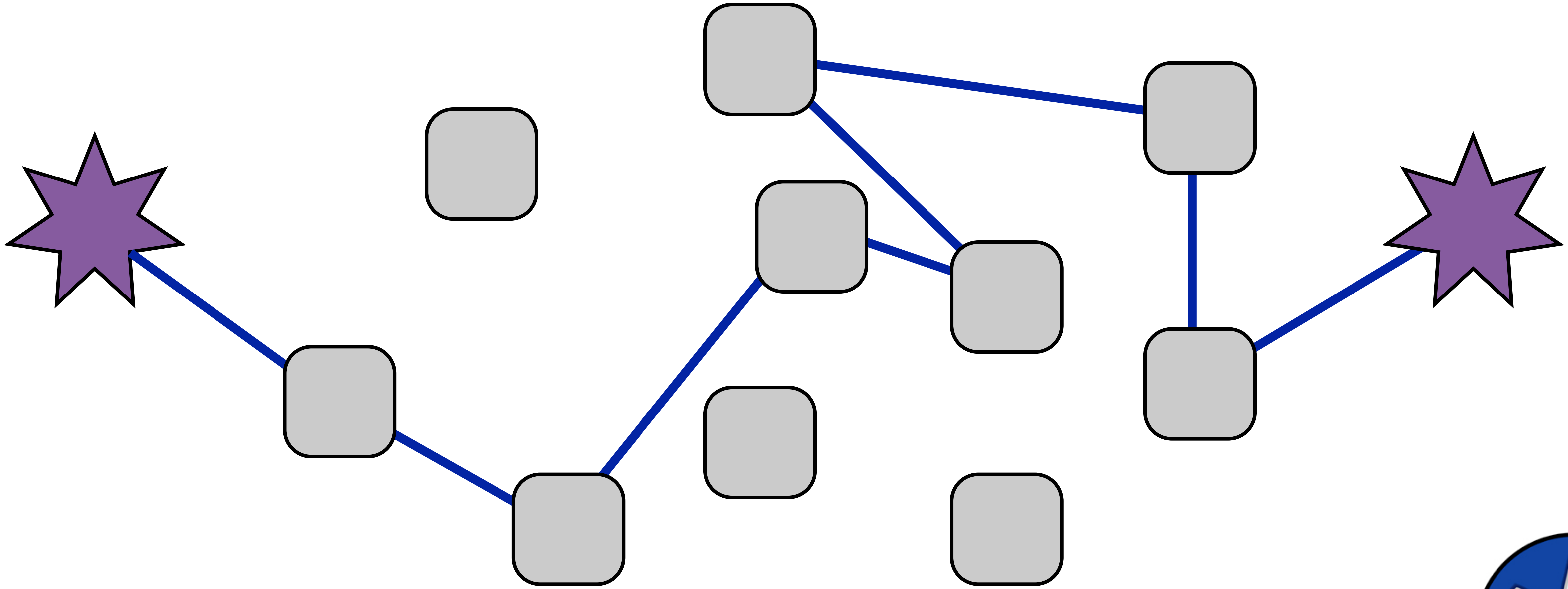
Layer Four Traceroute (and related tools)

A modern, flexible path-discovery solution with advanced features for network (reverse) engineers

VOSTROM

So, what is path discovery and why is it important?

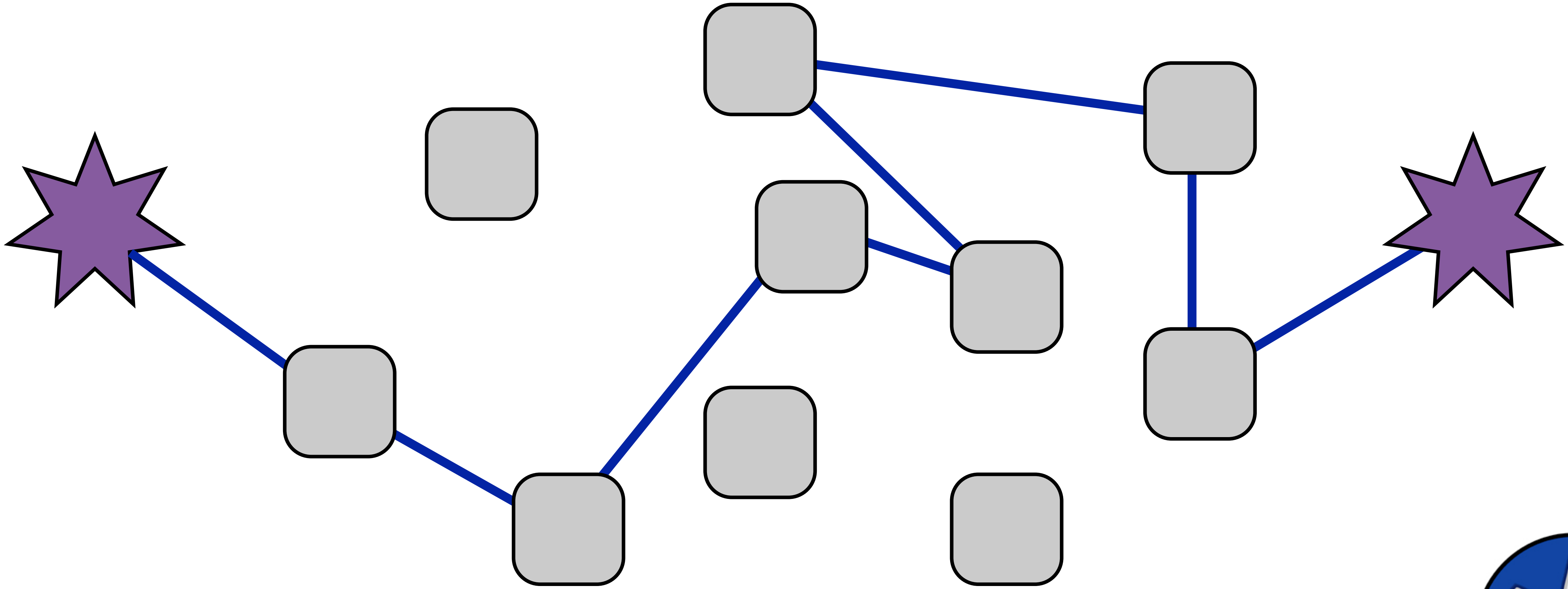
Path discovery is the act of finding the path packets take between two network hosts



VOSTROM

So, what is path discovery and why is it important?

It’s important because as networks get larger and more complicated, finding the gateway that is mishandling your communications can be difficult



VOSTROM

How do we perform path discovery?

The common ‘**traceroute**’ program was created as a simple way to find the path packets will take between you and another network host

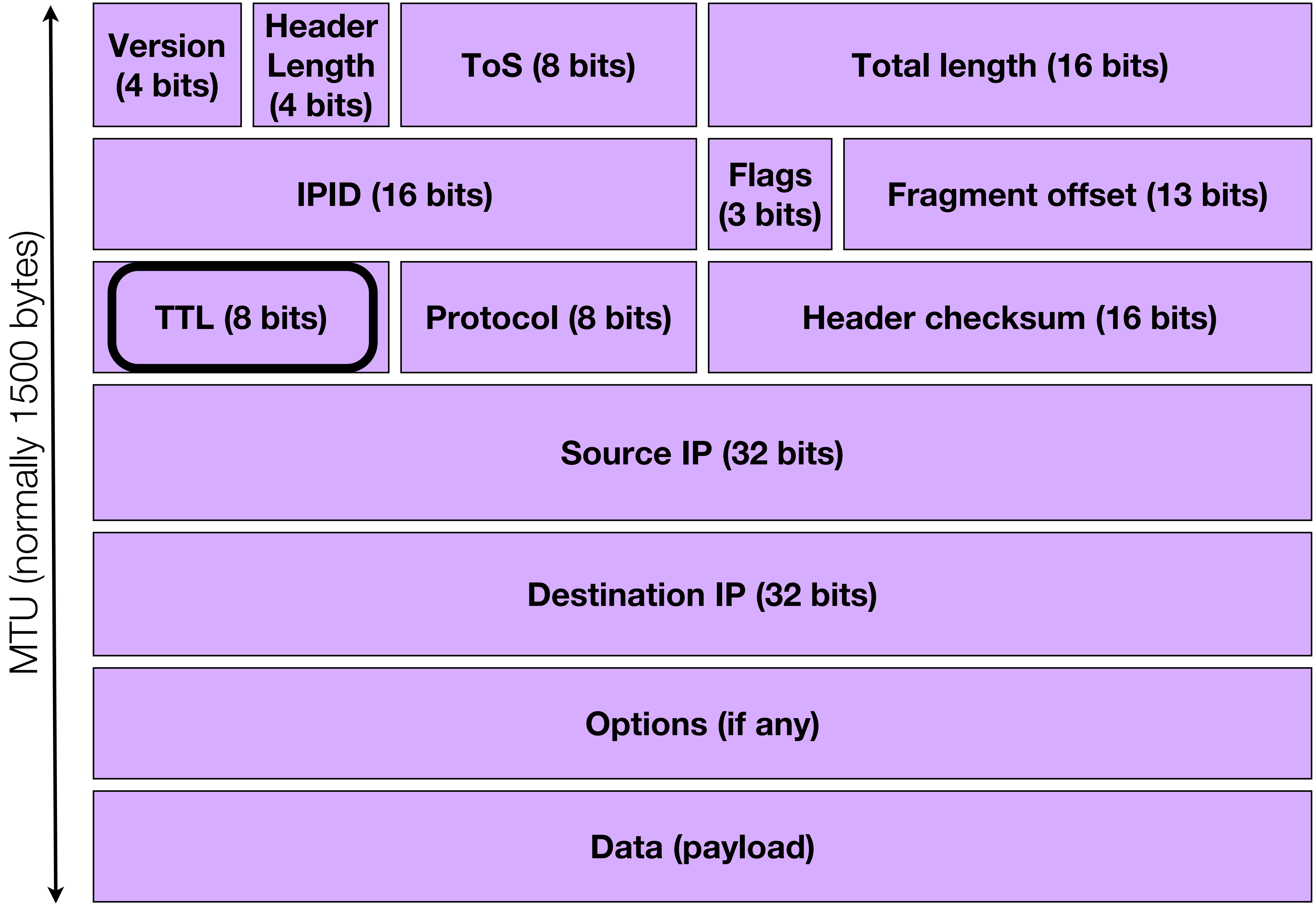
Invented by Van Jacobson at LBL in 1988

It has remained largely unchanged ...

VOSTROM

So, how does traceroute work?

IPv4 Datagram



Version (4 bits) | Header Length (4 bits) | ToS (8 bits) | Total length (16 bits)

IPID (16 bits) | Flags (3 bits) | Fragment offset (13 bits)

TTL (8 bits) | Protocol (8 bits) | Header checksum (16 bits)

Source IP (32 bits)

Destination IP (32 bits)

Options (if any)

Data (payload)

NOTE: In IPv6, TTL is now called “hop limit”

How do routers deal with TTLs?

RFC 1812, page 84:

A router MUST generate a Time Exceeded message Code 0 (In Transit) when it discards a packet due to an expired TTL field. A router MAY have a per-interface option to disable origination of these messages on that interface, but that option MUST default to allowing the messages to be originated.

RFC 1812, page 85:

The Time-to-Live (TTL) field of the IP header is defined to be a timer limiting the lifetime of a datagram....When a router forwards a packet, it MUST reduce the TTL by at least one. If it holds a packet for more than one second, it MAY decrement the TTL by one for each second.

If the TTL is reduced to zero (or less), the packet MUST be discarded, and if the destination is not a multicast address the router MUST send an ICMP Time Exceeded message, Code 0 (TTL Exceeded in Transit) message to the source. Note that a router MUST NOT discard an IP unicast or broadcast packet with a non-zero TTL merely because it can predict that another router on the path to the packet's final destination will decrement the TTL to zero.

So, we can gradually increment the TTL to discover each hop, one-by-one. The process looks like this:

- Send packet with TTL 1 to endpoint/target

- First router in the path gets the packet, decrements the TTL by 1 (making it zero), discards the packet, and sends us back an ICMP "TTL exceeded in transit" message, thereby exposing itself in our path to the target

- We have discovered hop 1 (TTL1)

- Now, send a packet with a TTL of 2 and so on..... repeating until we receive a response from the target

VOSTROM

Works great. But it begs questions...

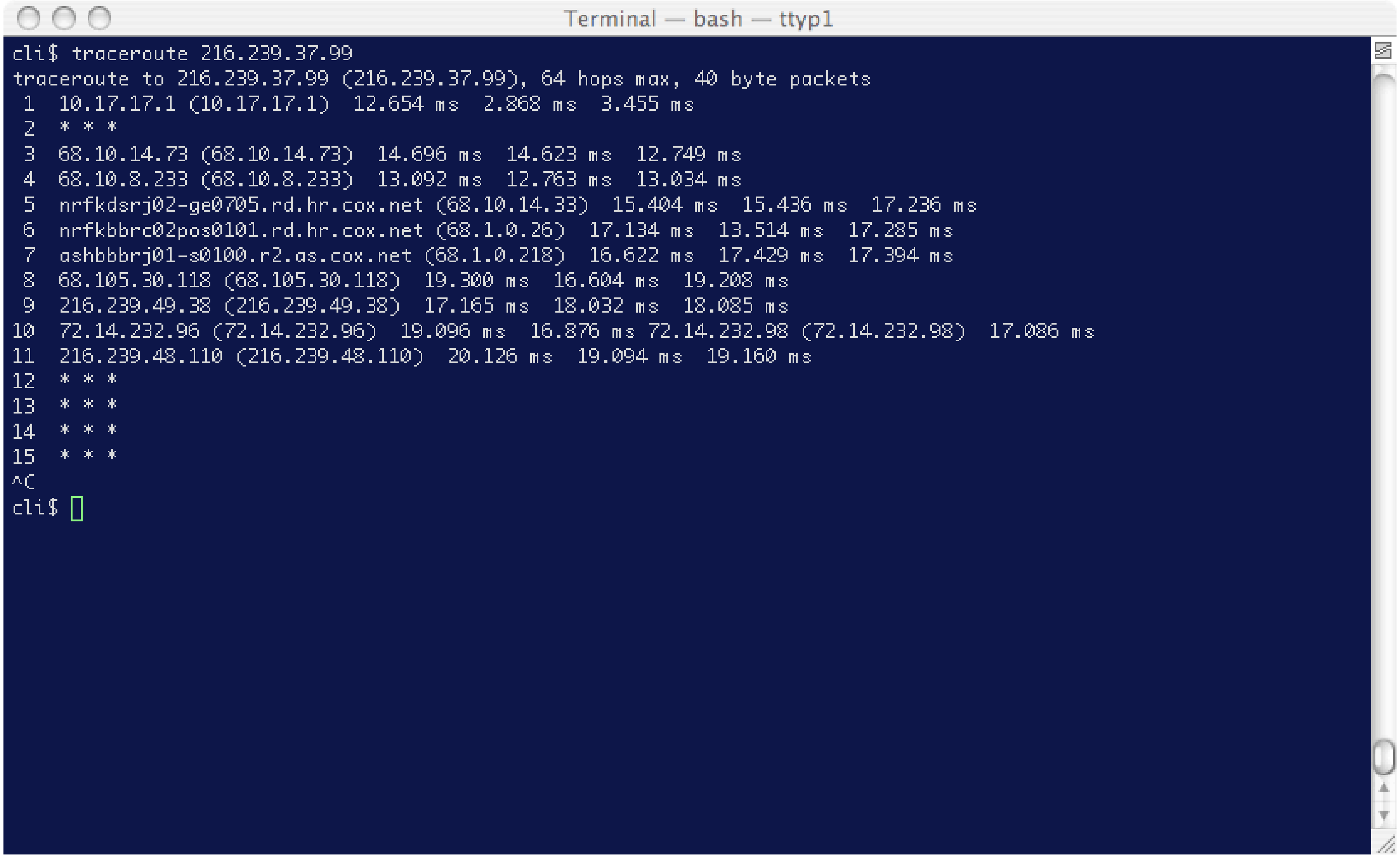
- What layer-4 protocol do we use to send the TTL-doctored IP packets? Will it be filtered?
- If we send/receive one TTL at a time, it's slow. But if we send more than one TTL before waiting for a reply, how do we know which hop is replying to which TTL?

In 1988...

- We use UDP because it’s *really* easy to implement
- We’ll send more than one packet before waiting for replies, but we’ll send each packet to a different UDP port so we can tell the replies apart.
  - This works because routers send back the contents of the layer-4 packet header INSIDE the ICMP “TTL exceeded” messages

VOSTROM

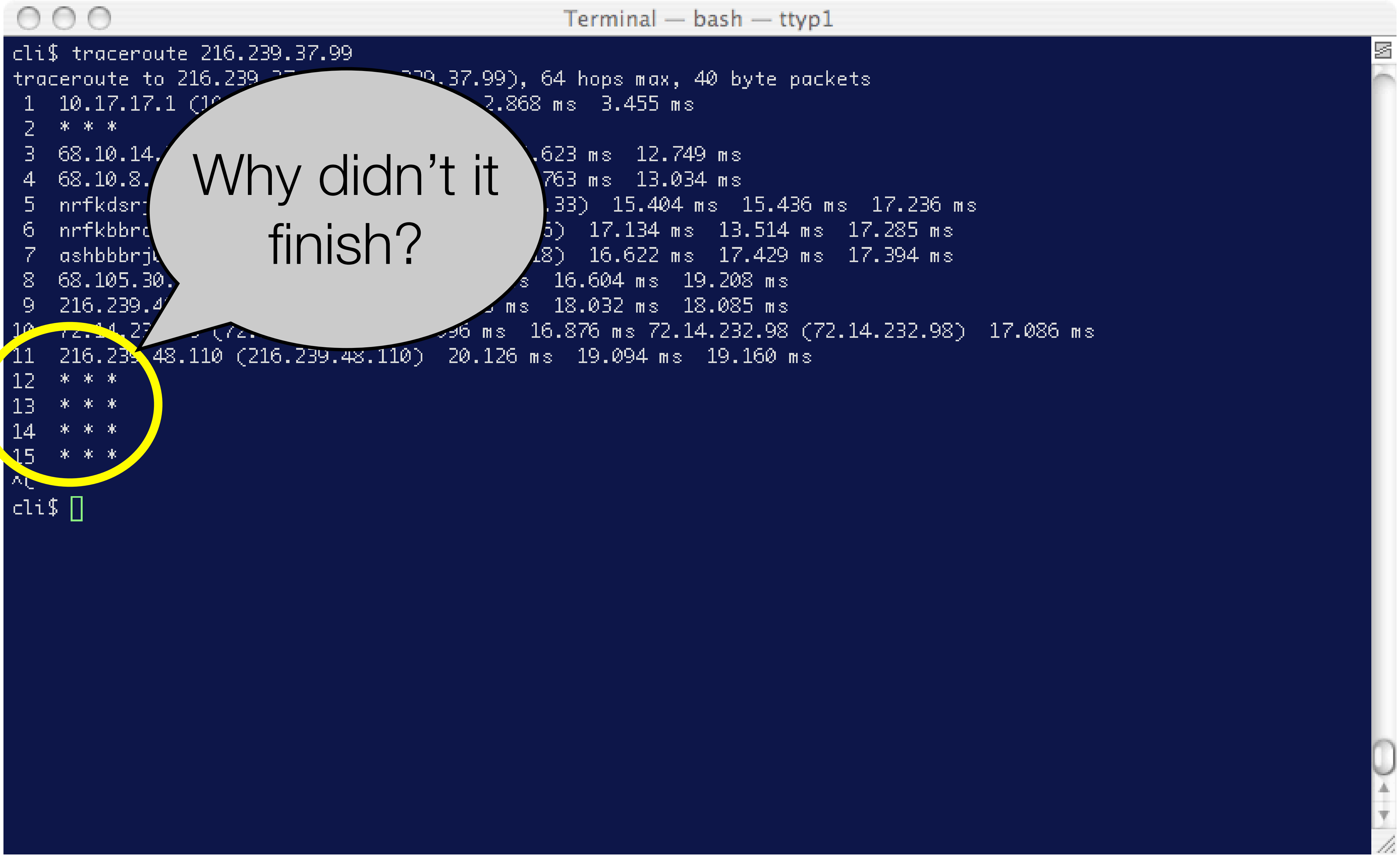
Look familiar?



```
cli$ traceroute 216.239.37.99
traceroute to 216.239.37.99 (216.239.37.99), 64 hops max, 40 byte packets
 1  10.17.17.1 (10.17.17.1)  12.654 ms  2.868 ms  3.455 ms
 2  * * *
 3  68.10.14.73 (68.10.14.73)  14.696 ms  14.623 ms  12.749 ms
 4  68.10.8.233 (68.10.8.233)  13.092 ms  12.763 ms  13.034 ms
 5  nrfkdsrj02-ge0705.rd.hr.cox.net (68.10.14.33)  15.404 ms  15.436 ms  17.236 ms
 6  nrfkbbrc02pos0101.rd.hr.cox.net (68.1.0.26)  17.134 ms  13.514 ms  17.285 ms
 7  ashbbbrj01-s0100.r2.as.cox.net (68.1.0.218)  16.622 ms  17.429 ms  17.394 ms
 8  68.105.30.118 (68.105.30.118)  19.300 ms  16.604 ms  19.208 ms
 9  216.239.49.38 (216.239.49.38)  17.165 ms  18.032 ms  18.085 ms
10  72.14.232.96 (72.14.232.96)  19.096 ms  16.876 ms 72.14.232.98 (72.14.232.98)  17.086 ms
11  216.239.48.110 (216.239.48.110)  20.126 ms  19.094 ms  19.160 ms
12  * * *
13  * * *
14  * * *
15  * * *
^C
cli$ 
```

VOSTROM

Notice anything?



```
cli$ traceroute 216.239.37.99
traceroute to 216.239              239.37.99), 64 hops max, 40 byte packets
 1  10.17.17.1 (1              2.868 ms  3.455 ms
 2  * * *
 3  68.10.14.                    .623 ms  12.749 ms
 4  68.10.8.                     763 ms  13.034 ms
 5  nrfkdsr                      .33)  15.404 ms  15.436 ms  17.236 ms
 6  nrfkbbr                      5)  17.134 ms  13.514 ms  17.285 ms
 7  ashbbbrj                     18)  16.622 ms  17.429 ms  17.394 ms
 8  68.105.30.                 s  16.604 ms  19.208 ms
 9  216.239.4                ms  18.032 ms  18.085 ms
10  72.14.23   (72.          96 ms  16.876 ms 72.14.232.98 (72.14.232.98)  17.086 ms
11  216.239.48.110 (216.239.48.110)  20.126 ms  19.094 ms  19.160 ms
12  * * *
13  * * *
14  * * *
15  * * *
^C
cli$
```

Why didn’t it finish?

Things have changed since 1988

Challenges and Limitations (changes since 1988)

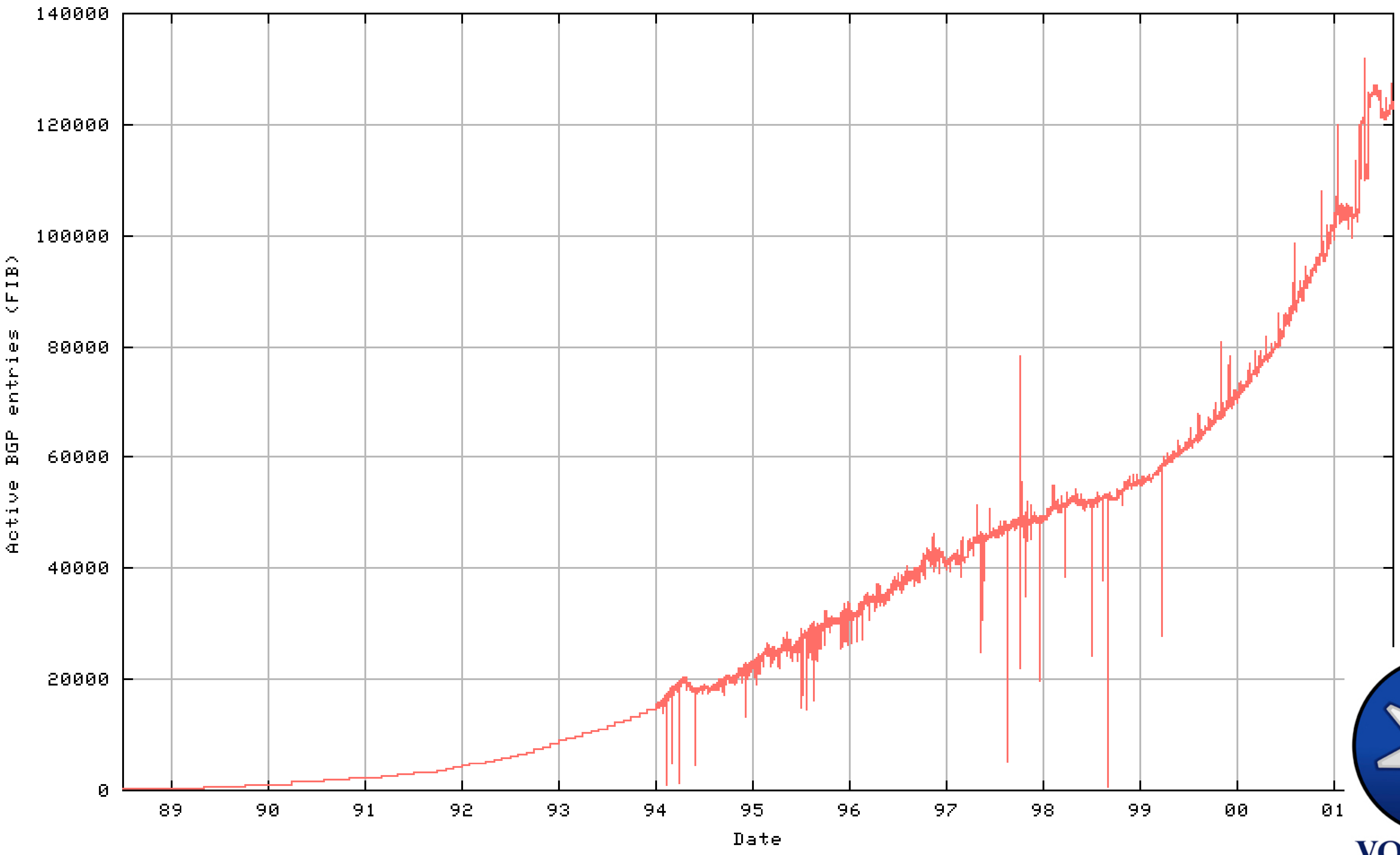
- Networks filter packets (firewalls were invented)
- Network/Port Address Translation widely adopted
- The TCP protocol has become the heavy lifter

Challenges and Limitations (changes since 1988)

- Routers differentiate/prioritize between TCP/UDP/ICMP
- Network allocations are disparate and organized by ASN, not class-based addressing or even CIDR
- Can’t trust whois registry network data when it comes to private networks (ISPs)
- There is no “what’s causing my connectivity problem” network tool or even a standard workflow to follow

VOSTROM

Oh, and the backbone is growing...



VOSTROM

We need a new path discovery tool

We need a new path discovery tool

- We need to be able to differentiate between different layer-4 protocols (TCP, UDP, ICMP)

- Networks are big and non-contiguous in terms of IPv4 address space. We need to rationalize where our packets are based on autonymous system (AS) boundaries, not just just IP addresses

- We know routing changes frequently and network registries like the RADB don't usually have the right information, so we also need a new tool to track routing changes and registry information as things change

- We're in a hurry, so path discovery should be fast

Enter **Layer Four Traceroute**

and **Prefix WhoIs**

Layer Four Traceroute: Major Features

- Supports path discovery using multiple layer-4 protocols
- Understands “state” to discover firewalls in the path
- Understands “load balancing” to discover LBs
- Can connect to multiple sources to resolve AS information
- Supports esoteric characteristics such as IP ToS field
- Fast and extremely user-configurable

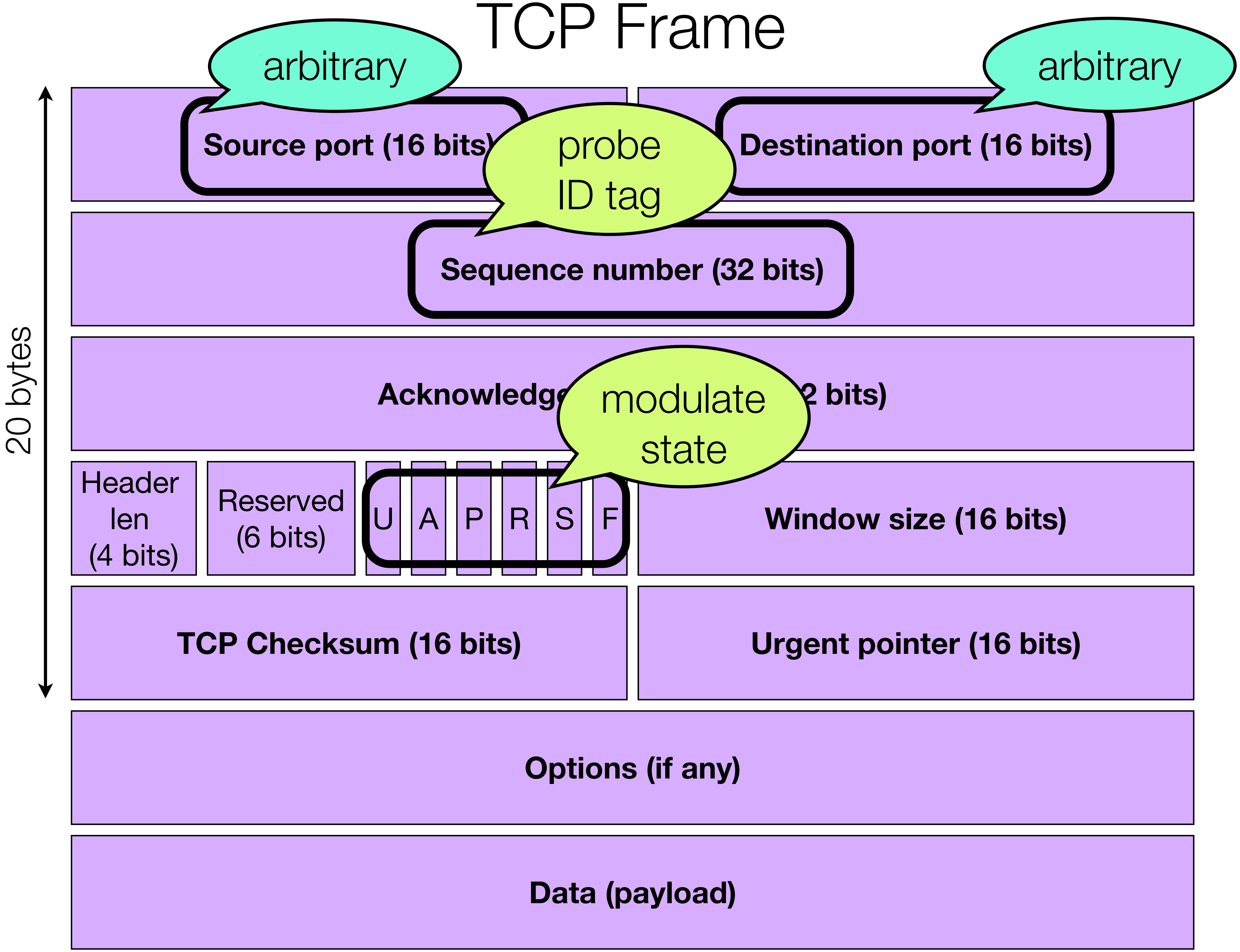
Layer Four Traceroute: Layer 4?

- Can send both TCP and UDP probes
- Can listen for TCP/UDP/ICMP responses
- Can modulate “state” in probes to detect filters
- Can take advantage of layer-4 protocols’ specific attributes to increase speed and target precision
    - Find arbitrary src/dst combos that are unfiltered
    - Speed increases of **2-20x**

VOSTROM

Let’s take a look at LFT’s average probe...

TCP Frame



arbitrary

Source port (16 bits)

probe ID tag

arbitrary

Destination port (16 bits)

Sequence number (32 bits)

Acknowledge 2 bits)

modulate state

Header len (4 bits)

Reserved (6 bits)

U A P R S F

Window size (16 bits)

TCP Checksum (16 bits)

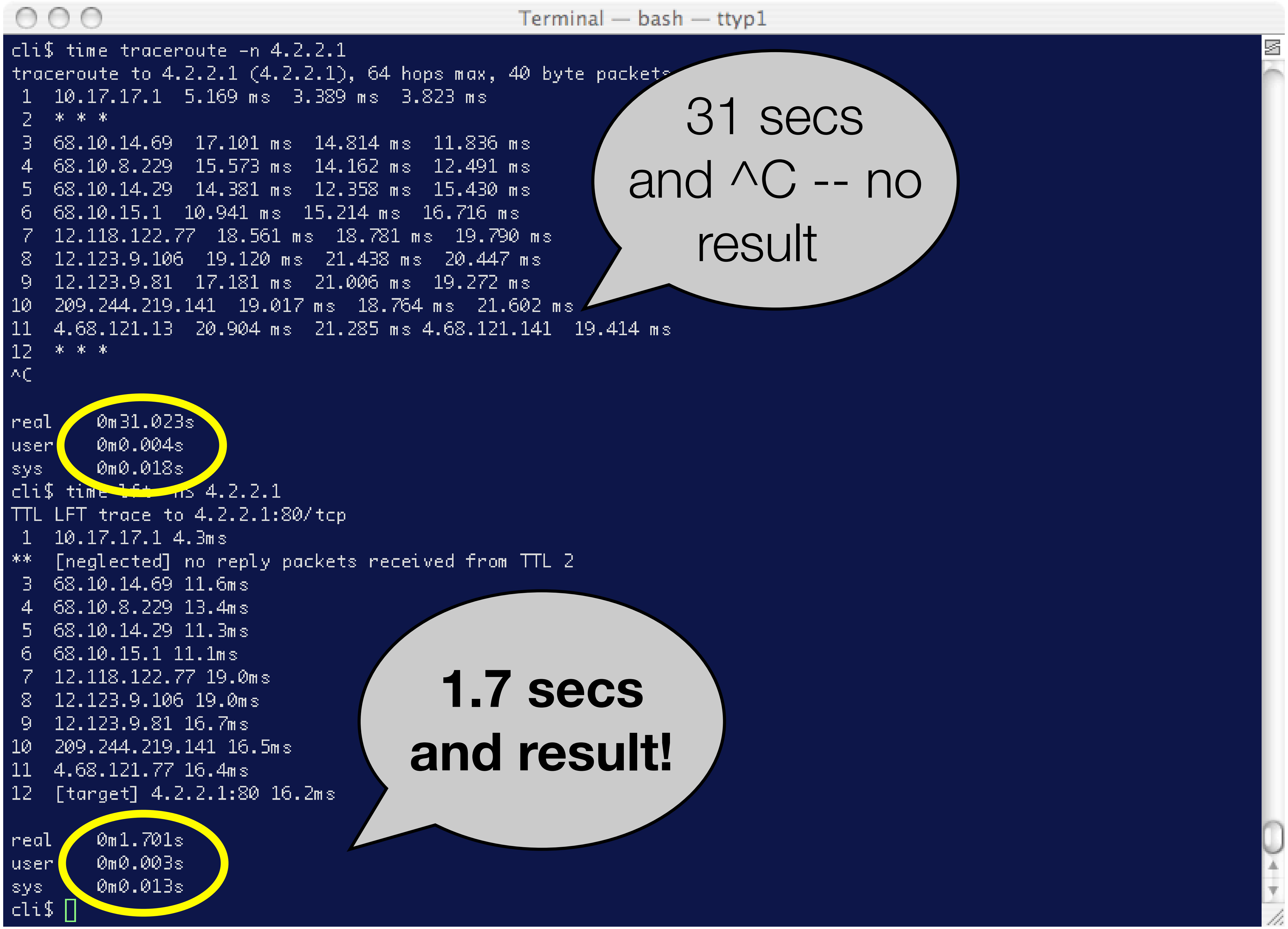
Urgent pointer (16 bits)

Options (if any)

Data (payload)

20 bytes

Example of LFT's Speed

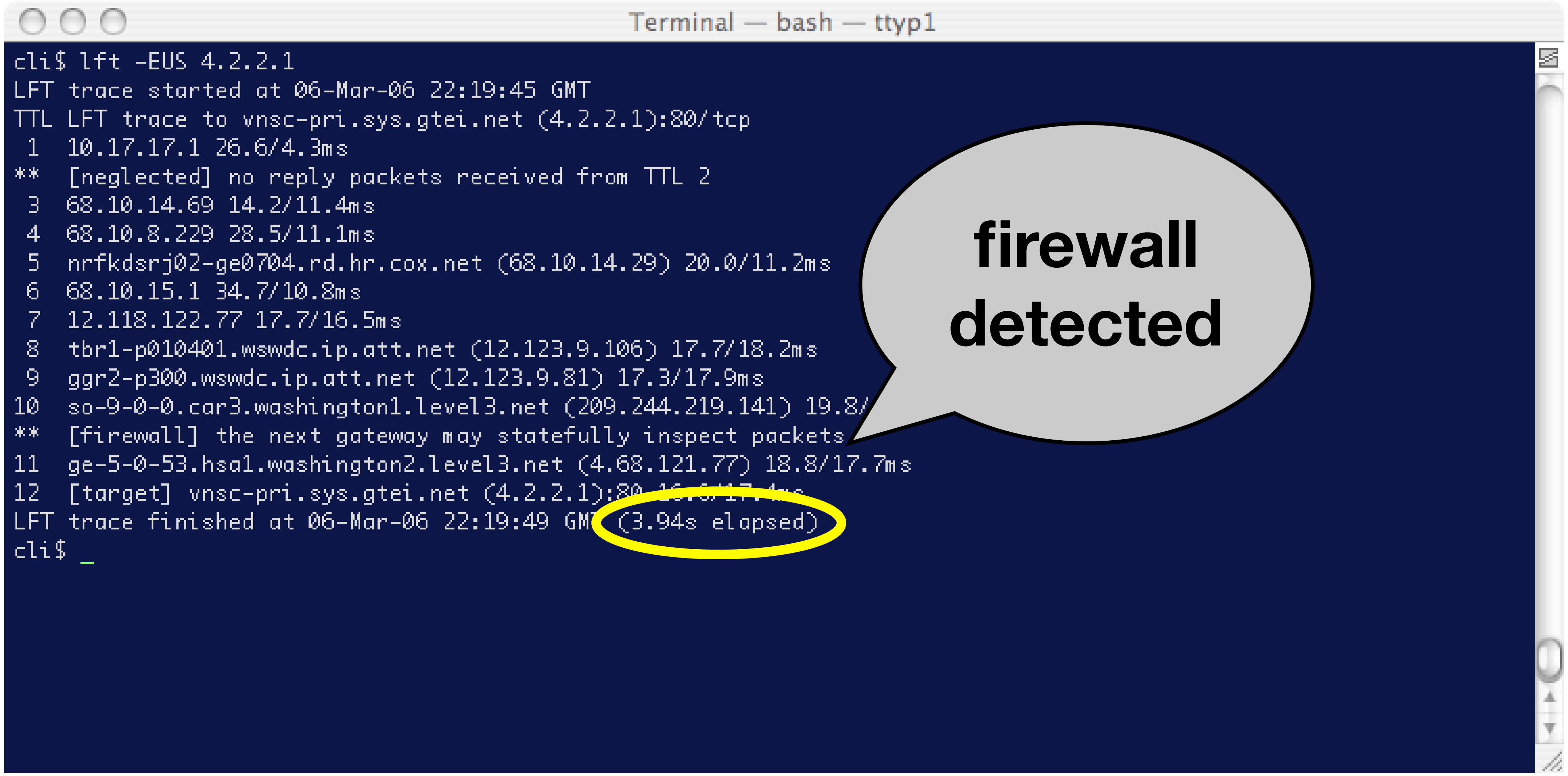


```
cli$ time traceroute -n 4.2.2.1
traceroute to 4.2.2.1 (4.2.2.1), 64 hops max, 40 byte packets
 1  10.17.17.1  5.169 ms  3.389 ms  3.823 ms
 2  * * *
 3  68.10.14.69  17.101 ms  14.814 ms  11.836 ms
 4  68.10.8.229  15.573 ms  14.162 ms  12.491 ms
 5  68.10.14.29  14.381 ms  12.358 ms  15.430 ms
 6  68.10.15.1  10.941 ms  15.214 ms  16.716 ms
 7  12.118.122.77  18.561 ms  18.781 ms  19.790 ms
 8  12.123.9.106  19.120 ms  21.438 ms  20.447 ms
 9  12.123.9.81  17.181 ms  21.006 ms  19.272 ms
10  209.244.219.141  19.017 ms  18.764 ms  21.602 ms
11  4.68.121.13  20.904 ms  21.285 ms 4.68.121.141  19.414 ms
12  * * *
^C

real    0m31.023s
user    0m0.004s
sys     0m0.018s
cli$ time lft -ns 4.2.2.1
TTL LFT trace to 4.2.2.1:80/tcp
 1  10.17.17.1 4.3ms
**  [neglected] no reply packets received from TTL 2
 3  68.10.14.69 11.6ms
 4  68.10.8.229 13.4ms
 5  68.10.14.29 11.3ms
 6  68.10.15.1 11.1ms
 7  12.118.122.77 19.0ms
 8  12.123.9.106 19.0ms
 9  12.123.9.81 16.7ms
10  209.244.219.141 16.5ms
11  4.68.121.77 16.4ms
12  [target] 4.2.2.1:80 16.2ms

real    0m1.701s
user    0m0.003s
sys     0m0.013s
cli$ 
```

31 secs and ^C -- no result

1.7 secs and result!

Example of LFT’s Advanced Features

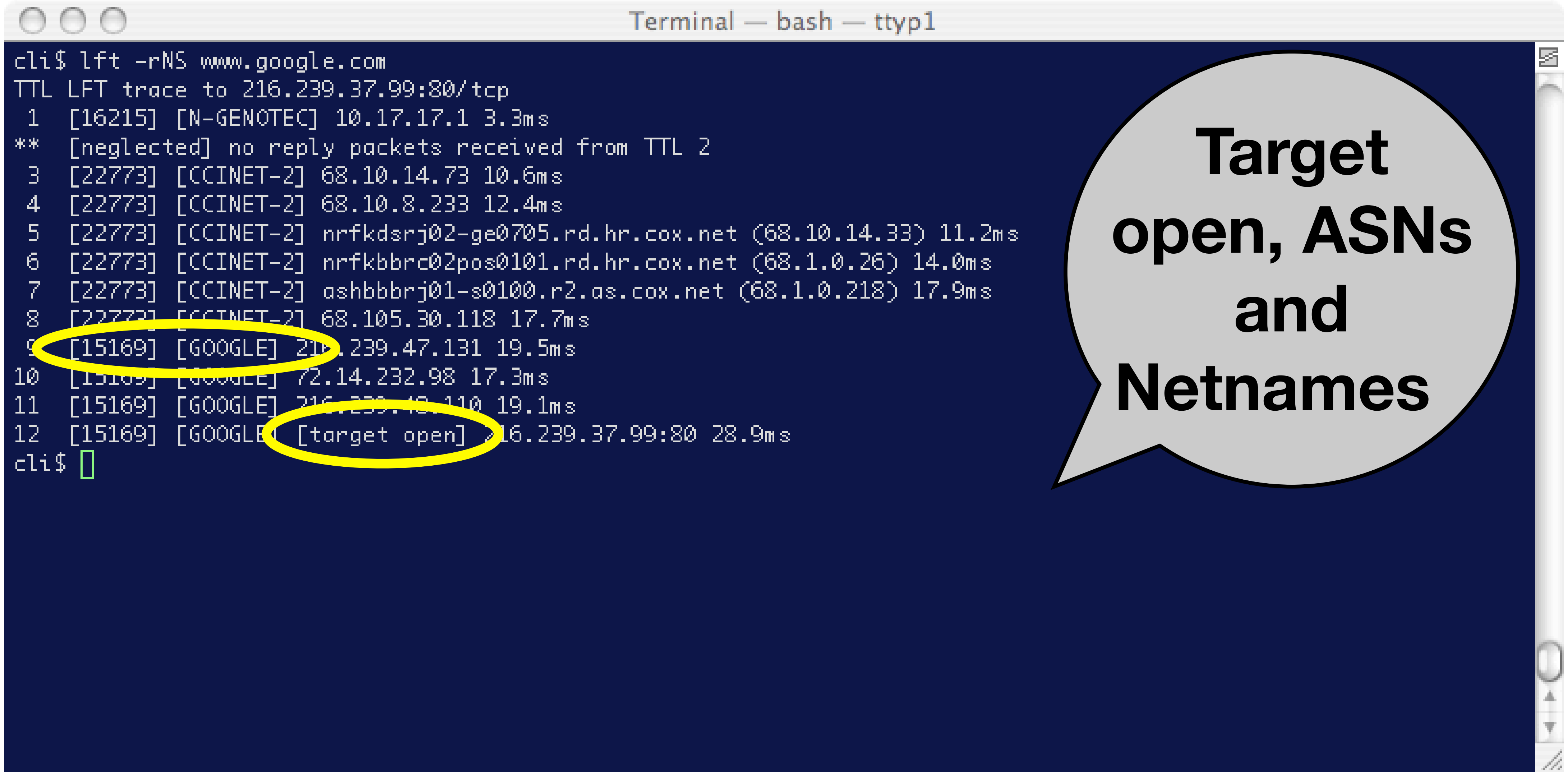


```
cli$ lft -EUS 4.2.2.1
LFT trace started at 06-Mar-06 22:19:45 GMT
TTL LFT trace to vnsc-pri.sys.gtei.net (4.2.2.1):80/tcp
 1  10.17.17.1 26.6/4.3ms
**  [neglected] no reply packets received from TTL 2
 3  68.10.14.69 14.2/11.4ms
 4  68.10.8.229 28.5/11.1ms
 5  nrfkdsrj02-ge0704.rd.hr.cox.net (68.10.14.29) 20.0/11.2ms
 6  68.10.15.1 34.7/10.8ms
 7  12.118.122.77 17.7/16.5ms
 8  tbr1-p010401.wswdc.ip.att.net (12.123.9.106) 17.7/18.2ms
 9  ggr2-p300.wswdc.ip.att.net (12.123.9.81) 17.3/17.9ms
10  so-9-0-0.car3.washington1.level3.net (209.244.219.141) 19.8/
**  [firewall] the next gateway may statefully inspect packets
11  ge-5-0-53.hsa1.washington2.level3.net (4.68.121.77) 18.8/17.7ms
12  [target] vnsc-pri.sys.gtei.net (4.2.2.1):80 16.6/17.4ms
LFT trace finished at 06-Mar-06 22:19:49 GMT (3.94s elapsed)
cli$ _
```

firewall detected

LFT times itself in UTC, shows elapsed time, and discovers a firewall in the path.

Example of LFT’s Advanced Resolution Capabilities



```
cli$ lft -rNS www.google.com
TTL LFT trace to 216.239.37.99:80/tcp
 1  [16215] [N-GENOTEC] 10.17.17.1 3.3ms
**  [neglected] no reply packets received from TTL 2
 3  [22773] [CCINET-2] 68.10.14.73 10.6ms
 4  [22773] [CCINET-2] 68.10.8.233 12.4ms
 5  [22773] [CCINET-2] nrfkdsrj02-ge0705.rd.hr.cox.net (68.10.14.33) 11.2ms
 6  [22773] [CCINET-2] nrfkbbrc02pos0101.rd.hr.cox.net (68.1.0.26) 14.0ms
 7  [22773] [CCINET-2] ashbbbrj01-s0100.r2.as.cox.net (68.1.0.218) 17.9ms
 8  [22773] [CCINET-2] 68.105.30.118 17.7ms
 9  [15169] [GOOGLE] 216.239.47.131 19.5ms
10  [15169] [GOOGLE] 72.14.232.98 17.3ms
11  [15169] [GOOGLE] 216.239.42.110 19.1ms
12  [15169] [GOOGLE] [target open] 216.239.37.99:80 28.9ms
cli$ 
```

Target open, ASNs and Netnames

LFT resolves both ASNs and Network names. LFT also indicates the port combination used resulted in the target attempting a 3WHS (3-way handshake).

VOSTROM

What's Prefix WhoIs and why is it necessary?

- We need to resolve the **origin-as** of a packet (and other helpful routing info). Quickly.

- We need to do it from **anywhere** without a route-view and without managing a telnet session to a route-view server/router

    - Doing it within a shell by issuing one command would be really nice

    - Doing it from PHP would be nice too

Why ASN Resolution Matters

- Security/Authenticity of global BGP prefix announcements is questionable. Prefix Hijacking is commonplace now with spamming, DoS, etc.
- Accounting, peering (justification), critical infrastructure
- Display of routing related information

Related Projects

Cymru's whois.cymru.com	provides AS info, some registrar information (ASname/ORGname), handles bulk nicely, quick, suitable output format
RIPE NCC's riswhois.ripe.net	sophisticated, data from many routing peers, RPSL-compliant, quick

VOSTROM

Why Another?

1. Open source so you can run your own with YOUR specific RIB and modify as needed

2. We noticed a few "differences" in the output of existing services (unreliable/assumptive RIB)

3. We needed something we could scale to handle large data sets for our research projects and shouldn't rely on public services

4. We wanted flexible/different output formats

Project Components

- “**pwhois-updatedb**” agent that periodically updates a relational database after parsing a RIB digest. Can retrieve the RIBs by:
  - Downloading RIB digests from routeviews.org
  - Building a RIB digest from your router through an automated telnet session
- “**pwhoisd**” server process that answers queries on port tcp/43.

Project Components

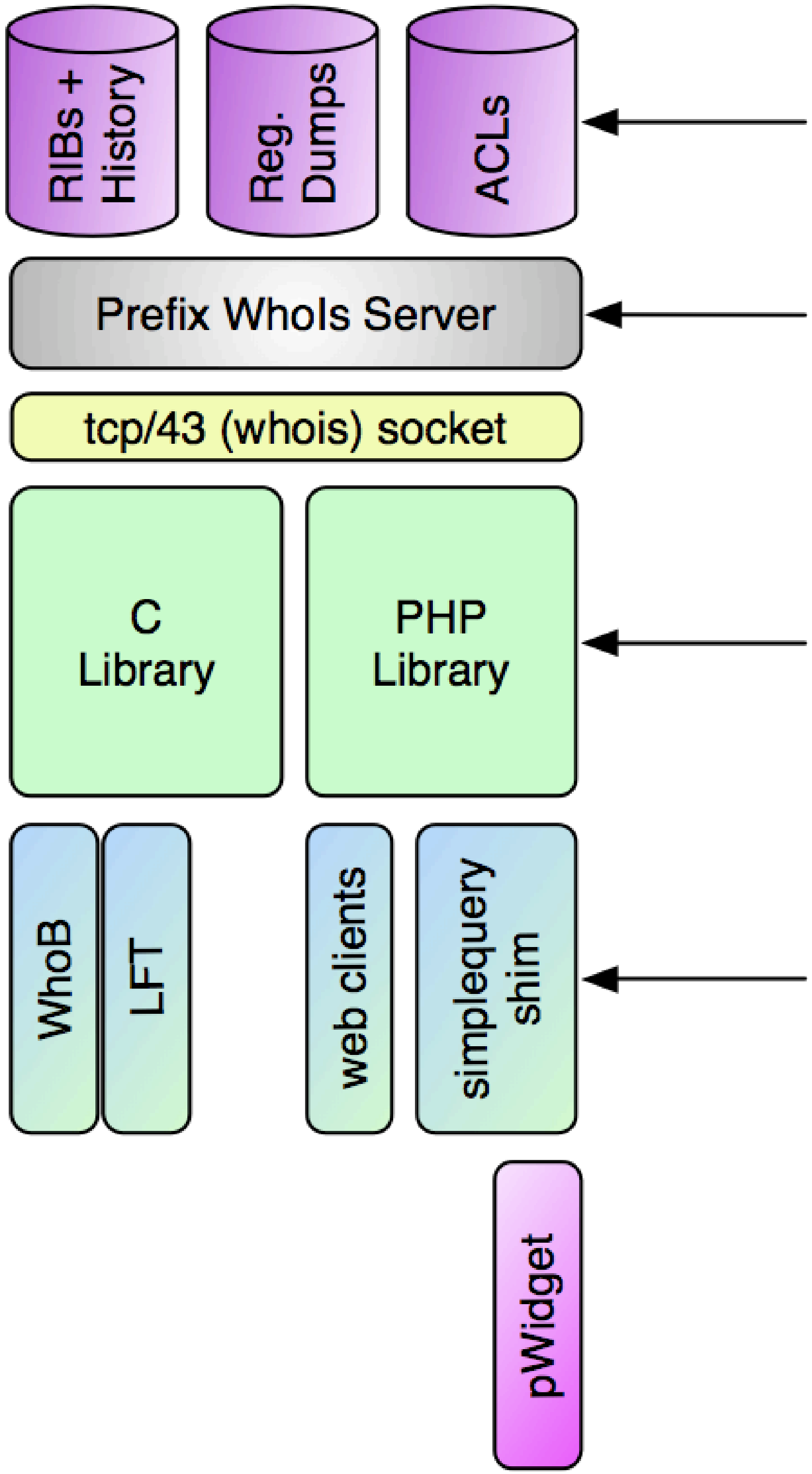
- “**pwhois C library**” *C-Language* programming library that provides a streamlined way of accessing whois-type information from pWhoIs and other whois servers.
- “**pwhois PHP library**” PHP-Language (5.x+) programming library that provides convenient access to pWhoIs using PHP-native socket interface (no more /usr/bin/whois calls)

Project Components

- "**whob**" simplified whois/pwhois client for network engineers. Less crap to parse, most of the content you care about.

- "**pWidget**" Apple Mac OS X dashboard client with access to pWhoIs through the simplequery HTTP interface. Just your basic eye candy.

Software Components



pwhois-updatedb
- maintenance

pwhoisd
- server

interface libraries
- language-dependent

clients
direct-interface clients

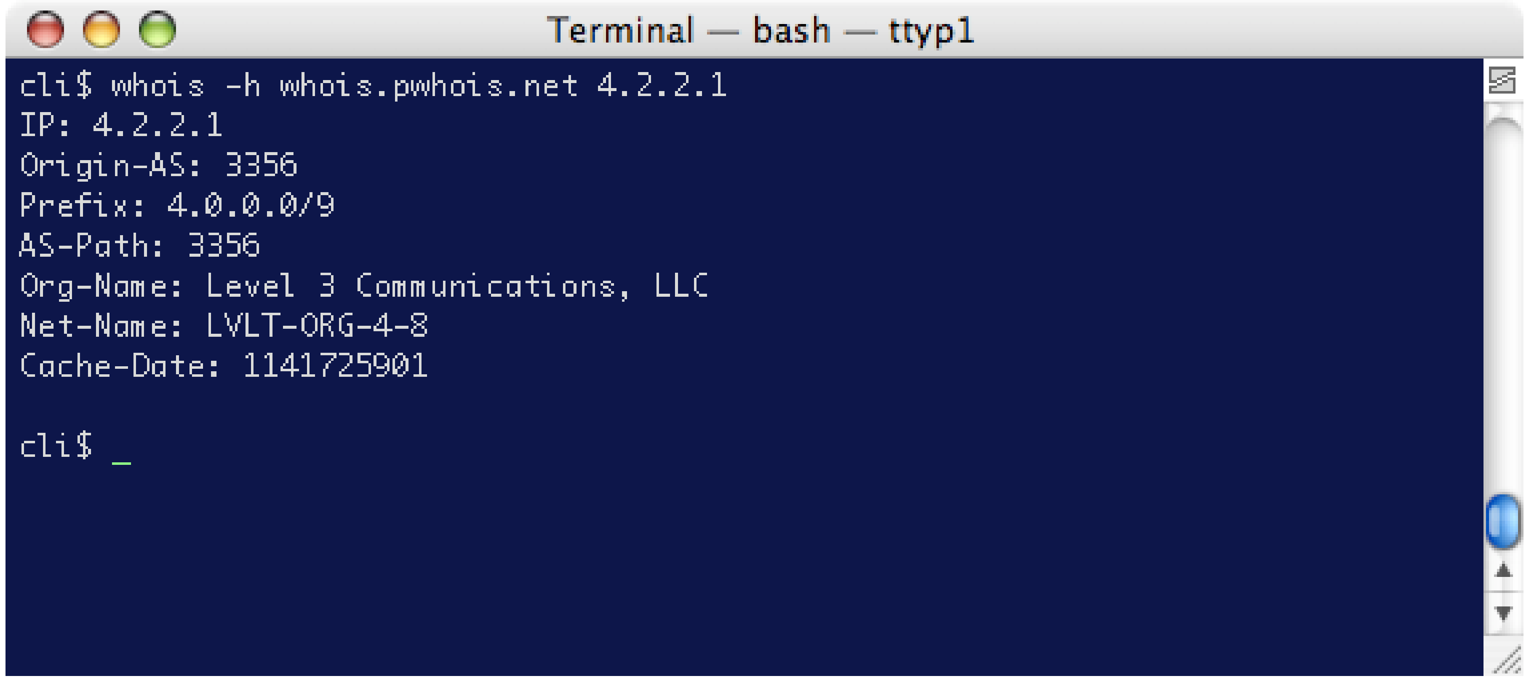
simple web clients
- interfacing through an
HTTP shim to pWhoIs

Usage Examples

Using whois clients to Interact with pWhoIs

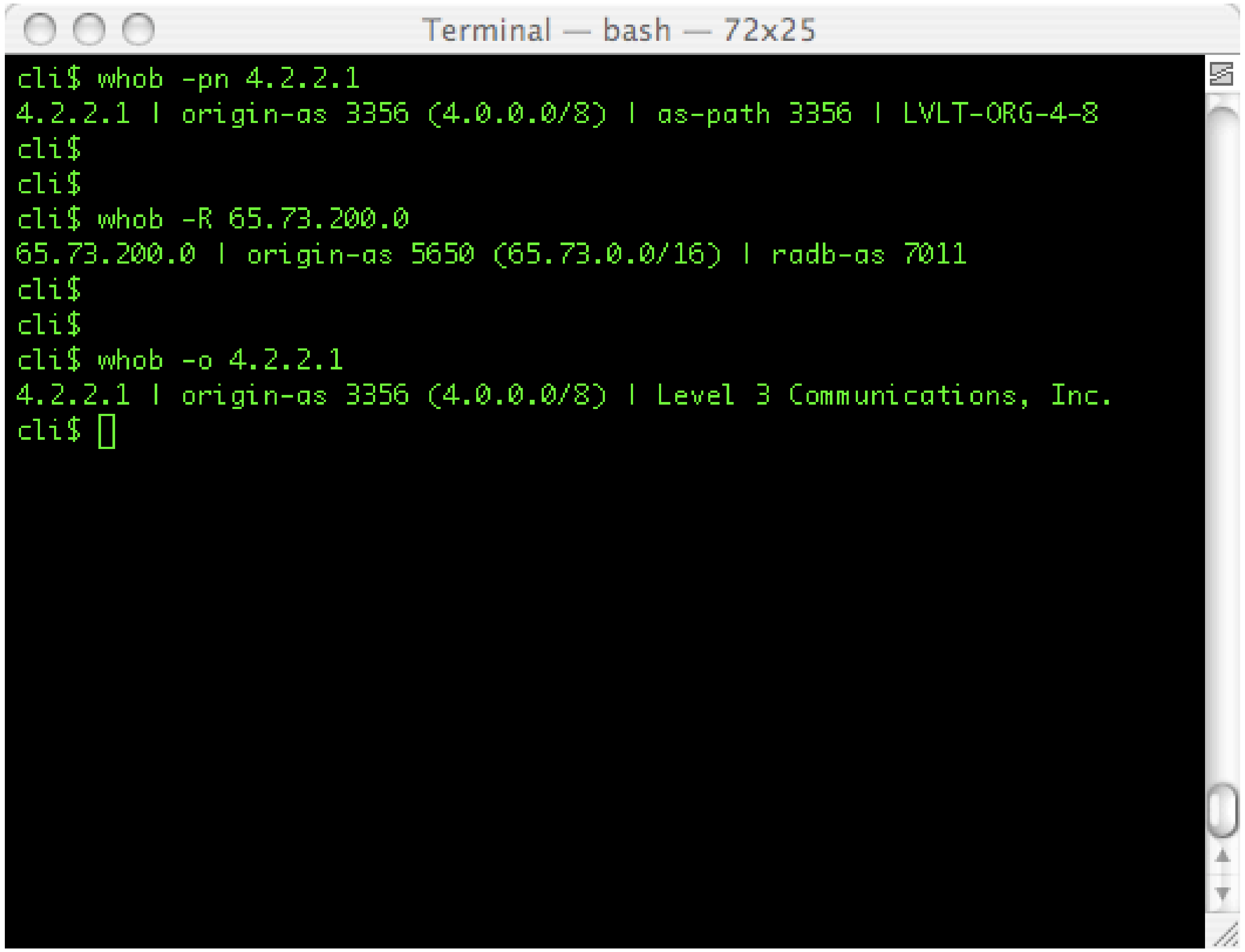
VOSTROM

Using a Standard whois client



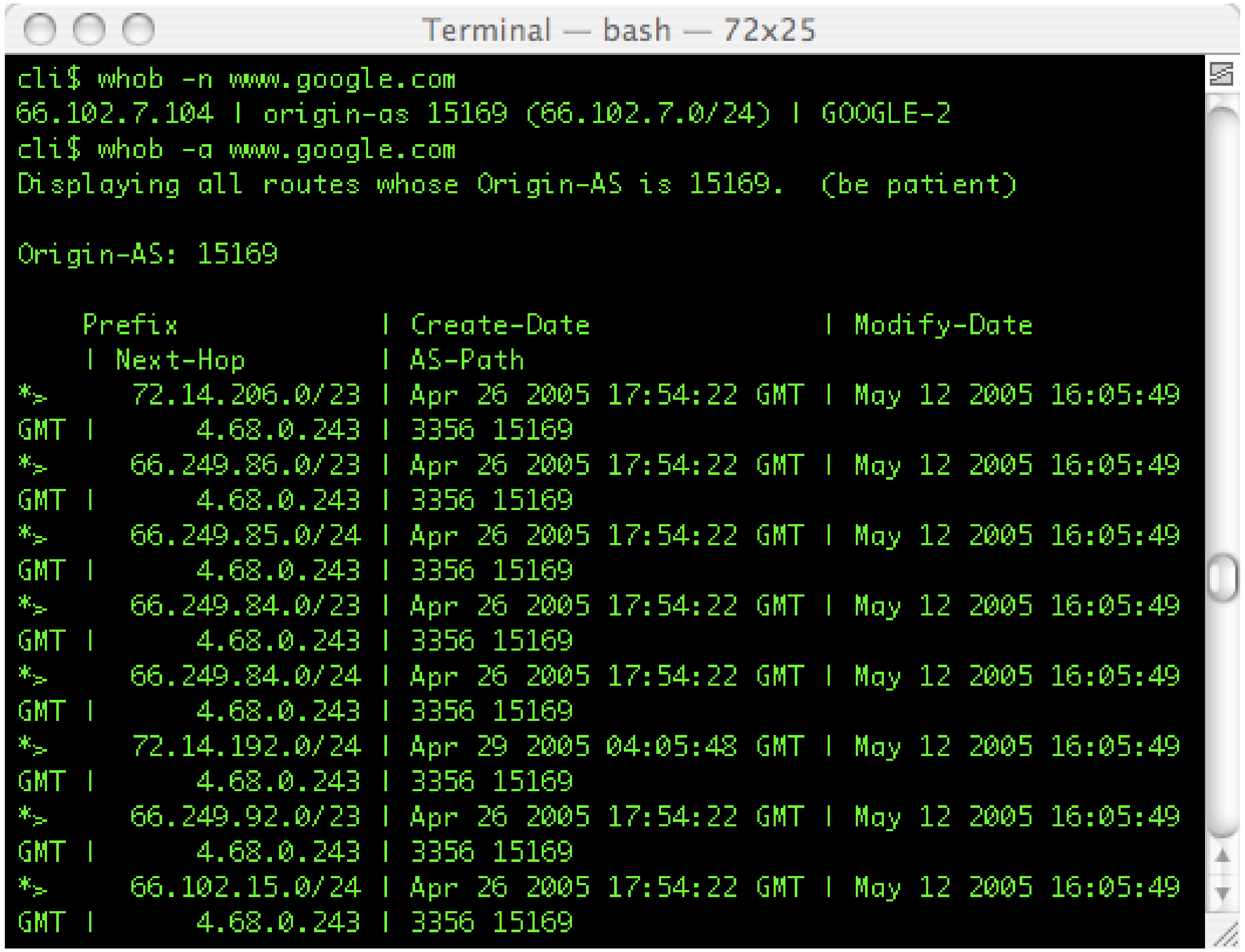
```
cli$ whois -h whois.pwhois.net 4.2.2.1
IP: 4.2.2.1
Origin-AS: 3356
Prefix: 4.0.0.0/9
AS-Path: 3356
Org-Name: Level 3 Communications, LLC
Net-Name: LVLT-ORG-4-8
Cache-Date: 1141725901

cli$ _
```

Some Unique WhoB Features



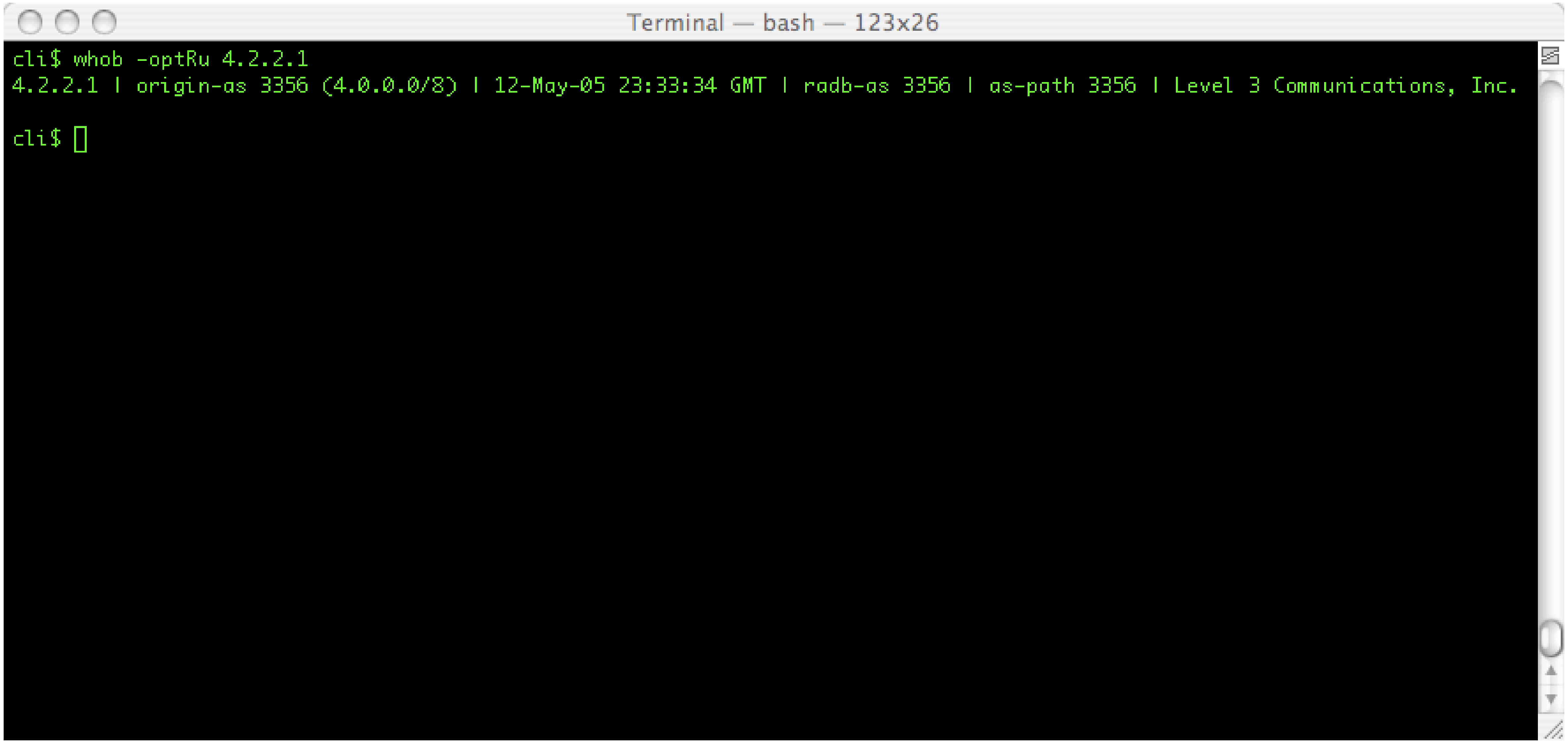
```
cli$ whob -pn 4.2.2.1
4.2.2.1 | origin-as 3356 (4.0.0.0/8) | as-path 3356 | LVLT-ORG-4-8
cli$
cli$
cli$ whob -R 65.73.200.0
65.73.200.0 | origin-as 5650 (65.73.0.0/16) | radb-as 7011
cli$
cli$
cli$ whob -o 4.2.2.1
4.2.2.1 | origin-as 3356 (4.0.0.0/8) | Level 3 Communications, Inc.
cli$ 
```

More Unique WhoB Features



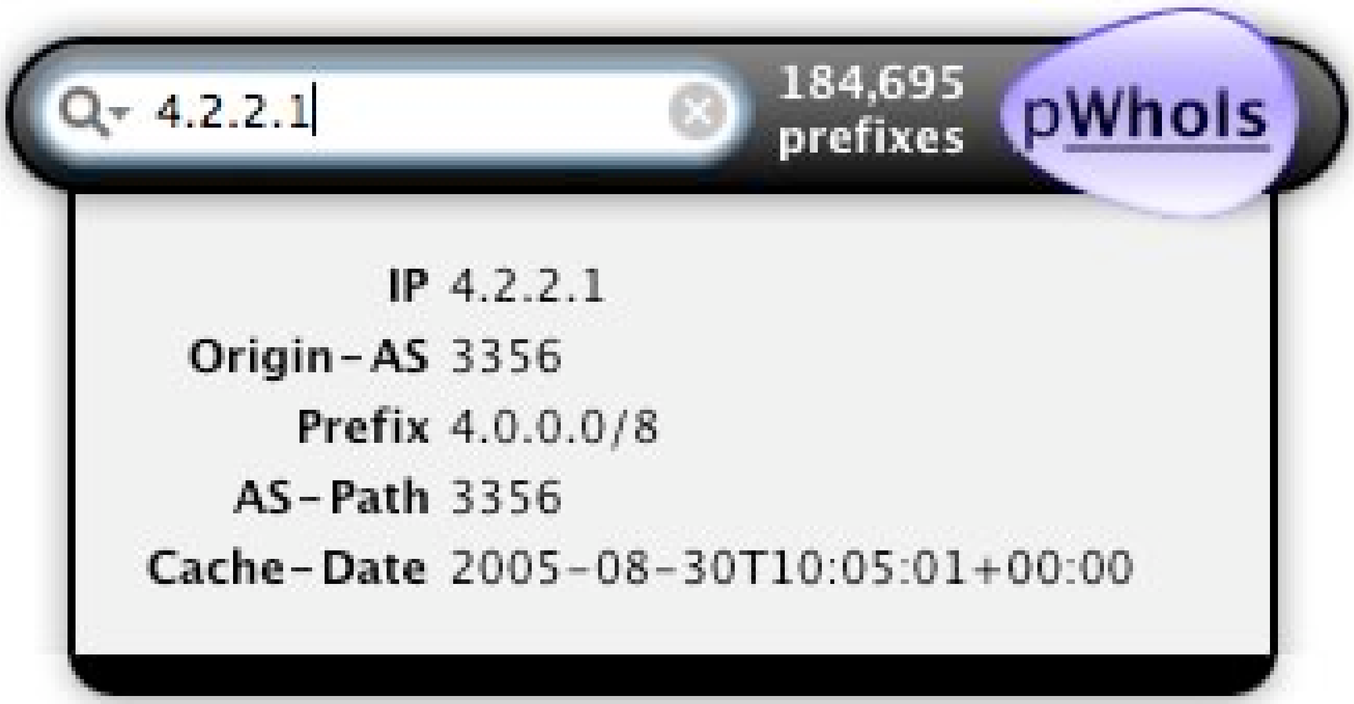
```
cli$ whob -n www.google.com
66.102.7.104 | origin-as 15169 (66.102.7.0/24) | GOOGLE-2
cli$ whob -a www.google.com
Displaying all routes whose Origin-AS is 15169.  (be patient)

Origin-AS: 15169

    Prefix            | Create-Date              | Modify-Date
    | Next-Hop        | AS-Path
*>     72.14.206.0/23 | Apr 26 2005 17:54:22 GMT | May 12 2005 16:05:49
GMT |      4.68.0.243 | 3356 15169
*>     66.249.86.0/23 | Apr 26 2005 17:54:22 GMT | May 12 2005 16:05:49
GMT |      4.68.0.243 | 3356 15169
*>     66.249.85.0/24 | Apr 26 2005 17:54:22 GMT | May 12 2005 16:05:49
GMT |      4.68.0.243 | 3356 15169
*>     66.249.84.0/23 | Apr 26 2005 17:54:22 GMT | May 12 2005 16:05:49
GMT |      4.68.0.243 | 3356 15169
*>     66.249.84.0/24 | Apr 26 2005 17:54:22 GMT | May 12 2005 16:05:49
GMT |      4.68.0.243 | 3356 15169
*>     72.14.192.0/24 | Apr 29 2005 04:05:48 GMT | May 12 2005 16:05:49
GMT |      4.68.0.243 | 3356 15169
*>     66.249.92.0/23 | Apr 26 2005 17:54:22 GMT | May 12 2005 16:05:49
GMT |      4.68.0.243 | 3356 15169
*>     66.102.15.0/24 | Apr 26 2005 17:54:22 GMT | May 12 2005 16:05:49
GMT |      4.68.0.243 | 3356 15169
```

It can get ugly...



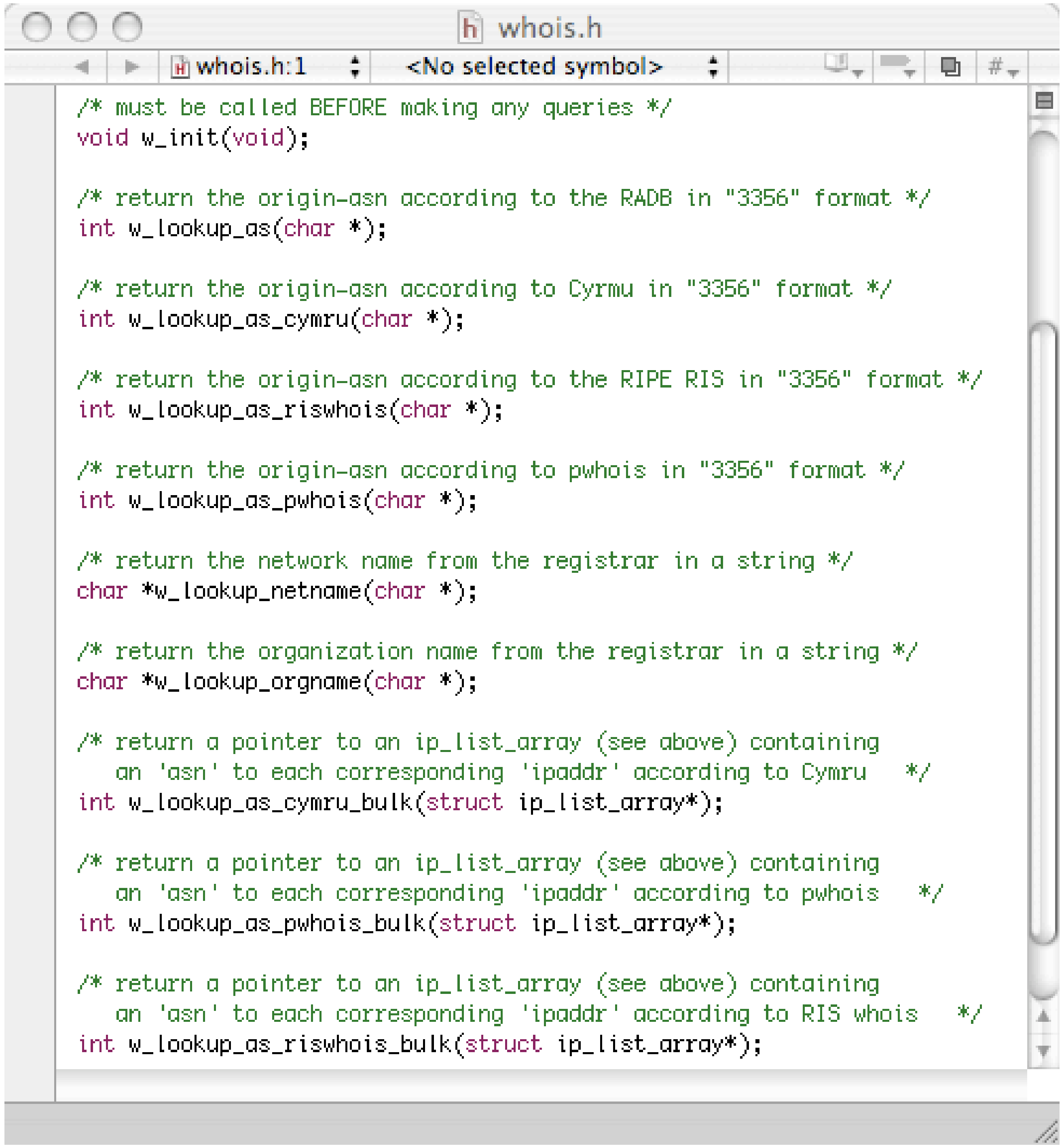
```
cli$ whob -optRu 4.2.2.1
4.2.2.1 | origin-as 3356 (4.0.0.0/8) | 12-May-05 23:33:34 GMT | radb-as 3356 | as-path 3356 | Level 3 Communications, Inc.

cli$ 
```

Prefix WhoIs, the Widget?



IP 4.2.2.1
Origin-AS 3356
Prefix 4.0.0.0/8
AS-Path 3356
Cache-Date 2005-08-30T10:05:01+00:00

Easy to access from a C program

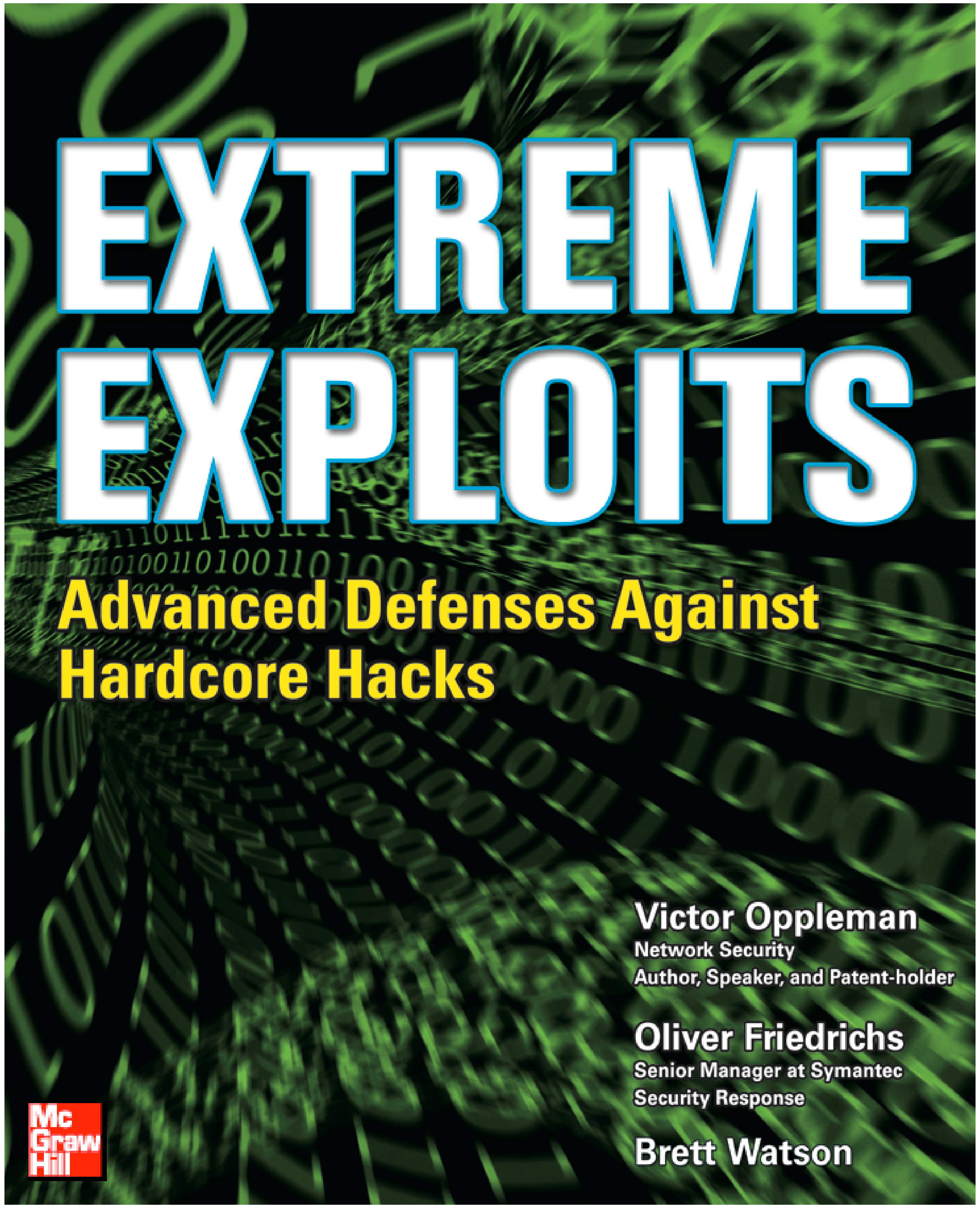


```c
/* must be called BEFORE making any queries */
void w_init(void);

/* return the origin-asn according to the RADB in "3356" format */
int w_lookup_as(char *);

/* return the origin-asn according to Cyrmu in "3356" format */
int w_lookup_as_cymru(char *);

/* return the origin-asn according to the RIPE RIS in "3356" format */
int w_lookup_as_riswhois(char *);

/* return the origin-asn according to pwhois in "3356" format */
int w_lookup_as_pwhois(char *);

/* return the network name from the registrar in a string */
char *w_lookup_netname(char *);

/* return the organization name from the registrar in a string */
char *w_lookup_orgname(char *);

/* return a pointer to an ip_list_array (see above) containing
   an 'asn' to each corresponding 'ipaddr' according to Cymru   */
int w_lookup_as_cymru_bulk(struct ip_list_array*);

/* return a pointer to an ip_list_array (see above) containing
   an 'asn' to each corresponding 'ipaddr' according to pwhois   */
int w_lookup_as_pwhois_bulk(struct ip_list_array*);

/* return a pointer to an ip_list_array (see above) containing
   an 'asn' to each corresponding 'ipaddr' according to RIS whois   */
int w_lookup_as_riswhois_bulk(struct ip_list_array*);
```

Easy to access from a PHP script

```php
/**
 *
 *  Prefix WhoIs Bulk Query Interface
 *
 *  -- a native interface to Prefix WhoIs implemented in PHP*
 *
 *                                        * requires PHP >= 5
 *
 *  Simply call doPWLookupASBulk(array $queryArray) and it will
 *  return an associative array of AS numbers indexed by the
 *  IP addresses passed to it in the $queryArray argument.
 *
 */

function doPWLookupASBulk($queryarray) {

$pwserver = 'whois.pwhois.org'; // Prefix WhoIs Server (public)
$pwport = 43;                   // Port to which Prefix WhoIs listens
$socket_timeout = 20;           // Timeout for socket connection operations
$socket_delay = 5;              // Timeout for socket read/write operations

------------ SNIP

  // An example of calling the function...

  $test_array = array('4.2.2.1','12.0.0.0');
  if (!($pwresp = doPWLookupASBulk($test_array))) {
          print "<h2>Your query wasn't answered.</h2>\n";
          exit();
  }

  foreach ($pwresp as $ip => $as) {
    print 'IP: '.$ip.', ASN: '.$as.'<br />';
  }
```

Please support our research by purchasing a book



EXTREME EXPLOITS

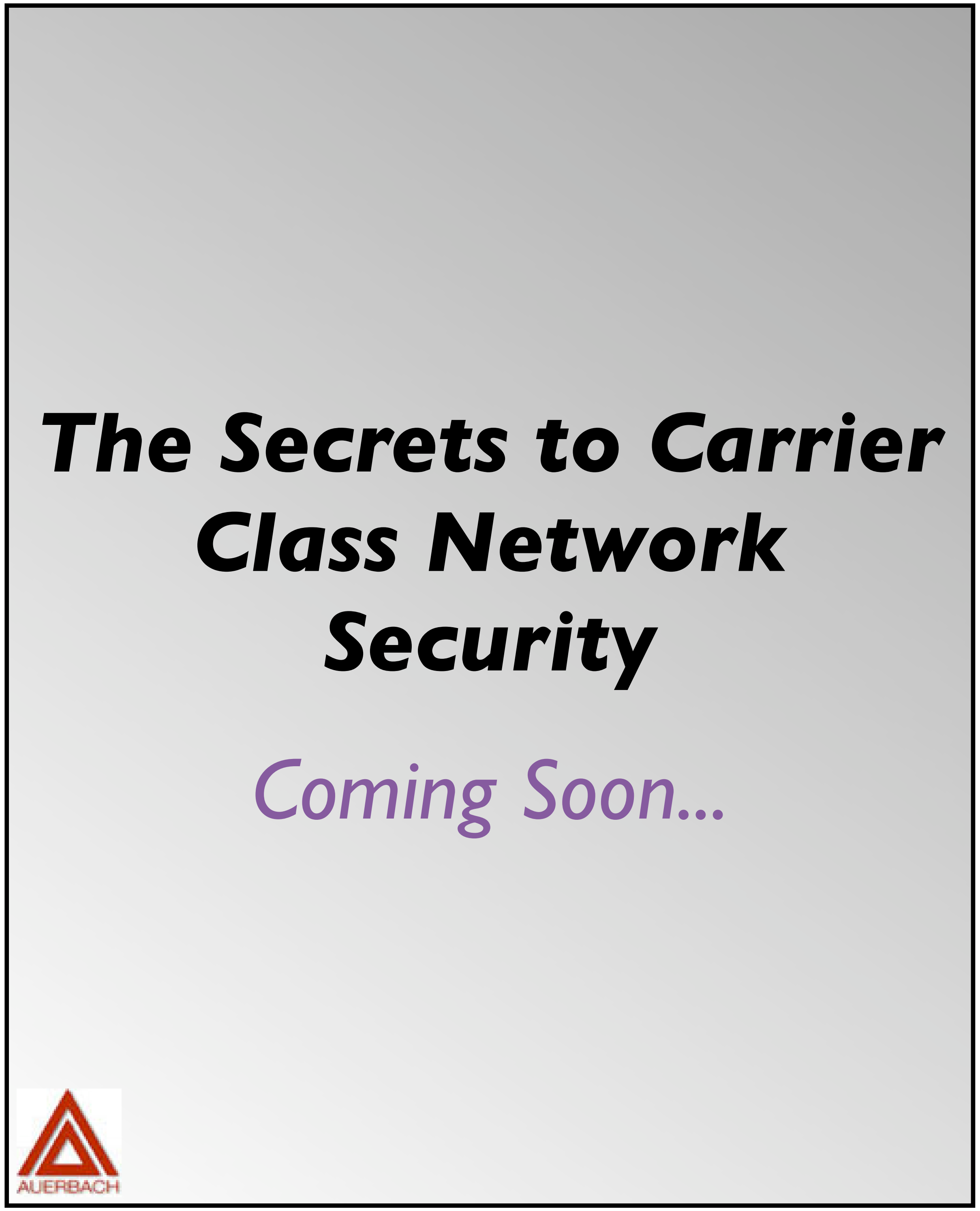
Advanced Defenses Against Hardcore Hacks

Victor Oppleman
Network Security
Author, Speaker, and Patent-holder

Oliver Friedrichs
Senior Manager at Symantec
Security Response

Brett Watson

Mc Graw Hill



The Secrets to Carrier Class Network Security

Coming Soon...

AUERBACH

USE IT	whois.pwhois.org
GET IT	http://pwhois.org
REACH OUT	victor @ pwhois.org