Introduction to NETGRAPH on FreeBSD Systems

Dr. Adrian Steinmann <ast@marabu.ch>
AsiaBSDCon 2012
Tokyo University of Science, Tokyo, Japan
22 – 25 March, 2012
Tutorial Materials

Extended Abstract
Tutorial Slides
‘All About Netgraph’
man 8 ngctl
man 8 nghook
man 4 netgraph
About Myself

Ph.D. in Mathematical Physics (long time ago)

Webgroup Consulting AG (now)

IT Consulting: Open Source, Security, Perl

FreeBSD since version 1.0 (1993)

NetBSD since version 3.0 (2005)

Traveling, Sculpting, Go
Tutorial Outline

(1) Netgraph in a historical context
(2) Getting to know netgraph
(3) Working with netgraph
(4) Details of frequently used netgraph node types
(5) Examples using netgraph nodes as building blocks
(6) Investigate some more sophisticated examples
(7) Guidelines for implementing custom node types
What is Netgraph

Netgraph is in-kernel ‘network plumbing’

Drawn as a graph, a communication protocol can be seen as data packets flowing (bidirectionally) along the edges (lines) between nodes where the packets are processed.

In FreeBSD since version 3.0 (2000)

Created on FreeBSD 2.2 (1996) by Archie Cobbs <archie@freebsd.org> and Julian Elischer <julian@freebsd.org> for the Whistle InterJet at Whistle Communications, Inc.
Ancient History

Dennis Ritchie (Eighth Edition Research Unix, Bell Labs)
October 1984: ‘A Stream Input-Output System’

After open() of a plain device
Ancient History

Dennis Ritchie (Eighth Edition Research Unix, Bell Labs)
October 1984: ‘A Stream Input-Output System’

After open() of TTY device
Ancient History

Dennis Ritchie (Eighth Edition Research Unix, Bell Labs)
October 1984: ‘A Stream Input-Output System’

Networked TTY device
System V STREAMS

SVR3, SVR4, X/Open

Push modules onto a stack pointed to by the handle returned by the open() system call to the underlying device driver.
System V STREAMS

FreeBSD supports basic STREAMS system calls – see `streams(4)`

Linux STREAMS (LiS)
STREAMS Multiplexors

A transport protocol (TP) supporting multiple simultaneous STREAMS multiplexed over IP.

The TP routes data from the single lower STREAM to the appropriate upper STREAM.
Best intro to NETGRAPH:
‘All About Netgraph’
by Archie Cobbs
Daemon News, March 2000

http://people.freebsd.org/~julian/netgraph.html

(last retrieved February 2012)
Netgraph Platforms

• FreeBSD 2.2; mainline as of FreeBSD 3.0
• Port to NetBSD 1.5 (from FreeBSD 4.3) but not in mainline NetBSD
• DragonFly BSD “Netgraph7” (upgrade from netgraph from FreeBSD 4.x to FreeBSD 7.x)
• In one commercial Linux 6WindGate’s VNB (Virtual Networking Blocks) are derived from netgraph
New Approaches Today
Streamline (on Linux)

Herbert Bos
www.cs.vu.nl/~herbertb
VU University, Amsterdam

Scalable I/O Architecture (ACM TOCS 2011)

Address network latency problem on OS design level
Beltway Buffers, Ring Buffers: minimize copying in IPC
PipeFS for visualizing/manipulating nodes/graph
New Approaches Today
netmap (on FreeBSD)

Luigi Rizzo, Università di Pisa, Italy
http://info.iot.unipi.it/~luigi/netmap/

Multicore CPU systems hit buffer contention at high speeds
New buffer sharing needed when cores and memory are cheap
New Approaches Today

netmap (on FreeBSD)

Luigi Rizzo, Università di Pisa, Italy
http://info.iit.unipi.it/~luigi/netmap/

Multicore CPU systems hit buffer contention at high speeds
New buffer sharing needed when cores and memory are cheap
New Approaches Today
netmap (on FreeBSD)

Memory mapped access to network devices
Fast and safe network access for user space applications
Evolution on FreeBSD

• Netgraph continues to be in mainline FreeBSD tree since 3.0 (started with 10 netgraph modules)

• New networking abstractions appearing in FreeBSD remain netgraph-aware via additional netgraph modules
  (4.4: ~20, 4.9: ~25, 5.4: ~30, 6.3: ~45, 7.2: ~50, 8.x: ~60)

• Netgraph in 5.x: added SMP (that is: locking, queuing, timers, bidirectional hooks)

• Netgraph in 8.x: support for VIMAGE
  most recent addition: ng_patch(4)
Timeline on FreeBSD

Initial “NETGRAPH 2”

FreeBSD 2.2 customized for Whistle InterJet

10 modules
Timeline on FreeBSD

First Public Release

FreeBSD 3.3

16 modules

async cisco echo frame_relay hole iface ksocket lmi ppp pppoe rfc1490 socket tee tty UI vjc
Timeline on FreeBSD

NETGRAPH

FreeBSD 4.11

16 + 2 modules

async cisco echo frame_relay hole iface ksocket lmi ppp pppoe rfc1490 socket tee tty UI vjc

  bpf ether
Timeline on FreeBSD

NETGRAPH

FreeBSD 5.5 / FreeBSD 6.x

18 + ~30 modules

async cisco echo frame_relay hole iface ksocket lmi ppp pppoe rfc1490 socket tee tty UI vjc

bpf ether

atm atmllc atmpif **bluetooth bridge** bt3c btsocket device eiface etf fec gif gif_demux h4 hci hub ip_input 12cap **l2tp** mppc netflow one2many pptpgre split sppp ssclf sscop ubt uni **vlan**
Timeline on FreeBSD

“NETGRAPH 7”

FreeBSD 7.4 / 8.x

~48 + ~12 modules

async cisco echo frame_relay hole iface ksocket lmi ppp pppoe rfc1490 socket tee tty UI vjc

bpf ether

atm atmllc atmpif bluetooth bridge bt3c btsocket device eiface etf fec gif gif_demux h4 hci hub ip_input 12cap 12tp mppc netflow one2many pptpgre split sppp ssccfu sscop ubt uni vlan

car ccatm deflate ipfw nat patch pred1 source sync_ar sync_sr tag tcpmms
Software Licensing

The netgraph framework is in FreeBSD kernel, hence it is under BSD license.

Netgraph nodes may have any license.

‘A Gentlemen's Agreement – Assessing The GNU General Public License and its Adaptation to Linux’ by Douglas A. Hass, Chicago-Kent Journal of Intellectual Property, 2007 mentioned netgraph as an example to be followed:

For example, FreeBSD uses a networking architecture called netgraph. Along with the rest of FreeBSD, this modular architecture accepts modules under virtually any license. Unlike Linux, Netgraph’s API integrates tightly with the FreeBSD kernel, using a well-documented set of standard function calls, data structures, and memory management schemes. Regardless of the underlying licensing structure, modules written for netgraph compliance must interact with netgraph’s structure in a predictable, predefined manner.
The aim of netgraph is to supplement rather than replace the existing kernel networking infrastructure:

- A flexible way of combining protocol and link level drivers
- A modular way to implement new protocols
- A common framework for kernel entities to inter-communicate
- A reasonably fast, kernel-based implementation
More

★ ‘All About Netgraph’
★ man -k netgraph

• netgraph(3) is the programmer’s API

• News archives:

Many (sometimes good) questions, less answers, some working examples, and very often “I did that once but don’t have the working example here right now” ... (for some counterexamples, see URLs at end of tutorial slides)
How to Build a Netgraph

(i) Create a node `node_A` (with unconnected hooks)

(ii) Create a peer node `node_B`, connecting one of the hooks of `node_A` (`hook_A`) to one of the hooks of `node_B` (`hook_B`) – this creates an edge from `node_A` to `node_B` along `hook_A` to `hook_B`

(iii) Repeat step (i) or step (ii), or:

    Connect an unconnected hook to another one, creating a new edge between existing nodes
Control Messages

- Nodes can receive control messages, they reply to them by setting a reply flag
- Control messages are C structures with a (generic) netgraph type cookie and variable payload
- A node can also define its own message types by taking a unique netgraph type cookie
Addressing Nodes

• Every node has an address (also called path) and an internal ID

• A named node can be addressed absolutely

    nodename: (for example, em0 : )

• A nameless node can be addressed absolutely via its internal ID

    If node has internal ID 0000007e, it can be address as [7e] :
Where NETGRAPH hooks live in FreeBSD 4.x kernel

http://people.freebsd.org/~julian/layer2c.pdf

Protocol stacks (e.g. IP)

ether_demux

demux

ipfw hook

dummy tag?

ether_output

mux

ipfw hook

Netgraph

orphan

upper

lower

bpf

bpf_mtap hook

ether_input

bridge hook

netgraph hook

bpf_mtap hook

ether_output

frame

rx

vlan

tx

hw tags

to physical

from virtual

http://people.freebsd.org/~julian/layer2c.pdf
Where NETGRAPH hooks live in FreeBSD 6.x kernel
Netgraph User/Kernel Interface

BSD Sockets!

AF_NETGRAPH address family for ctrl, data protocols (netstat -f ... -p ...)

User space
ngctl
nghook

Kernel space
netgraph.ko
ng_socket.ko
ng_* .ko
# ngctl

Available commands:

- config: Get or set configuration of node at `<path>`
- connect: Connects hook `<peerhook>` of the node at `<relpath>` ...
- debug: Get/set debugging verbosity level
- dot: Produce a GraphViz (.dot) of the entire netgraph.
- help: Show command summary or get more help on a specific ..
- list: Show information about all nodes
- mkpeer: Create and connect a new node to the node at "path"
- msg: Send a netgraph control message to the node at "path"
- name: Assign name `<name>` to the node at `<path>`
- read: Read and execute commands from a file
- rmhook: Disconnect hook "hook" of the node at "path"
- show: Show information about the node at `<path>`
- shutdown: Shutdown the node at `<path>`
- status: Get human readable status information from the node ...
- types: Show information about all installed node types
- write: Send a data packet down the hook named by "hook".
- quit: Exit program

+ ^D
Starting Netgraph

Netgraph automatically loads necessary kernel modules

# kldstat
Id  Refs  Address   Size    Name
  1     8  0xc0400000  9fad10  kernel
  2     1  0xc0dfb000  6a45c   acpi.ko

# ngctl list
There are 1 total nodes:
  Name: ngctl39213  Type: socket  ID: 00000001  Num hooks: 0

# kldstat
Id  Refs  Address   Size    Name
  1     8  0xc0400000  9fad10  kernel
  2     1  0xc0dfb000  6a45c   acpi.ko
  6     1  0xc85ba000  4000    ng_socket.ko
  7     1  0xcc648000  b000    netgraph.ko
Querying Netgraph Status

```
# ngctl &
# netstat -f ng
Netgraph sockets
Type  Recv-Q Send-Q Node Address   #Hooks
ctrl  0      0 ngctl1314:        0
data  0      0 ngctl1314:        0

# ngctl list
There are 1 total nodes:
    Name: ngctl99971   Type: socket   ID: 00000008   Num hooks: 0
```

There used to be a bug in 7.x, 8.x introduced 2006 (when ng_socket.c became a loadable module) which caused netstat -f netgraph to fail (see kern/140446)
Creating Nodes

• ng_ether and ng_gif nodes are created automatically once the corresponding kernel module is loaded (and once loaded, they cannot be unloaded)

• ngctl mkpeer <node> <ngtype> <hook> <peerhook>
  node is usually [id]: or name:
    – the default is the current node
  ngctl mkpeer em3: tee lower right
Naming Nodes

• ngctl name <target> <name>
  where <target> is node:hook

  ngctl name em3:lower T

• Example: hook lower is one of the three hooks for the
  ng_ether (lower connects to raw ether device, upper
to the upper protocols, orphans is like lower, but only
  receives unrecognized packets – see man ng_ether)

  ngctl mkpeer T: MyType right2left MyHookR2L

  ngctl name T:right2left MyNode
**Connecting Nodes**

- Connecting nodes creates edges
- Hooks that are not connected do nothing, i.e. packets that would go there according to node type are dropped
- `ngctl connect` is used when both nodes already exist (as opposed to `mkpeer`, where a new node is created and connected in one step)

```
ngctl connect <node_a> <node_b> <hook_a> <hook_b>
ngctl connect MyNode: T: MyHookL2R left2right
ngctl connect T: em0: left upper
```
A first Net-graph

MyType=hole
ngctl mkpeer em0: tee lower right
ngctl name em0:lower T
ngctl mkpeer T: $MyType right2left MyHookR2L
ngctl name T:right2left MyNode
ngctl connect MyNode: T: MyHookL2R left2right
ngctl connect T: em0: left upper
Another Net-graph
And another Net-graph
Speaking with Nodes

• `ngctl msg <node> <command> [args ... ]`
  
  `ngctl msg T: getstats`

• What messages a particular node accepts (and what arguments it takes) depends on the node – read section 4 of the manual for `ng_xyz`

• Netgraph has ascii syntax for passing binary information to node – see, for example, `ng_ksocket(4)` or `ng_bpf(4)`
Removing an Edge

- `ngctl rmhook <hook>`
  `ngctl rmhook MyHookR2L`
- Specifying the node is optional:
  `ngctl rmhook <node> <hook>`
  `ngctl rmhook MyNode MyHookL2R`
Removing a Node

• Shutting down a node (all edges to hooks on this node are removed)

• `ngctl shutdown <node>`

  `ngctl shutdown T:`

These edges disappear:

  T:left – em0:lower
  T:right – em0:upper
  T:left2right – MyNode:MyHookL2R
  T:right2left – MyNode:MyHookL2R
Common Node Types

ng_ether(4)  fxp0:

• Hooks: upper, lower, orphans
• Process raw ethernet traffic to/from other nodes
• Attach to actual 802.11 hardware and are named automatically
• Messages
  getifname, getifindex, getenaddr, setenaddr,
  getpromisc, setpromisc, getautosrc, setautosrc,
  addmulti, delmulti, detach
Common Node Types

**ng_eiface(4) ngeth0:**

- **Hooks:** ether
- **Virtual ethernet interface providing ethernet framing**
- **Messages:**
  - set, getifname
More Nodes Types

ng_iface(4)  Virtual interface for protocol-specific frames
  • Hooks:      inet, inet6, ipx, atalk, ns, atm, natm
  • Messages:   getifname, point2point, broadcast,
                 getipaddr, getifindex

ng_tee(4)    Useful for tapping (for example, to snoop traffic)
  • Hooks:     left, right, left2right, right2left
  • Messages:  getstats, clrstats, getclrstats
Example: Snooping

kldload ngEther
ngctl mkpeer ${int}: tee lower left
ngctl name ${int}:lower T

ngctl connect ${int}: T: upper right
ngctl show T:
Name: T               Type: tee             ID: 0000003e   Num hooks: 2
Local hook Peer name Peer type Peer ID Peer hook
---------- --------- --------- ------- --------
right      ${int}    ether        00000019 upper
left       ${int}    ether        00000019 lower

nghook -an T: left2right
...
0020: d0 aa e4 d2 14 84 47 ae 24 fb 40 fd df 9b 80 10 ......G.$.@.....

ngctl msg T: getstats
Rec'd response "getstats" (1) from "[66]:":
Args:   { right={ inOctets=15332 inFrames=117 outOctets=13325 outFrames=126 }  
        left={ inOctets=13325 inFrames=126 outOctets=15332 outFrames=117   }  
        left2right={ outOctets=1027 outFrames=6  }  }

ngctl shut T:
ngctl shut ${int}:
Nodes of Special Interest

ng_socket(4)

- Enables the user-space manipulation (via a socket) of what is normally a kernel-space entity (the associated netgraph node)
- Hooks: arbitrary number with arbitrary names

ng_ksocket(4)

- Enables the kernel-space manipulation (via a netgraph node) of what is normally a user-space entity (the associated socket)
- Hooks: exactly one, name defines <family>/<type>/<proto>

These two examples show how to use ng_ksocket(4):
- /usr/share/examples/netgraph/ngctl
- /usr/share/examples/netgraph/udp.tunnel
Visualizing Netgraph

• Graph Visualization Software from AT&T and Bell Labs: http://www.graphviz.org/

• Port graphics/graphviz; various layouts:
  dot filter for hierarchical layouts of graphs
  neato filter for symmetric layouts of graphs
  twopi filter for radial layouts of graphs
  circo filter for circular layout of graphs
  fdp filter for symmetric layouts of graphs
Graphing Netgraph Graphs

```bash
# ngctl
+ name [8]: JustAPlainSocket
+ dot
graph netgraph {
    edge [ weight = 1.0 ];
    node [ shape = record, fontsize = 12 ] {
        "8" [ label = "\{JustAPlainSocket:|{socket|[8]}\}" ];
    };
    subgraph cluster_disconnected {
        bgcolor = pink;
        "8";
    };
};
^D
$ dot -Tpng ngctl.dot > ngctl.png
```

JustAPlainSocket:

<table>
<thead>
<tr>
<th>socket</th>
<th>[8]:</th>
</tr>
</thead>
</table>

User space
ngctl
nghook

Kernel space
netgraph.ko
g_socket.ko
ng_* .ko
Snoop Example Revisited

int=re0
ngctl mkpeer ${int}: tee lower left
ngctl name ${int}:lower T
ngctl connect ${int}: T: upper right
nghook -an T: left2right >/dev/null &
ngctl dot
ngctl shut T:
ngctl shut ${int}:

<unnamed>:
  socket [56]:
  hook

ngctl11690:
  socket [57]:

T:
  tee [53]:
  left2right
  right
  ether [19]:
  left

re0:
  ether [19]:
  upper
  lower

graph netgraph {
  edge [ weight = 1.0 ];

  node [ shape = record, fontsize = 12 ] {
    "5e" [ label = "\{<unnamed>:\{socket\[[5e]\]:}\}" ];
    "56" [ label = "\{<unnamed>:\{socket\[[56]\]:}\}" ];
    "19" [ label = "\{$(int)\{ether\[[19]\]:}\}" ];
    "5f" [ label = "\{ngctl11738:\{socket\[[5f]\]:}\}" ];
    "5b" [ label = "\{(T:\{tee\[[5b]\]:}\}" ];
  };

  subgraph cluster_disconnected {
    bgcolor = pink;
    "5f";
  }

  node [ shape = octagon, fontsize = 10 ] {
    "5e.hook" [ label = "hook" ];
  };

  edge [ weight = 2.0, style = bold ];
    "5e" -- "5e.hook";

  "5e.hook" -- "5b.left2right";

  node [ shape = octagon, fontsize = 10 ] {
    "19.upper" [ label = "upper" ];
    "19.lower" [ label = "lower" ];
  };

  edge [ weight = 2.0, style = bold ];
    "19" -- "19.upper";
    "19" -- "19.lower";

  node [ shape = octagon, fontsize = 10 ] {
    "5b.left2right" [ label = "left2right" ];
    "5b.right" [ label = "right" ];
    "5b.left" [ label = "left" ];
  };

  edge [ weight = 2.0, style = bold ];
    "5b" -- "5b.left2right";
    "5b" -- "5b.right";
    "5b" -- "5b.left";

  "5b.right" -- "19.upper";
  "5b.left" -- "19.lower";
}
# Bridge

**ng_bridge(4)**

**Hooks:**

- link0, ..., link31 (NG_BRIDGE_MAX_LINKS = 32)

**Messages:**

- setconfig, getconfig, reset, getstats, clrstats, getclrstats, gettable

This module does bridging on ethernet node types, each link carries raw ethernet frames so `ng_bridge(4)` can learn which MACs are on which link (and does loop detection). Typically, the link0 hook is connected to the first `ng_ether(4)` lower hook with `ngctl mkpeer`, and the subsequent ethernet interfaces' lower hooks are then connected with `ngctl connect`, making sure that the ethernet interfaces are in promiscuous mode and do not set the source IP:

```bash
ngctl msg ${int}: setpromisc 1 && ngctl msg ${int}: setautosrc 0
```

– see /usr/share/examples/netgraph/ether.bridge
One2Many

ng_one2many(4)

Hooks:

one, many1, many2, ...

Messages:

setconfig, getconfig, getstats, clrstats, getclrstats, gettable

Similar to ng_bridge() the one hook is connected to the first ng_ether(4) upper hook with ngctl mkpeer, and the subsequent ethernet interfaces’ upper hooks are connected with ngctl connect, making sure that the ethernet interfaces are in promiscuous mode and do not set the source IP.
Example: Interface Bonding

kldload ng_ether
kldload ng_one2many

intA=vr0
intB=vr1
IP=192.168.1.1/24

# start with clean slate
ngctl shut ${intA}:
ngctl shut ${intB}:

# Plumb nodes together
ngctl mkpeer ${intA}: one2many upper one
ngctl connect ${intA}: ${intA}:upper lower many0
ngctl connect ${intB}: ${intA}:upper lower lower many1

# Allow ${intB} to xmit/recv ${intA} frames
ngctl msg ${intB}: setpromisc 1
ngctl msg ${intB}: setautosrc 0

# Configure all links as up
ngctl msg ${intA}:upper setconfig "{ xmitAlg=1 failAlg=1 enabledLinks=[ 1 1 ] }"

# Bring up interface
ifconfig ${intA} ${IP}

ngctl msg ${intA}:lower getconfig
Rec'd response "getconfig" (1) from "[11]:":
Args:  { xmitAlg=1 failAlg=1 enabledLinks=[ 1 1 ] }
ETHER_IF=vr0

kldload ng_ether
kldload ng_vlan

ngctl shutdown ${ETHER_IF}:
ngctl mkpeer ${ETHER_IF}: vlan lower downstream
ngctl name ${ETHER_IF}:lower vlan
ngctl connect ${ETHER_IF}: vlan: upper nomatch

ngctl mkpeer vlan: eiface vlan123 ether
ngctl msg vlan: addfilter '{ vlan=123 hook="vlan123" }'

ngctl show vlan:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>ID</th>
<th>Num hooks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>000000038</td>
<td>3</td>
</tr>
<tr>
<td>Local hook</td>
<td>Peer name</td>
<td>Peer type</td>
<td>Peer ID</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>vlan123</td>
<td>&lt;unnamed&gt;</td>
<td>eiface</td>
<td>00000003c</td>
</tr>
<tr>
<td>nomatch</td>
<td>vr0</td>
<td>ether</td>
<td>0000000b</td>
</tr>
<tr>
<td>downstream</td>
<td>vr0</td>
<td>ether</td>
<td>0000000b</td>
</tr>
</tbody>
</table>

ng_vlan(4)

Hooks: downstream, nomatch, <arbitrary_name>

Messages: addfilter, delfilter, gettable
Ethernet Filter

ng_etf(4)

Hooks:

downstream, nomatch, <any_name>

Messages:

getstatus, setfilter

The downstream hook is usually connected to an ng Ether lower hook and the nomatch hook to the corresponding ng Ether upper; the <any_name> hook can then be captured by, say, an nghook process
Example: \texttt{ng\_etf(4)}

\begin{verbatim}

kldload ng Ether
kldload ng Eth

int=xps0

MATCH1=0x834
MATCH2=0x835

# Clean up any old connection, add etf node and name it
ngctl shutdown \{int\}:lower
ngctl mkpeer \{int\}: etf lower downstream
ngctl name \{int\}:lower myfilter

# Connect the nomatch hook to the upper part of the same interface.
# All unmatched packets will act as if the filter is not present.
ngctl connect \{int\}: myfilter: upper nomatch

# Show packets on a new hook \"newproto\" in background, ignore stdin
nghook -an myfilter: newproto &

# Filter the two random ethertypes to this newhook
ngctl msg myfilter: setfilter \"{ matchhook="newproto\" ethertype=${MATCH1} \}"
ngctl msg myfilter: setfilter \"{ matchhook="newproto\" ethertype=${MATCH2} \}"
\end{verbatim}
Example: ng_etf(4)
Berkeley Packet Filter

ng_bpf(4)

Hooks: an arbitrary number of <arbitrary_name> hooks

Messages: setprogram, getprogram,

getstats, clrstats, getclrstats

The setprogram message expects a BPF(4) filter program

bpf_prog_len=16 bpf_prog=[...]

which is generated from tcpdump -ddd output. Together with this filter one sets the
thisHook="inhook", ifMatch="matchhook", and
ifNotMatch="notmatchhook" hooks for use elsewhere.
Generating BPF Program

#!/bin/sh
i=0
{
    tcpdump -s 8192 -ddd "$@" \
    | while read line; do
        set -$- $line
        i=$(( $i + 1 ))
        if [ $i -eq 1 ]; then
            echo "bpf_prog_len=$1"
            continue
        elif [ $i -eq 2 ]; then
            echo "bpf_prog="
        fi
        echo " { code=$1 jt=$2 jf=$3 k=$4 }"
    done
    echo " ]" done
} | xargs
exit 0

tcpdump2bpf.sh tcp dst port 80
bpf_prog_len=16 bpf_prog=[ { code=40 jt=0 jf=0 k=12 } { code=21 jt=0 jf=4 k=34525 } { code=48 jt=0 jf=0 k=20 } ... { code=6 jt=0 jf=0 k=8192 } { code=6 jt=0 jf=0 k=0 } ]
ng_ipfw(4), ng_tag(4)

- **ng_ipfw(4)** interface between IPFW and Netgraph
  
  **Hooks:** arbitrary number, name must be numeric
  
  **Messages:** none (only generic)

- **ng_tag(4)**
  
  **Hooks:** arbitrary number and name
  
  **Messages:** sethookin, gethookin, sethookout, gethookout,
  
  getstats, clrstats, getclrstats
BPF + IPFW + TAG = L7 Filter
RTFM ng_tag(4)

DirectConnect P2P TCP payloads contain the string "$Send!", filter these out with ng_bpf(4), tag the packets with ng_tag(4), and then block them with an ipfw rule hooking them to a ng_ipfw(4) node.

```
kldload ng_ipfw; kldload ng_bpf; kldload ng_tag
ngctl mkpeer ipfw: bpf 41 ipfw
ngctl name ipfw:41 dcbpf
ngctl mkpeer dcbpf: tag matched th1
ngctl name dcbpf:matched ngdc

grep MTAG_IPFW /usr/include/netinet/ip_fw.h
#define MTAG_IPFW 1148380143 /* IPFW-tagged cookie */

ngctl msg ngdc: sethookin { thisHook="th1" ifNotMatch="th1" }
ngctl msg ngdc: sethookout { thisHook="th1" 
  tag_cookie=1148380143 tag_id=412 }
ngctl msg dcbpf: setprogram { thisHook="matched" 
  ifMatch="ipfw" bpf_prog_len=1 
  bpf_prog=[ { code=6 k=8192 } ] }

tcpdump2bpf.sh "ether[40:2]=0x244c && ether[42:4]=0x6f636b20"
  bpf_prog_len=6 bpf_prog=[ { ... { code=6 jt=0 jf=0 k=8192 } ... } ]

ipfw add 100 netgraph 41 tcp from any to any iplen 46
ipfw add 110 reset tcp from any to any tagged 412
sysctl net.inet.ip.fw.one_pass=0
```

"$Lock"
BPF + IPFW + TAG = L7 Filter

DirectConnect P2P TCP payloads contain the string “$Send|”, filter these out with \texttt{ng\_bpf(4)}, tag the packets with \texttt{ng\_tag(4)}, and then block them with an \texttt{ipfw} rule hooking them to a \texttt{ng\_ipfw(4)} node.

\begin{verbatim}
  ipfw add 200 allow ip from any to any
  kldload ng_ipfw; kldload ng_bpf; kldload ng_tag

  ngctl mkpeer ipfw: bpf 41 ipfw
  ngctl name ipfw:41 dcbpf
  ngctl mkpeer dcbpf: tag matched th1
  ngctl name dcbpf:matched ngdc

grep MTAG_IPFW /usr/include/netinet/ip_var.h
#define MTAG_IPFW 1148380143 /* IPFW-tagged cookie */

  ngctl msg ngdc: sethookin { thisHook="th1" ifNotMatch="th1" }
  ngctl msg ngdc: sethookout { thisHook="th1" 
                             tag_cookie=1148380143 tag_id=412 }
  ngctl msg dcbpf: setprogram { thisHook="matched" 
                                ifMatch="ipfw" bpf_prog_len=6 
                                bpf_prog=[ { ... (see below) ... } ] }

  tcpdump2bpf.sh "ether[40:2]=0x2453 && ether[42:4]=0x656e647c"
bpf_prog_len=6 bpf_prog=[ bpf_prog_len=6 bpf_prog=[ { code=40 jt=0jf=0 k=40 }
{ code=21 jt=0jf=3 k=9299 } { code=32 jt=0jf=0 k=42 } { code=21 jt=0jf=1 
k=1701733500 } { code=6 jt=0jf=0 k=8192 } { code=6 jt=0jf=0 k=0 } ]

  ipfw add 100 netgraph 41 tcp from any to any iplen 46
  ipfw add 110 reset tcp from any to any tagged 412

  sysctl net.inet.ip.fw.one_pass=0
\end{verbatim}
BPF + IPFW + TAG = L7 Filter

```
ngctl mkpeer ipfw: bpf 41 ipfw
ngctl name ipfw:41 dcbpf
ngctl mkpeer dcbpf: tag matched th1
ngctl name dcbpf:matched ngdc
ngctl msg ngdc: sethookin   { thisHook="th1" ifNotMatch="th1" }  
ngctl msg ngdc: sethookout  { thisHook="th1" 
                              tag_cookie=1148380143 tag_id=412 }
ngctl msg dcbpf: setprogram { thisHook="matched"
                              ifMatch="ipfw" bpf_prog_len=6
                              bpf_prog=[ { ... } ] }

ipfw add 100 netgraph 41 tcp from any to any iplen 46
ipfw add 110 reset tcp from any to any tagged 412

sysctl net.inet.ip.fw.one_pass=0
```
Service Control Engine

http://shveitser.blogspot.com/

ng_state

(December 2011)

- User bandwidth policing
- Possibility of having common CIR for group of IP addresses
- Traffic classification and traffic policing according to class
- Netflow v5 export
- WEB traffic redirect to display informational messages
- Flood and attack protection (internal and external)
- P2P traffic detection by behavior

Number of supported users - 65k.
Tested on 5-7Gbits with 4 million flows.
Cisco Service Control Engine Clone

(December 2011)

- User bandwidth policing
- Possibility of having common CIR for group of IP addresses
- Traffic classification and traffic policing according to class
- Netflow v5 export
- WEB traffic redirect to display informational messages
- Flood and attack protection (internal and external)
- P2P traffic detection by behavior

Number of supported users - 65k.
Tested on 5-7Gbits with 4 million flows.
ng_state

- Ether
  - Em0:
    - One to many
      - Many1
      - Many2
    - Car
      - Down1
      - Down2
      - Up1
      - Up2
    - State
      - Newflows_down
      - Newflows_up
      - Match1
      - Match2
      - Match3
      - Notmatch1
      - Notmatch2
      - Notmatch3
    - BPF
      - In1
      - In2
      - In3

- Ether
  - Em1:
    - One to many
      - Many1
      - Many2

Notes:
- After classify
- Before classify
- 1st rule not matched
- But 2nd do
MPD Champion of netgraph(3) User Library

http://sourceforge.net/projects/mpd/files/

“MPD is a netgraph based PPP implementation for FreeBSD. MPD5 supports thousands of Sync, Async, PPTP, L2TP, PPPoE, TCP and UDP links in client, server and access concentrator (LAC/PAC/TSA) modes. It is very fast and functional”

Uses netgraph User Library (libnetgraph, -lnetgraph)

man 3 netgraph

Scales well

Built-in web monitor
• **modem** to connect using different asynchronous serial connections, including modems, ISDN terminal adapters, and null-modem. Mpdp includes event-driven scripting language for modem identification, setup, manual server login, etc.

• **pptp** to connect over the Internet using the Point-to-Point Tunnelling Protocol (PPTP). This protocol is supported by the most OSes and hardware vendors

• **l2tp** to connect over the Internet using the Layer Two Tunnelling Protocol (L2TP). L2TP is a PPTP successor supported with modern clients and servers

• **pppoed** to connect over an Ethernet port using the PPP-over-Ethernet (PPPoE) protocol. This protocol is often used by DSL providers

• **tcp** to tunnel PPP session over a TCP connection. Frames are encoded in the same was as asynchronous serial connections

• **udp** to tunnel PPP session over a UDP connection. Each frame is encapsulated in a UDP datagram packet

• **ng** to connect to netgraph nodes
MPD Multi-link PPP Daemon Configuration

• MPD operates on several protocol layers (OSI layering) and acts as a PPP terminator with Radius or other AAA and IP accounting; usually used for IP but could be used for other protocols

• Can act in PPP repeater mode too (L2TP or PPTP access concentrator)

• PPP Terminator Layers:
  Interface – NCPs – Compression/Encryption – Bundle – Links

A set of Links is a Bundle connecting to the peer, after optional compression/encryption the NCP (IPCP or IPv6CP) layer presents the interface which is visible via the ifconfig command
**l2tp_slave:**

create bundle template B
set ipcp yes vjcomp
set ipcp ranges 10.10.10.2/32 10.10.10.1/32
set bundle enable compression

create link static L1 l2tp
set l2tp self 10.10.10.11 1701
set l2tp peer 10.10.10.10 1701
set l2tp enable outcall
set l2tp hostname slave
set l2tp secret MySecret
set link max-redial 0
set link action bundle B
l2tp_slave:
create bundle template B
set ipcp yes vjcomp
set ipcp ranges 10.10.10.2/32 10.10.10.1/32
set bundle enable compression

create link static L1 l2tp
set l2tp self 10.10.10.11 1701
set l2tp peer 10.10.10.10 1701
set l2tp enable outcall
set l2tp hostname slave
set l2tp secret MySecret
set link max-redial 0
set link action bundle B
Creating Custom Node Type

• “Only” two steps:
  1. Define your new custom struct `ng_type`
  2. `NETGRAPH_INIT(tee, &ng_tee_typestruct)`

• `sys/netgraph/ng_tee.c` is a good example

• `netgraph.h`
  Defines basic netgraph structures

• `ng_message.h`
  Defines structures and macros for control messages, here you see how the generic control messages are implemented
static struct ng_type typestruct = {
    .version = NG_ABI_VERSION,
    .name = NG_XXX_NODE_TYPE,
    .constructor = ng_XXX_constructor,
    .rcvmsg = ng_XXX_rcvmsg,
    .shutdown = ng_XXX_shutdown,
    .newhook = ng_XXX_newhook,
    /*
    .findhook = ng_XXX_findhook,
    .connect = ng_XXX_connect,
    .rcvdata = ng_XXX_rcvdata,
    .disconnect = ng_XXX_disconnect,
    .cmdlist = ng_XXX_cmdlist,
    */
};
About mbuf(9)

An mbuf is a basic unit of memory management in kernel. Network packets and socket buffers use mbufs. A network packet may span multiple mbufs arranged into a linked list, which allows adding or trimming headers with minimal overhead.

Netgraph must have M_PKTHDR flag set, i.e., struct pkthdr m_pkthdr is added to the mbuf header. This means you also have access and are responsible for the data packet header information.

`m_pullup(mbuf, len)` is expensive, so always check if is needed:

```c
struct foobar *f;

if (m->m_len < sizeof(*f) && (m = m_pullup(m, sizeof(*f))) == NULL) {
    NG_FREE_META(meta);
    return (ENOBUFFS);
}
f = mtod(m, struct foobar *);
...
```
User-space Prototyping

- User library libnetgraph – netgraph(3) – has a sufficiently similar API to the netgraph(4) kernel API, so that node types can be prototyped in user space.

- Use an intermediate ng_tee(4) node and attach nghook(8) for debugging.
Q & A

(1) Netgraph in a historical context
(2) Getting to know netgraph
(3) Working with netgraph
(4) Details of frequently used netgraph node types
(5) Examples using netgraph nodes as building blocks
(6) Investigate some more sophisticated examples
(7) Guidelines for implementing custom node types
THANK YOU for attending the Introduction to NETGRAPH on FreeBSD Systems Tutorial
‘All About Netgraph’ (complete introduction to FreeBSD netgraph)
http://people.freebsd.org/~julian/netgraph.html

‘Netgraph in 5 and beyond’
A BAFUG talk where Julian Elischer points out things he fixed as FreeBSD transition from 4.x to 5.x (slides and movie)
http://people.freebsd.org/~julian/BAFUG/talks/Netgraph/

Re: diagram of 4.10 layer 2 spaghetti
http://www.mail-archive.com/freebsd-net@freebsd.org/msg16970.html
http://people.freebsd.org/~julian/layer2c.pdf

‘Netgraph7’ on DragonFly BSD
http://gitweb.dragonflybsd.org/~nant/dragonfly.git/shortlog/refs/heads/netgraph7

Debugging a netgraph node
STREAMS

http://cm.bell-labs.com/cm/cs/who/dmr/st.html (Initial article by Dennis Ritchie)
http://www.linuxjournal.com/article/3086 (LiS: Linux STREAMS)
http://en.wikipedia.org/wiki/STREAMS

“Hacking” The Whistle InterJet © (i486 internet access appliance 1996)
http://www.anastrophe.com/~paul/wco/interjet/ (Information on the Whistle Interjet)

6WINDGate™ Linux
Closed source virtual network blocks (VNB) stack derived from netgraph technology (press release 2007)
http://www.windriver.com/partner-validation/1B-2%20%20-%206WINDGate%20Architecture%20Overview%20v1.0.pdf

(ACM TOCS’11) Application-tailored I/O with Streamline

Marko Zec – network emulation using the virtualized network stack in FreeBSD
http://www.imunes.net/virtnet/eurobsdcon07_tutorial.pdf (Network stack virtualization for FreeBSD 7.0, EuroBSDCon Sept 2007)
http://meetbsd.org/files/2_05_zec.pdf (Network emulation using the virtualized network stack in FreeBSD, MeetBSD July 2010)
http://www.youtube.com/watch?v=wh09MirPd5Y (MeetBSD talk, July 2010)
FreeBSD CVS Repository (to see all the currently available netgraph modules)
http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/netgraph/
Latest addition ng_patch(4) 2010 by Maxim Ignatenko:
http://www.mail-archive.com/freebsd-net@freebsd.org/msg32164.html

A Gentlemen's Agreement
Assessing The GNU General Public License and its Adaptation to Linux
by Douglas A. Hass, Chicago-Kent Journal of Intellectual Property, 2007 mentions that netgraph has a good license model

Subject: NetBSD port of the freebsd netgraph environment
NetBSD 1.5, that is (port from the FreeBSD 4.3 netgraph)
http://mail-index.netbsd.org/tech-net/2001/08/17/0000.html

Netgraph on Debian GNU / kFreeBSD (in russian)
http://morbow.blogspot.com/2011/02/netgraph-debian.html
Subject: [PATCH] ng_tag - new netgraph node, please test (L7 filtering possibility)

"Yes, netgraph always was a semi-programmer system"
Includes an example of in-kernel L7 (bittorrent) pattern matching filter using ng_bpf, ng_mtag and ipfw

(SIGCOMM Poster 2011 Winner) netmap: fast and safe access to network for user programs
http://info.iet.unipi.it/~luigi/netmap/rizzo-ancs.pdf

Re: option directive and turning on AOE
In a discussion on ATA over Ethernet (frame type 0x88a2) other questions came up:
Does netgraph have locking issues? Is netgraph performant? – it depends:

ng_state
http://shveitser.blogspot.com/

Mikrotik's EoIP tunneling
http://imax.in.ua/ng_mikrotik_eoip/