12         12081
 1016.12:09:58.042  1016.12:09:58.995      0.953 1006 00 1b
...
```

Printing Interface Info

- Use -f flag to choose flow-print output format.
- hardware flow sensors include interface info, view with -f 0

```
# flow-cat ft-v05.2008-12-01.171500-0500 | flow-print -f 0 | less
Sif  SrcIPaddress     Dif  DstIPaddress      Pr SrcP DstP  Pkts       Octets
0000 36.85.32.9       0000 158.43.192.1      11 915  35    1          59
0000 158.43.192.1     0000 36.85.32.9        11 35   915   1          134
0000 36.85.32.37      0000 83.243.35.204     06 19   1013  14         1035
0000 83.243.35.204    0000 36.85.32.37       06 1013 19    12         1527
…
```

- ports are now in hex, useful for ICMP

-f 1: two lines, TCP flags, hex ports

```
# flow-cat ft-v05.2008-12-01.171500-0500 | flow-print -n -f 1 | less
Sif  SrcIPaddress     DIf  DstIPaddress      Pr SrcP DstP  Pkts  Octets
 StartTime          EndTime             Active   B/Pk Ts Fl

0000 36.85.32.9       0000 158.43.192.1      11 915  35    1          59
 1201.17:09:46.750  1201.17:09:46.869      0.119 59  00 00

0000 158.43.192.1     0000 36.85.32.9        11 35   915   1          134
 1201.17:09:46.750  1201.17:09:46.869      0.119 134 00 00

0000 36.85.32.37      0000 83.243.35.204     06 19   1013  14         1035
 1201.17:09:46.912  1201.17:09:51.890     4.978  73  00 1b

0000 83.243.35.204    0000 36.85.32.37       06 1013 19    12         1527
 1201.17:09:46.912  1201.17:09:51.890     4.978  127 00 1b
...
```

-f 4: BGP

```
# flow-cat ft-v05.2008-12-01.171500-0500 | flow-print -n -f 4 | less
srcIP              dstIP              prot  srcAS  dstAS  octets      packets
36.85.32.9/0       158.43.192.1/0     udp   0      701    59          1
158.43.192.1/0     36.85.32.9/0       udp   701    0      134         1
36.85.32.37/0      83.243.35.204/0    tcp   0      4713   1035        14
83.243.35.204/0    36.85.32.37/0      tcp   4713   0      1527        12
…
```

- tcp/udp ports are missing
- locally-sourced traffic is from AS 0

-f 5: widescreen

- my favorite format, 132 characters
- Includes:
  - start & end time
  - source & dest interface
  - source & dest IP
  - sorce & dest port
  - protocol
  - flags
  - packets
  - octets

-f 6: IP accounting

- Cisco veterans have IP accounting format burned into their brain

```
# flow-cat ft-v05.2008-12-01.171500-0500  | flow-print -f 6 | sort -rnk 4 | less
 36.85.32.36      64.18.6.14                 12820           19216320
 36.85.32.36      64.18.6.13                 12820           19216320
 207.46.209.247   36.85.32.4                 10977           16458558
 84.96.92.121     36.85.32.37                 6904            9671951
...
```

flow-nfilter

- “What is the most heavily used port on this server?”
- “Who is connecting from this foreign network?”
- “What sort of traffic are we seeing from this application?”

Netflow can answer all of these, once you filter your data!

Filtering Components

- Filter primitives: tiny individual identifiers for a single component of traffic: TCP/IP port, host, protocol, etc
- Assemble primitives into a report definition.

For example: write primitives for a server's IP address, the TCP protocol, and port 80, and combine them into the “web server traffic” report. This is a different report than “all TCP traffic to this server.”

filter primitives

Every primitive has this rough format:

```
filter-primitive label
   type primitive-type
   permit whatever
   default deny|accept
```

Selected Primitives

- as (autonomous system, for BGP users)
- ip-protocol (1=ICMP, 6=TCP, 17=UDP)
- ip-tcp-flags (OR of all flags set in flow)
- ip-port (TCP or UDP port)
- ip-address
- ip-address-prefix (subnet)

See flow-nfilter(1) for the full list.

Complete Primitives

```
filter-primitive smtpports
   type ip-port
   permit 25
   default deny

filter-primitive washdc-addresses
   type ip-address-prefix
   permit 172.16.3.0/24
   default deny
```

More Primitives

```
filter-primitive mailservers
   type ip-address
   permit 172.16.1.5
   default deny

filter-primitive not-mailservers
   type ip-address
   deny 172.16.1.5
   permit 172.16.0.0/12
   default deny
```

Still More Primitives

You can combine several like entries, i.e.:

```
filter-primitive redflag-ports
   type ip-port
   permit 23
   permit 110
   permit 138
   permit 139
   permit 1433
   default deny
```

Illegal Primitives

You cannot combine different types of match in a single primitive. The following is wrong.

```
filter-primitive email
   type ip-port
   allow 25
   type ip-protocol
   allow 6
   default deny
```

combining primitives into filters

All filters have this format:

```
filter-definition filter-name
   match condition primitive1
   match condition primitive2
   match condition primitive3
   ...
```

Filter Conditionals

- ip-source-address
- ip-destination-address
- ip-protocol
- ip-tcp-flags
- ip-source-port
- ip-destination-port

...and more, see flow-nfilter(1) for details

Writing a Filter

```
filter-definition inboundmail
  match dst-ip-addr mailservers
  match dst-ip-port smtpports
  match ip-protocol tcp
```

This filter only captures the inbound side of the TCP conversations -- incoming email.

Writing a Broader Filter

```
filter-definition allmail
   match ip-source-address mailservers
   match ip-destination-port smtpports
   or
   match ip-destination-address mailservers
   match ip-destination-port smtpports
```

This filter catches all email, inbound and outbound – but not the server's SMTP replies.

This quickly gets ugly. Don't do it.

Be certain of your question before asking it!

Smaller, Simple Filters

```
filter-definition smtp
   match dst-ip-port smtp
   or
   match src-ip-port smtp

filter-definition mailservers
   match ip-source-address mailservers
   or
   match ip-destination-address mailservers
```

using filters

Specify the filter definition with *-F filtername*

```
flow-cat ft-* | flow-nfilter -F allmail |
  flow-print
```

Concatenate filters to make more complex queries on the fly

```
flow-cat ft-* | flow-nfilter -F smtp | flow-
  nfilter -F mailservers | flow-print
```

filtered results

```
# flow-cat * | flow-nfilter -F mailservers | flow-nfilter -F smtp | flow-print

srcIP            dstIP            prot  srcPort  dstPort  octets      packets
66.196.82.7      63.85.32.36      6     25       15650    2466        53
92.107.53.246    63.85.32.37      6     49609    25       2357        12
63.85.32.37      92.107.53.246    6     25       49609    1128        12
92.107.53.246    63.85.32.37      6     49611    25       2359        12
63.85.32.37      92.107.53.246    6     25       49611    1168        13
63.85.32.36      206.190.53.191   6     15658    25       118715      88
206.190.53.191   63.85.32.36      6     25       15658    2288        50
...
```

Other Queries

Traffic that should never happen:

```
filter-definition badtraffic
   match dst-ip-addr ourIPrange
   match dst-ip-port redflags
```

Email that isn't email:

```
filter-definition bogus_mail

   match dst-ip-addr not_mailservers
   match dst-ip-port smtpports
```

Exclusions

- We want to know everything except a certain type of traffic. "What else is our DNS server doing?"

```
filter-definition notDNS
   invert
   match dst-ip-port dnsports
   or
   match src-ip-port dnsports
```

More Exclusions

- Who is sending mail without using our mailserver?

```
filter-definition not-mailservers
   invert
   match ip-source-address mailservers
   or
   match ip-destination-address mailservers

# flow-cat ft-* | flow-nfilter -F smtp | flow-
  nfilter -F not-mailservers | flow-print
```

Defining Filters on the Command Line

- Three variables: ADDR (address), PORT, and PROT (protocol)
- defined by primitives VAR_ADDR, VAR_PORT, and VAR_PROT
- built-in reports:
  - ip-src-addr
  - ip-dst-addr
  - ip-prot
  - ip-src-port
  - ip-dst-port

using command-line filters

- define variable with -v

```
# flow-cat ft-* | flow-nfilter -F ip-src-addr -v ADDR=198.22.63.8 | flow-print
```

Your Own Command-Line Filters

- show bidirectional traffic to an address

```
filter-definition ip-addr
  match ip-source-address VAR_ADDR
  or
  match ip-destination-address VAR_ADDR
```

- show traffic to or from a port

```
filter-definition ip-port
  match ip-source-port VAR_PORT
  or
  match ip-destination-port VAR_PORT
```

Using custom command-line filters

- Can replace many filters defined in filter.cfg.

```
# flow-cat ft-* | flow-nfilter -F ip-addr -v
ADDR=198.22.63.8 | flow-nfilter -F ip-port
-v PORT=25 | flow-print -f 5
```

When to use Which Filters

- use command-line filters for individual addresses, ports, or protocols
- Use filter.cfg for multiples or ranges

flow-report

Creates statistics about flows: most popular IPs and ports, broad generalizations about traffic, etc. Successor to flow-stat.

Configured in stat.cfg

```
# flow-cat ft-* | flow-report
```

default report part 1

```
#  --- ---- ---- Report Information --- --- ---
# build-version:          flow-tools 0.68.4
# name:                   default
# type:                   summary-detail
# options:                +header,+xheader,+totals
# fields:                 +other
# records:                0
# first-flow:             1230785402 Wed Dec 31 23:50:02 2008
# last-flow:              1230785698 Wed Dec 31 23:54:58 2008
# now:                    1241395118 Sun May  3 19:58:38 2009
#
# mode:                   streaming
# compress:               off
# byte order:             little
# stream version:         3
# export version:         5
#
```

Default report part 2

```
#  ['/usr/local/bin/flow-rptfmt', '-f', 'ascii']
Ignores:                    0
Total Flows:                3539
Total Octets:               10613168
Total Packets:              20687
Total Duration (ms):        7550928
Real Time:                  1230785698
Average Flow Time:          2133.000000
Average Packets/Second:     513.000000
Average Flows/Second:       2998.000000
Average Packets/Flow:       5.000000
Flows/Second:               0.001364
Flows/Second (real):        0.000003
```

Default Report part 3

```
Average IP packet size distribution:

   1-32   64   96  128  160  192  224  256  288  320  352  384  416  448  480
   .000 .283 .413 .084 .129 .027 .021 .009 .006 .005 .002 .002 .004 .001 .003

    512  544  576 1024 1536 2048 2560 3072 3584 4096 4608
   .000 .002 .000 .003 .007 .000 .000 .000 .000 .000 .000

Packets per flow distribution:

      1    2    4    8   12   16   20   24   28   32   36   40   44   48   52
   .814 .021 .028 .041 .051 .015 .007 .003 .001 .003 .001 .001 .001 .001 .000

     60  100  200  300  400  500  600  700  800  900 >900
    .002 .001 .002 .002 .002 .001 .001 .001 .001 .000 .000
```

Default Report part 4

```
Octets per flow distribution:

     32   64  128  256  512 1280 2048 2816 3584 4352 5120 5888 6656 7424 8192
   .000 .247 .440 .149 .029 .066 .013 .022 .006 .001 .003 .002 .001 .001 .001

   8960 9728 10496 11264 12032 12800 13568 14336 15104 15872 >15872
   .001 .001  .000  .002  .001  .000  .001  .000  .001  .001  .010

Flow Time Distribution (ms):

    10    50  100  200  500 1000 2000 3000 4000 5000 6000 7000 8000 9000
10000
   .815 .001 .006 .016 .021 .004 .013 .023 .017 .012 .010 .008 .003 .006 .006

  12000 14000 16000 18000 20000 22000 24000 26000 28000 30000 >30000
   .004  .001  .004  .001  .002  .002  .002  .003  .002  .002  .015
```

Report Options

- flow-report supports 5 different options.
  - TYPE: which report to run
  - SORT: data sorting order
  - FIELDS: columns to display
  - OPTIONS: enables/disables report-wide features
  - RPTOPT: formatting program features
- enable or disable options with -v

Running Reports

```
# flow-cat ft-v05.2008-12-01.12* | flow-report -v
TYPE=ip-source-address | less
...
#  ['/usr/local/bin/flow-rptfmt', '-f', 'ascii']
ip-source-address flows octets    packets duration
85.63.32.37       12163 107898108 204514  159749392
158.43.128.72     16389 1962766   16711   49357139
85.63.32.4        54280 127877204 785592  831419980
198.6.1.1         7627  970898    7992    26371278
…
```

sorting reports by flows

- set SORT with column name to sort
- + for descending order, - for ascending

```
# flow-cat ft-v05.2008-12-01.12* | flow-report -v
TYPE=ip-source-address -v SORT=+flows | less
...
#  ['/usr/local/bin/flow-rptfmt', '-f', 'ascii']
ip-source-address flows octets    packets duration
85.63.32.4        54280 127877204 785592  831419980
158.43.128.72     16389 1962766   16711   49357139
85.63.32.37       12163 107898108 204514  159749392
198.6.1.5         8826  1425518   11339   24124445
85.63.32.36       8786  21773315  38616   44443605
198.6.1.1         7627  970898    7992    26371278
...
```

sorting reports by octets

- same flow file, sorted by octets

```
# flow-cat ft-v05.2008-12-01.12* | flow-report -v
TYPE=ip-source-address -v SORT=+octets | less
...
#  ['/usr/local/bin/flow-rptfmt', '-f', 'ascii']
ip-source-address flows octets    packets duration
207.46.209.247    25    131391013 90275   2967322
85.63.32.4        54280 127877204 785592  831419980
85.63.32.37       12163 107898108 204514  159749392
85.63.32.7        116   72083511  55415   15057488
85.63.32.130      145   49604492  74852   36232749
85.63.32.8        88    48766466  36166   7181558
...
```

- different top hosts – connections are not bandwidth!

valid SORT values

Check the comments at report top

```
# fields:  +key,+flows,+octets,+packets,+duration,+other


ip-source-address flows octets    packets duration
85.63.32.4        54280 127877204 785592  831419980
158.43.128.72     16389 1962766   16711   49357139
85.63.32.37       12163 107898108 204514  159749392
…
```

- To sort by IP, use +key or -key

Removing Columns

- Not interested in certain data? Remove it with -fieldname
-

```
# flow-cat ft-v05.2008-12-01.12* | flow-report -v
TYPE=ip-source-address -v SORT=+octets -v FIELDS=-
duration | less
...
ip-source-address flows octets    packets
207.46.209.247    25    131391013 90275
85.63.32.4        54280 127877204 785592
85.63.32.37       12163 107898108 204514
...
```

IP Address Reports

- Generic IP address reports
  - ip-address
  - ip-destination-address
  - ip-source-address
- Most connected hosts
  - ip-source-address-destination-count
  - ip-destination-address-source-count
  - These reports don't offer to sort by number of connected hosts! Use `| sort -rnk 2 |less` to sort by number of connected hosts

port and protocol reports

- Generic port reports
  - ip-port
  - ip-source-port
  - ip-destination-port
  - ip-source/destination-port
- report on protocols with
  - ip-protocol

Traffic Size & Speed

- size
  - packet-size
  - octets (aka bytes per flow)
  - packets (aka packets per flow)
- speed
  - bps
  - pps
  - linear-interpolated-flows-octets-packets
    - my favorite, epochal seconds and flow data

Custom Reports

- not so much custom as customized
- goal: to have an easily run standard report, such as:

  ```
  # flow-cat * | flow-report -S resets
  ```

- made of stat-report and stat-definitions

```
stat-report packets
  type linear-interpolated-flows-octets-packets
  output
    fields -octets,-flows
    path |/usr/local/bin/flow-rptfmt
```

using stat-report in definitions

```
stat-definition resets
  filter rst-only
  report packets

stat-definition syn-only
  filter syn-only
  report packets
```

Running Custom Reports

- Use -S flag

```
# flow-cat * | flow-report -S resets | less
```

- Output looks like:

```
unix-secs  packets
1228150595 0.068702
1228150596 0.068702
1228150597 0.068702
…
```

- Feed this to gnuplot, generate ad-hoc graphs

Netflow and Real-World Problems

"You've blamed the network for years, now it's my turn!"

- Reactions to Evidence
- Handling Problems
- Turning Opponents into Partners

“Your site is down!”

Write a specific Netflow report that captures traffic between the client network and your network.

“We're pushing 5Mb/s, but we're not seeing anything from your network. Our traceroutes to you die in MCI's Willow Springs facility.”

“Your network is blocking our single sign-on system!”

```
filter-primitive client-sso-server
    type ip-address
    permit 172.16.101.99
    default deny
filter-primitive our-app-server
    type ip-address
    permit 192.168.1.2
    default deny

filter-definition client-sso-traffic
    match src-ip-addr client-sso-server
    match dst-ip-addr our-app-server
    or
    match dst-ip-addr client-sso-server
    match src-ip-addr our-app-server
```

Possible Result #1

```
srcIP           dstIP          prot srcPort dstPort octets  packets
192.168.1.2     172.16.101.99  6    1871    5060    586           6
172.16.101.99   192.168.1.2    6    5060    1871    711           7
192.168.1.2     172.16.101.99  6    1873    5060    589           6
172.16.101.99   192.168.1.2    6    5060    1873    699           7
```

"Our app server is calling your SSO server on port 5060, and we're seeing traffic return. Each session contains 500-700 bytes of data in several packets each way. Can you do me a favor and confirm that your SSO system should be communicating on these ports?"

Possible Result #2

```
srcIP         dstIP         prot srcPort dstPort octets  packets
192.168.1.2   172.16.101.99 6    1871    5060     16          1
192.168.1.2   172.16.101.99 6    1873    5060     19          1
```

"We're transmitting data to your SSO servers, but nothing seems to be coming back. Can you check your firewall logs to see what you're seeing on your side?"

“How Much Bandwidth?”

- Fired up RTG (http://rtg.sourceforge.net), which makes SNMP queries every 20 seconds across the WAN. How much bandwidth does monitoring one spoke site take?

```
# flow-cat ft-v05.2007-09-26.121* | flow-nfilter -F loghost |
    flow-nfilter -F site1-router | flow-nfilter -F snmp-traffic
    | flow-report
```

This much bandwidth...

```
Ignores:                    0
Total Flows:                3539
Total Octets:               10613168
Total Packets:              20687
Total Duration (ms):        7550928
Real Time:                  1230785698
Average Flow Time:          2133.000000
Average Packets/Second:     513.000000
Average Flows/Second:       2998.000000
Average Packets/Flow:       5.000000
Flows/Second:               0.001364
Flows/Second (real):        0.000003
```

Flow-report tells us everything we need

Conficker!

- spreads over CIFS (TCP/445)
- filter on that, feed into connectedness report

```
# flow-cat ft-v05.2009-03-29.* | flow-nfilter -F ip-port
-v PORT=445 | flow-report -v TYPE=ip-source-address-
destination-count -v OPTIONS=-header,+names | sort -rnk
2 | less
```

- most connected hosts should be SMB file servers, is probably your conficker “Patient Zero”

Bonus Track: FlowViewer

- Suite of tools for analysis of flow-tools data
- Public domain, from NASA
- Includes: FlowViewer, FlowTracker, FlowGrapher
  - FlowViewer: front end to flow-print/flow-stat
  - FlowTracker: arbitrary RRD generator
  - FlowGrapher: arbitrary flow graphs

FlowViewer Installation

- Download from http://ensight.eos.nasa.gov/FlowViewer/
- Untar in Web server directory
- Most files owned by root
- Directories *tracker*, *tmp*, *reports*, and *graphs* owned by unprivileged Web user

- Other requirements: Perl, Perl GD, rrdtool, flow-tools

FlowViewer Web server

- Apache httpd.conf config:

```
Alias /FlowViewer/ /var/www/flowviewer/
<Directory "/var/www/flowviewer/">
        Options ExecCGI Indexes FollowSymLinks
#       AddHandler cgi-script .cgi
        Order allow,deny
        Allow from all
</Directory>
```

Configuring FlowViewer

- All FlowViewer tools share a comon configuration file, FlowViewer_Configuration.pm
- Default is ...pm.dist file, copy to .pm

- Petty annoyance: no index.html

FlowViewer_Configuration.pm

```
# Path variable
$ENV{PATH}.= ':/usr/local/bin:/usr/sbin';

# Server
$FlowViewer_server  = "loghost.mycompany.com";
# (IP address or hostname)

# Service
$FlowViewer_service       = "https";
  # (http, or https)
```

FlowViewer Config 2

Where FlowViewer saves reports

```
$reports_directory      = "/var/www/loghost/flowviewer/reports";
```

Where FlowGrapher saves graphs

```
$graphs_directory       = "/var/www/loghost/flowviewer/graphs";
```

Where FlowTracker saves trackers

```
$tracker_directory      = "/var/www/loghost/flowviewer/tracker";
```

Where are the FlowViewer Scripts?

```
$cgi_bin_directory      = "/var/www/loghost/flowviewer";
$work_directory         = "/tmp";
$names_directory        = "/tmp";
```

Where FlowViewer stores filters

```
$filter_directory       = "/var/www/loghost/flowviewer/tmp/filters";
```

RRD log space

```
$rrdtool_directory      = "/var/www/loghost/flowviewer/tmp/rrdtools";

$flow_bin_directory     = "/usr/local/bin";
$rrdtool_bin_directory  = "/usr/local/bin";

$trackings_title        = "Minions of Lucas";
```

FlowViewer Config 3

how many seconds long are your flows?

```
$flow_capture_interval    = (5 * 60);
```

how long are your flow files?

```
$flow_file_length         = (5 * 60);
```

how far back should graphs default to starting?

```
$start_offset             = (90 * 60);  # e.g., 90
  minutes ago
```

how far back should graphs default to ending?

```
$end_offset               = (30 * 60);  # e.g., 30
  minutes ago
```

should report times default to even hours

```
$use_even_hours           = "Y";
```

flow-capture's –N argument

```
$N                        = 0;
```

how far back does this go?

```
$maximum_days             = "91";
```

FlowViewer Logs & Devices

where are your flows?

```
$flow_data_directory    = "/var/log/flowviewer/";
```

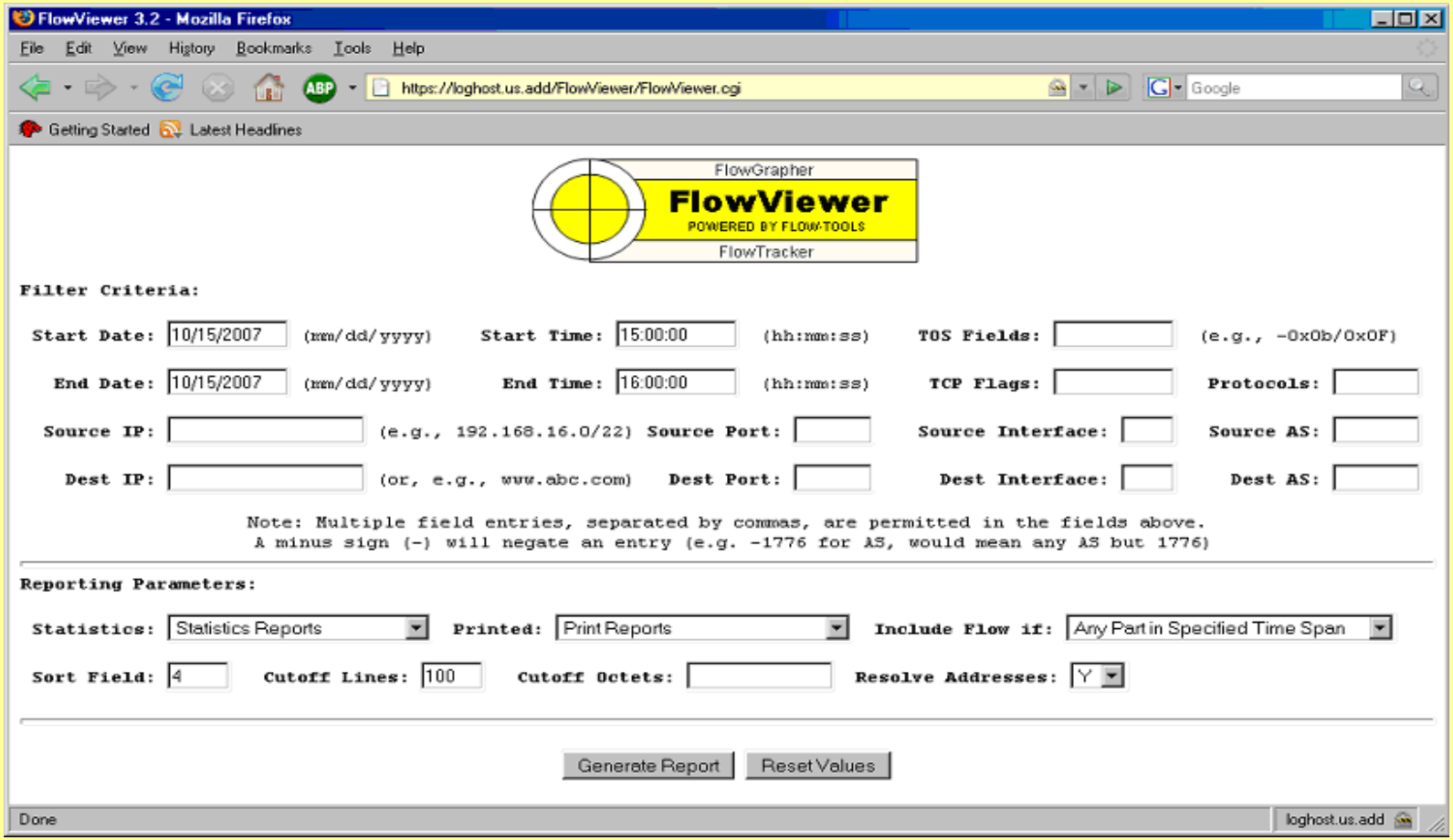
you can list multiple collectors if your log directory structure is set up for it

```
@devices                        = (“internet“,”mpls”,”core”);
```

- /var/log/flowviewer contains directories named “internet”, “mpls”, and “core”. These are symlinks to actual directories.
- FlowViewer flow-cat’s the contents of these directories!

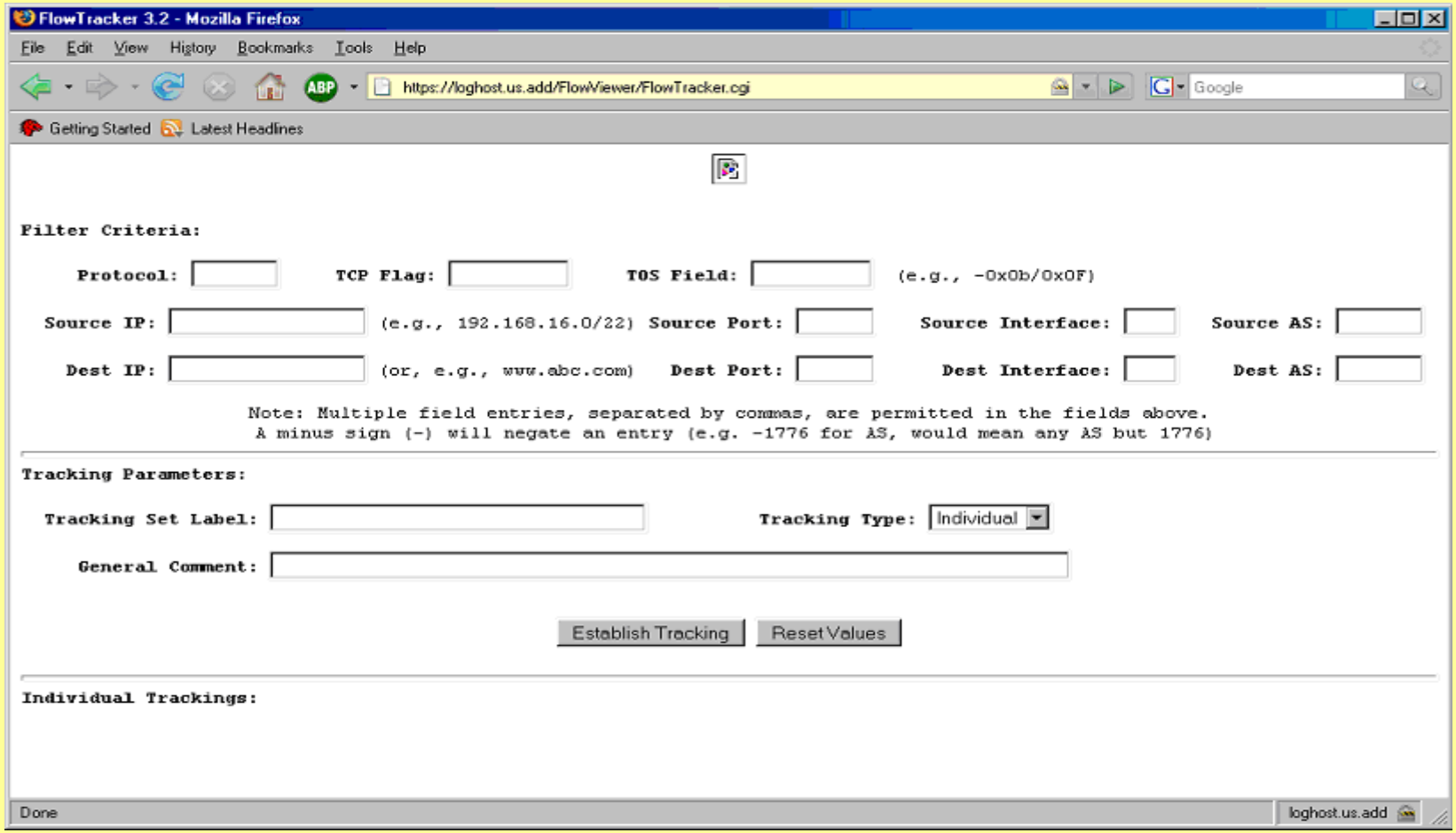
FlowViewer

- Front-end to flow-print and flow-stat



FlowTracker

- Define flowscan-like graphs

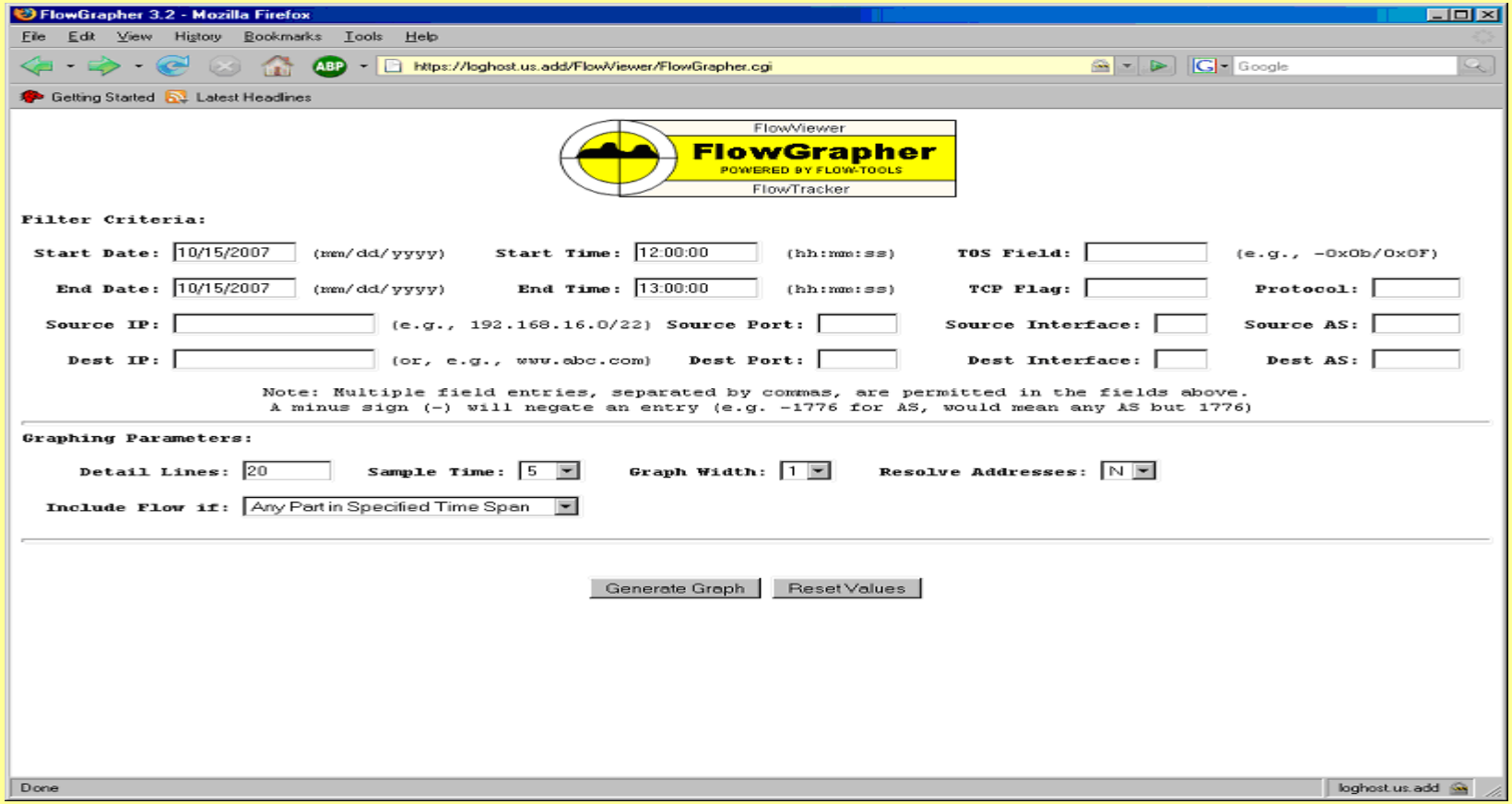


FlowTracker setup

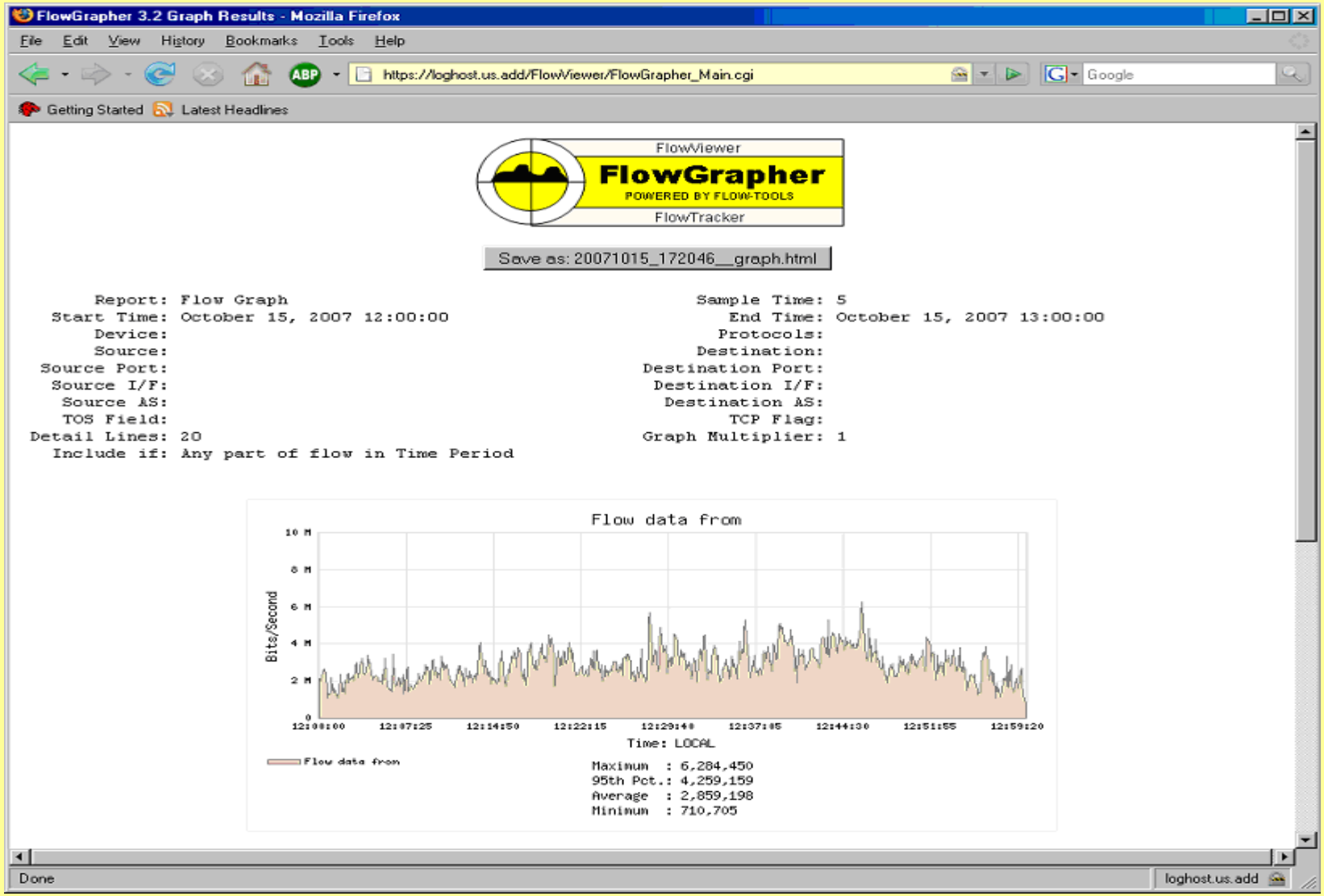
- creates RRD graphs, so you need processes to read and process flow files as they arrive.
- start FlowTracker_Collector and FlowTracker_Grapher in the background to process flow files and update RRDs.
- Use CGI script to dictate what you want to track. You'll get a confirmation screen.
- Trackers visible in http://whatever/FlowViewer/tracker/ directory, use Apache or .htaccess to restrict access.

FlowGrapher

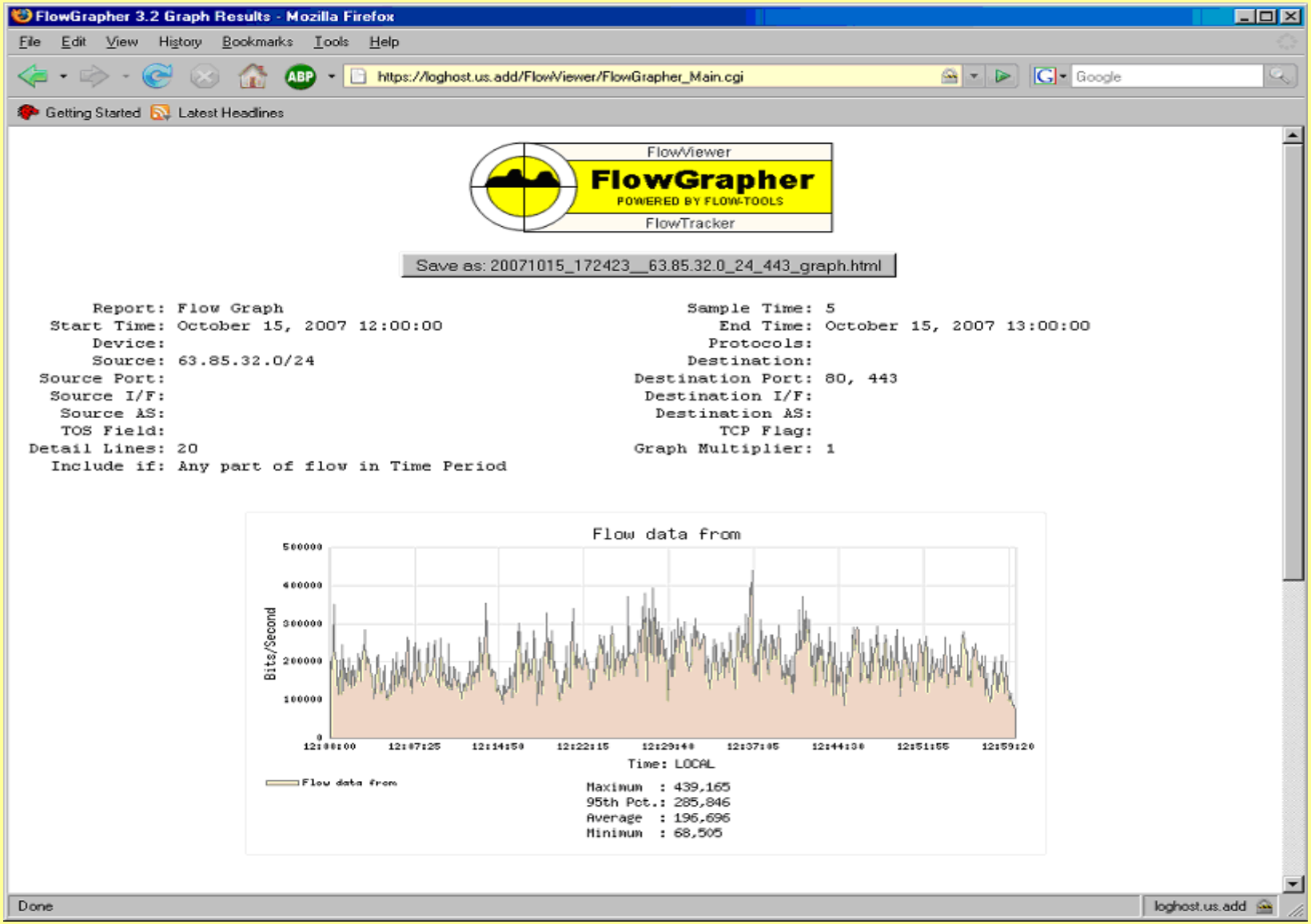
- Arbitrary time-based reporting



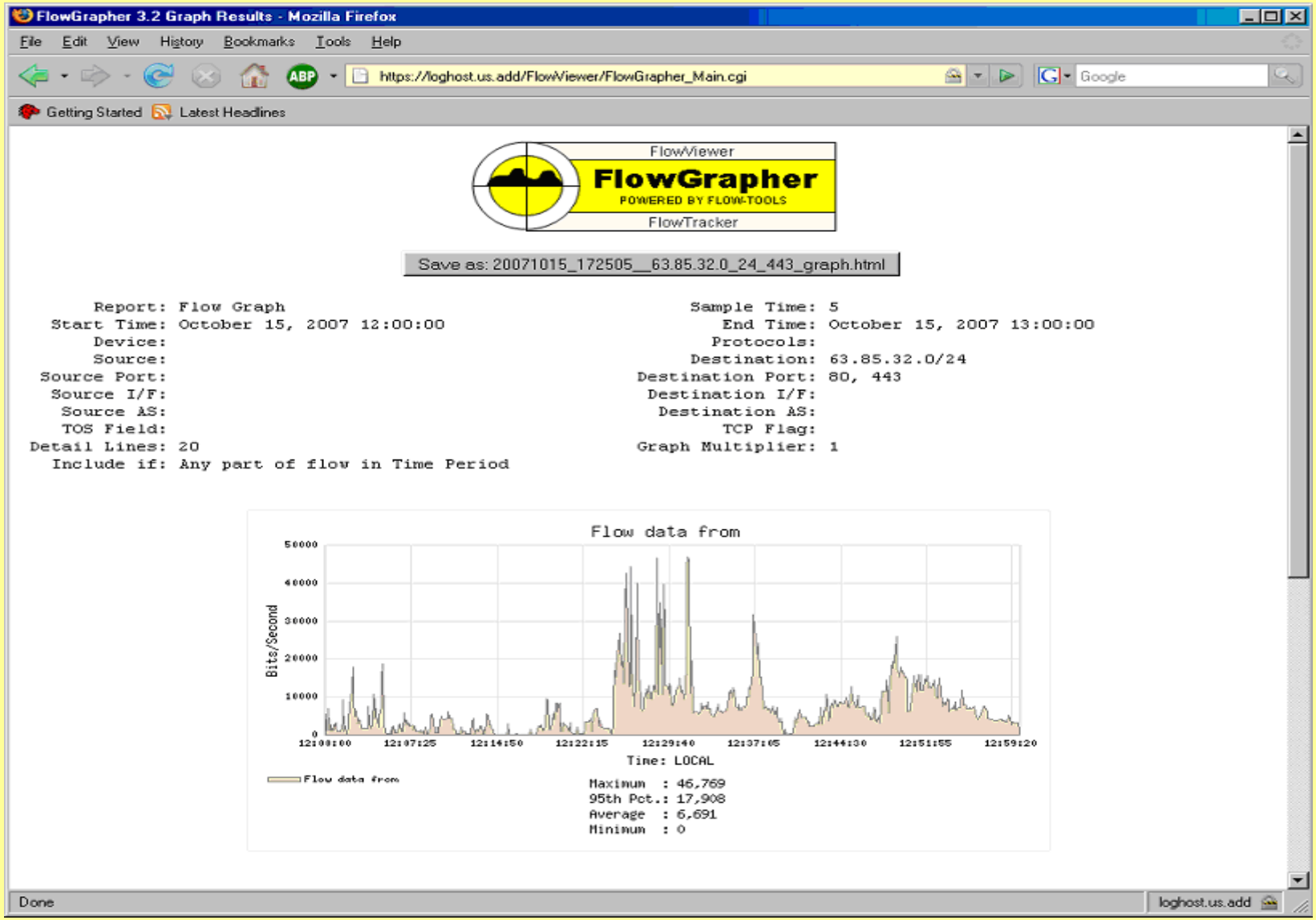
FlowGrapher Results



FlowGrapher results #2



Flowgrapher Results #3



How Not To Go BOFH

- You have evidence, you can afford to be generous
- “Here is the evidence that shows that the network is working. I am, however, happy to help *you* diagnose and troubleshoot the problem *you* are having with *your* application.”

Red Flags: RST-only and SYN-only flows

```
filter-primitive syn-rst
        type ip-tcp-flags
        permit 0x6
        default deny

filter-primitive rst-only
        type ip-tcp-flags
        permit 0x2
        default deny

filter-primitive syn-only
        type ip-tcp-flags
        permit 0x4
        default deny
```

the generic “weird crap” filter

```
filter-definition badtcp
  match ip-tcp-flags syn-rst
  or
  match ip-tcp-flags syn-only
  or
  match ip-tcp-flags rst-only
```

```
 # flow-cat * | flow-nfilter -F badtcp | flow-print -f 5 | less
Start             SrcIPaddress    SrcP  DstIPaddress DstP Fl Pkts Octet

1130.23:50:45.495 119.234.0.29    4369  63.85.32.37    443 4  10    460
1130.23:50:31.412 125.65.165.139 12200 63.85.32.188   443 2  1     46
1130.23:50:31.413 125.65.165.139 12200 63.85.32.230   443 6  1     46
...
```

Finding Port Scanners

find_scanners in flow-tools/contrib

Does not include any sort of limit; will print out the first X port-scanning candidates

I suggest a wrapper around this for identifying real scanner candidates, i.e.:

```
http://www.blackhelicopters.org/~mwlucas/scannercatch.pl
```

Netflow and choosing an ISP

ParseBGPDump.pl

takes a list of autonomous system numbers and a BGP-tagged netflow dump and determines which ISPs originate most of your traffic

Your Problems?

Your networks are more complicated and diverse than any I can have up here.

What are *your* problems?