OpenBGPD and OpenNTPD

Henning Brauer <henning@openbsd.org>

BGP - The Protocol

- Border Gateway Protocol, RFC 1771
- ISPs talk BGP to each other to announce reachability of their networks
- Networks are subsummarized into Autonomous Systems (AS)
- One ISP is typically one AS

BGP - The Protocol

- Network reachability is announced with so-called AS-Pathes, describing the path to the final network through intermediate ASes
- A BGP speaker usually announces directly connected networks, and prefixes with their pathes it learned from its neighbors
- An AS Path looks like "13237 174 3602 22512", listing the AS numbers we cross on the way to the destination, in this case, cvs.openbsd.org

BGP - Messages

- OPEN
  - Sent once at establishment of the tcp session. contains parameters such as the AS number.
- KEEPALIVE
  - Sent periodically to test wether the session is still alive.
- UPDATE
  - These messages carry the actual routing information.
- NOTIFICATION
  - Sent on fatal errors. After sending a notification the session is reset.

BGP - Existing Implementations

- Zebra: GPL, makes heavy use of cooperative threads. Suffers from losing sessions while busy. Documentation and error messages in japanese or missing. Commercialized, thus mostly dead since about 2 years.

- Quagga: frustrated zebra users try to fix the worst bugs

- gated: became unfree, then died. Nothing really usable left.

BGP - Existing Implementations

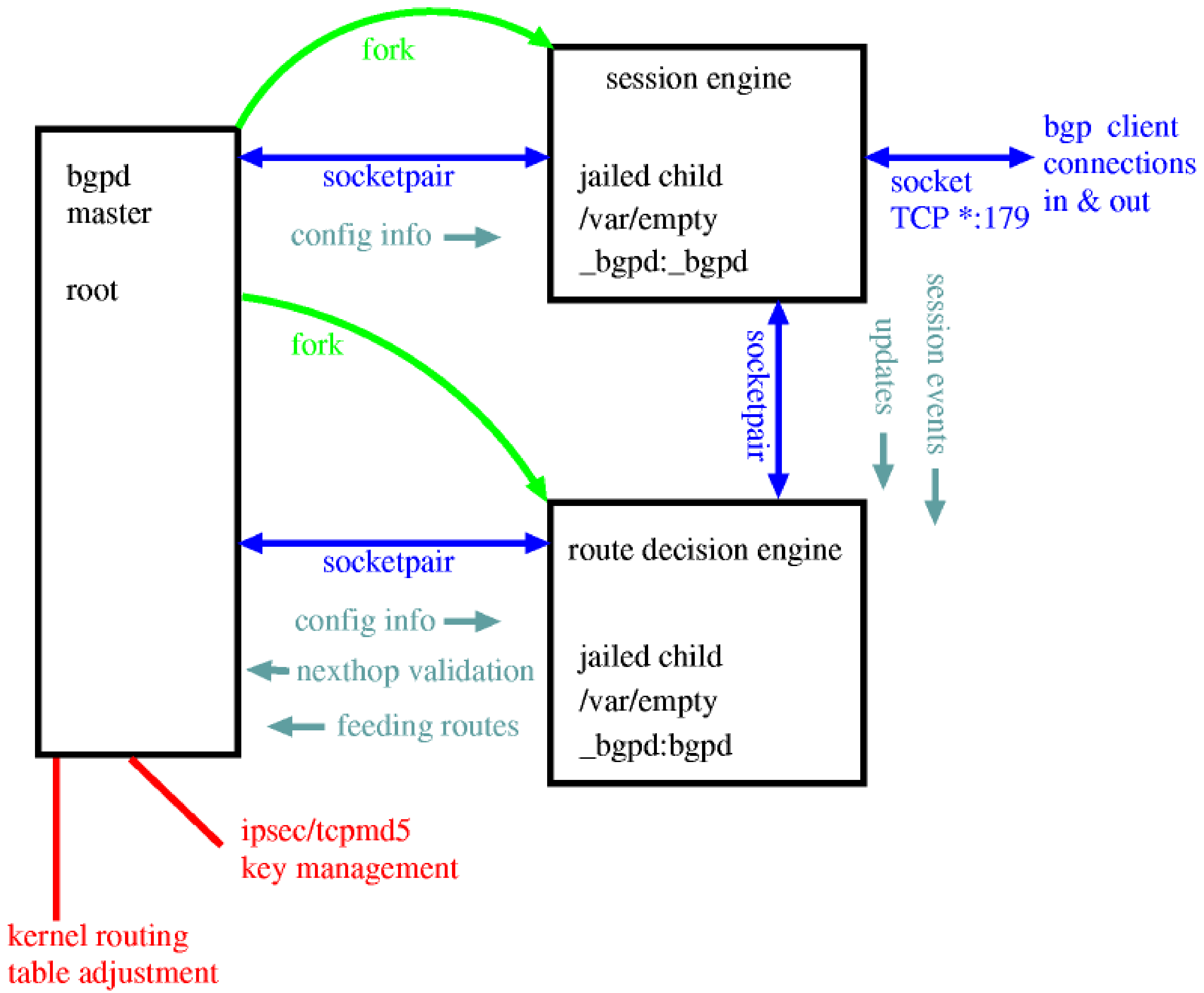
- Cisco: proprietary, only works on their overpriced routers. Usually works ok, unless you happen to hit one of its countless bugs, or the tiny CPUs they use are swamped with work.
- Juniper's JunOS: apparently works ok, but not free either.

bgpd - Design Prerequisites

- Security. Code careful, use bounded buffer operations, and account for own failure by using privilege separation.
- Don't lose sessions. There should be a fairly independent session engine.
- Performance and memory efficiency, of course.
- Well designed config and filter language.

bgpd - Design

- 3 processes
  - Session Engine (SE): manages bgp sessions
  - Route Decision Engine (RDE): holds the bgp tables, takes routing decisions
  - Parent: enters routes into the kernel, starts SE and RDE



fork
session engine
bgp client
connections
in & out
bgpd
master
socketpair
jailed child
/var/empty
_bgpd:_bgpd
socket
TCP *:179
config info
root
fork
socketpair
updates
session events
socketpair
route decision engine
config info
nexthop validation
feeding routes
jailed child
/var/empty
_bgpd:bgpd
ipsec/tcpmd5
key management
kernel routing
table adjustment

bgpd - Design

- Obviously, the Session Engine needs to be nonblocking, and use nonblocking sockets.
  - We need to handle all buffering ourselves.
- Invent an easy to use Buffer API
- For the internal messaging, invent an "imsg" API as well.
  - internal messaging is a core component in privilege separation
  - 44 message types now

bgpd - Session Engine

- Maintains a listening tcp socket
- Opens tcp connections to neighbors
- Negotiates parameters with neighbors via OPEN messages
- Once a session is established, it sends KEEPALIVE messages regularly, and receives ones from the neighbors

bgpd - Session Engine

- Finite State Machine for each neighbor
- UPDATEs received from a neighbor are passed to the RDE.
- Outgoing UPDATEs are generated in the RDE and the SE just relays them.

bgpd - Session Engine

- Maintains a Unix-Domain socket for the bgpctl program
- very lightweight: typically under 1 MB RAM on i386
- runs as unprivileged user _bgpd, chroots to /var/empty

bgpd - Route Decision Engine

- Maintains the Routing Information Base (RIB)
  - prefix table
  - AS path table
- BGP Filters run here
- Calculates the best path per prefix
- Generates UPDATE messages as needed

bgpd - Route Decision Engine

- RIB Layout
  - Split into many tables
  - Heavily linked
  - Avoid table walks
- UPDATE messages are processed to completion
- Generated UPDATEs are queued to use piggy-back optimization
- RIB Table and sessions can be dumped to mrt files

bgpd - Route Decision Engine

- Memory efficent
  - 1 full view needs around 20 MB
  - 2 full views need around 25 MB
- Fast
  - Around 10s to load a full view on a PIII 1GHz
  - Less than 5s to dump a full view to another router
- Runs as unprivileged user _bgpd, chroots to /var/empty

bgpd - Parent process, kernel interface

- Responsible for getting the routes into the kernel
- Does nexthop validation for the RDE
- Maintains its own copy of the kernel routing table
- Fetches the kernel routing table and interface list on startup

bgpd - Parent process, kernel interface

- Listens to the routing socket
  - Internal view of the kernel routing table is held in sync
    - If you fiddle with the routing table manually, we notice that and cope with it
  - Internal list of interfaces and their status is kept in sync
    - We know about interfaces' link status and use it for nexthop verification
    - Yes, we notice when you pull the cable!
- We don't need periodic nexthop table walks

bgpd - Parent process, kernel interface

- The internal view of the routing table can be coupled and decoupled from the kernel
  - Damn fast! With a full table (about 150000 entries), less than 3 seconds on a PIII 750.
- Needs about 5 MB in full-mesh configurations

bgpd - tcp md5 signatures

- bgp sessions are not really authenticated - just IP based access control
- An attacker could send a bgp notification message with a faked source address, resetting the connection -> DoS

bgpd - tcp md5 signatures

- RFC 2385 defines tcp md5 signatures
- An md5 hash of parts of the header and a shared secret is added to the tcp header and verified on the receiving side
  - (unless you happen to run FreeBSD, they don't bother verifying the signatures)
- Attacker has to know the shared secret

bgpd - tcp md5 signatures

- Very old code for tcp md5 signatures existed, but didn't work. We used it as starting point.
- We implemented tcp md5 signatures as Security Association within the IPsec framework
- bgpd got a pfkey interface to interact with the IPsec framework
- tcp md5sig is extremly easy to configure, works with ciscos and junipers, too: USE IT!

bgpd - tcp md5 signatures

- Keep in mind that tcp md5 sigs are rather weak
- Take care for the key length - use at least 12 bytes
- Make sure to read RFC 3562, "Key Management Considerations for the TCP MD5 Signature Option"

bgpd - ipsec integration

- As we had the pfkey interface already, it was not too hard to do real IPsec
  - bgpd loads the SAs into the kernel
  - bgpd sets up the flows
- Juniper can do static-keyed IPsec as well, we're compatible.
- Cisco cannot, of course
  - (could cause CPU load after all!)

bgpd - ipsec integration

- We can use isakmpd to do the keying for us
  - keys are changed on a regular basis
- bgpd asks the kernel for an unused pair of SPIs and uses them
- bgpd sets up the flows
  - it knows the endpoints and ports already
- isakmpd only needs to handle the keying
  - almost NO configuration needed!
  - copy key files (generated at first boot on OpenBSD 3.6) over
  - run "isakmpd -Ka"

bgpd - pf integration

- The BGP protocol is an efficient way to distribute lists of network prefixes, so we integrated bgpd with our pf packet filter
- bgpd can add prefixes learned from neighbors into a pf table
  - prefixes are selected using the bgpd filter language
  - tables use a radix tree, very fast even with lots of entries
- pf tables can be used for pretty much anything:
  - packet filtering
  - redirection to spamd (BGP distributed spam blacklists)
  - QoS processing

bgpd - carp integration

- The Common Address Redundancy Protocol allows two hosts to share an IP address in a master-backup scenario
  - kinda VRRP unencumbered, but better
- Typical case: Exchange Points. You get one IP in the IX-network.
  - What about using two machines and CARP
    - works without special support from bgpd, but we can do better

bgpd - carp integration

- Make bgpd aware of the CARP master/backup state
  - this is actually the link state for the carp interface
- For sessions depending on the carp interface, keep them in state IDLE as long as the carp interface is not master.
- The very same moment the carp interface gets master, all sessions depending on it go to Connect (or Active for passive sessions)
  - much faster failover

bgpd - configuration

- Split into 5 sections
  - Macro definitions - just like in pf
  - Global settings
  - Networks to announce
  - Neighbor definitions
  - Filter

bgpd - macros, global config, networks

```
#macros
peer1="10.0.0.2"
peer2="10.0.0.3"
myip="127.0.0.1"

# global configuration
AS 65001
router-id $myip
listen on $myip
holdtime 180
holdtime min 3
fib-update no

# networks we announce
network 10/8
network 192.168.2/23
```

bgpd - neighbor definition

```
neighbor 10.0.1.0 {
        remote-as         65003
        descr             upstream
        multihop          2
        local-address     10.0.0.8
        passive
        holdtime          180
        holdtime min      3
        announce          self
        tcp md5sig key    deadbeef
}
```

- Very cool: the annnounce keyword
  - none: don't announce any networks
  - self: announce only our own networks
  - all: announce everything we know
  - default-route: announce a default-route and nothing else
- On cisco/zebra you need filters for this

bgpd - neighbor groups

```
group "peering AS65002" {
        remote-as       65002
        passive
        holdtime        180
        holdtime min    3

        neighbor $peer1 {
                descr    "AS 65001 peer 1"
                announce self
                tcp md5sig password mekmitasdigoat
        }
        neighbor $peer2 {
                descr    "AS 65001 peer 2"
                announce all
        }
}
```

bgpd - ipsec configuration, static keying

```
neighbor 10.2.1.1 {
        remote-as 65023
        local-address 10.0.0.8
        ipsec esp in  spi 10 \
                sha1 0a4f1d1f1a1c4f3c9e2f6f0f2a8e9c8c5a1b0b3b \
                aes 0c1b3a6c7d7a8d2e0e7b4f3d5e8e6c1e
        ipsec esp out spi 12 \
                sha1 0e9c8f6a8e2c7d3a0b5d0d0f0a3c5c1d2b8e0f8b \
                aes 4e0f2f1b5c4e3c0d0e2f2d3b8c5c8f0b
}
```

bgpd - ipsec configuration, using IKE

```
neighbor 10.2.1.1 {
        remote-as 65023
        local-address 10.0.0.8
        ipsec esp ike
}

neighbor 10.2.1.2 {
        remote-as 65024
        local-address 10.0.0.8
        ipsec ah ike
}
```

filter language

```
# filter out prefixes longer than 24 or shorter than 8 bits
deny from any
allow from any prefixlen 8 - 24

# do not accept a default route
deny from any prefix 0.0.0.0/0

# filter bogus networks
deny from any prefix 10.0.0.0/8 prefixlen >= 8
deny from any prefix 172.16.0.0/12 prefixlen >= 12
deny from any prefix { 192.168.0.0/16 169.254.0.0/16 } \
        prefixlen >= 16
deny from any prefix 192.0.2.0/24 prefixlen >= 24
deny from any prefix { 224.0.0.0/4 240.0.0.0/4 } prefixlen >= 4
```

bgpctl

- Client connecting to bgpd via unix domain socket
  - query runtime information
  - reload configuration
  - (de-)couple kernel routing table
  - take specific sessions up/down

bgpctl

```
<henning@cr11>  $ bgpctl show summary
Neighbor        AS    MsgRcvd    MsgSent    OutQ  Up/Down   State/PrefixRcvd
carrier66       24953     118199     115193      0 01:14:05      17/50
christiansen    34181     114091     114064      0 23:14:16       1/50
otto            16378     178676     178657      0 01w1d11h       1/5
inetbone        25074     187600     178679      0 07w4d14h     167/200
Headlight        6666     178743     178665      0 07w5d11h      15/30
ISC              8805     157643     157616      0 07w4d20h       1/5
Artfiles         8893     192658     177125      0 07w5d11h       6/20
TNG             13101     179100     178670      0 07w5d11h      14/70
wizard.de       12923     178179     178180      0 4d01h06m       1/26
MCS Cityline     5521     178620     178601      0 06w5d11h       6/20
smartnet        12485          0          0      0 Never     Active
lynet           12822     178664     178662      0 01w3d05h       2/5
OMCnet          15388     178661     178665      0 07w4d20h       1/5
freenet          5430     178901     178670      0 02w3d21h      18/50
crew-kg         13135     179963     179793      0 04w5d05h       6/20
shlink.de       12518     178667     178661      0 07w5d11h       6/20
ppp.net          8687     175794     175716      0 07w5d11h      10/20
n@work           9211     178727     178615      0 04w5d21h      11/25
cogent          13129    6092047     178616      0 03w6d09h 151910
lambdanet       13237   23202463     178669      0 08w6d00h 152857
cr31            64514     178624     178669      0 08w6d00h       0
```

bgpd - status quo

- Very stable
- In use at quite some sites, including setups with many many many many many many many many many peers.
  - Quite some operators mail me, expressing that they are very happy with bgpd's performance, reliability and ease of use
    - That makes me happy ;)
- Some statistics...
  - bgpd: 17744 lines of code
  - bgpctl: 1384 lines of code
  - manpages: 2611 lines

bgpd - evil future plans

- Give pf access to some more information from bgpd
- allow for freetext labels attached to a route
  - 32 bytes we can use to attach arbitary information
  - implemented in route(8) and the kernel routing table, as well as in pf.
  - bgpd can't set it - will be there soonish...
- This is really evil:

```
pass in from route DTAG queue reallyslow keep state
```

OpenNTPD - Design Goals

- security
  - very tight validity checks in the network input path
  - all buffer operations bounded and/or properly guarded
  - privilege separation
- ease of use
  - lean implementation, sufficient for a majority
    - no overloaded feature monster
  - should "just work" in the background
  - should reach reasonable accuracy
    - we're not after the last microseconds
  - should only require a minimum of configuration
- performance, of course!

NTP - The Protocol

- Some much too chatty RFC about it (1305)
- The protocol itself is dead simple
- The math to do is harder - but it turned out the RFC describes an overdone implementation, accounting for an accuracy you'll never see on a typical Unix system's clock
  - suprised, anyone?
- far more than 100 pages...

NTP - The Protocol

- On-the-wire format is really dead simple.
- 64 bit timestamps: 32 bit integer part, 32 bit fraction
- 32 bit timestamps (16 bit int, 16 bit fraction) for informational stuff

The Protocol

```
struct ntp_msg {
    u_int8_t            status;         /* incl. leap info */
    u_int8_t            stratum;
    u_int8_t            ppoll;
    int8_t              precision;
    struct s_fixedpt    rootdelay;
    struct s_fixedpt    dispersion;
    u_int32_t           refid;
    struct l_fixedpt    reftime;
    struct l_fixedpt    orgtime;
    struct l_fixedpt    rectime;
    struct l_fixedpt    xmttime;
    u_int32_t           keyid;
    u_int8_t            digest[NTP_DIGESTSIZE];
};
```

The Protocol: Timestamps

- 4 really important ones

Timestamp Name	ID	When Generated
Originate Timestamp	T1	time request sent by client
Receive Timestamp	T2	time request received by server
Transmit Timestamp	T3	time reply sent by server
Destination Timestamp	T4	time reply received by client

- Local clock offset is now easy to calculate

$$t = ((T2 - T1) + (T3 - T4)) / 2$$

Implementation: Privilege Separation

- two processes
  - parent, runs as root
  - ntp engine, runs as _ntp:_ntp and chroots to /var/empty
- socketpair in between
  - use the buffer- and imsg-framework I wrote for bgpd
- three message types: IMSG_ADJTIME, IMSG_SETTIME, and IMSG_HOST_DNS

Implementation: Privilege Separation

- ntpd is a very good example for privilege separation
- it is simple enough to be understood easily
- the message types show the two common reasons we need to privilege separate for (instead of just dropping privileges)

Implementation: Privilege Separation

- IMSG_ADJTIME: ntp engine asks the parent to do the adjtime() call
  - requires root
- same IMSG_SETTIME, calls settime()
- IMSG_HOST_DNS: ntp engine asks the parent to resolve hostnames
  - requires access to /etc/resolv.conf, YP maps, and whatnot

Implementation: Privilege Separation

- very important: very very very strict validity checks upon receival of the messages - the unprivileged client is untrusted

- if something is wrong with a message from the unprivilged process, fail immediately and hard - exit, without ever talking to the client again

Implementation: Server side

- very easy
  - recvfrom(2)
  - decode request
  - gettimeofday(2)
  - build reply
  - sendmsg(2)
- oups, not that easy... we might reply with the wrong src address
  - many implementations will refuse our answer
  - listen on each individual IP, so we know which IP the request was sent to and can use that as src address when replying
  - use getifaddrs(3) to get the IPs
    - not available on Solaris, so there people have to specify the addresses to listen on manually
      - until Sun gets a clue at least

Implementation: Client side

- bit harder
  - send queries to all peers
  - little state engine so we don't wait forever for replies
  - on receival of the replies, calculate offsets and such
  - collapse the offsets learned from each peer into a single offset and call adjtime()
- Unfortunately, it's a little more complicated...

Implementation: Client side

- to increase accuracy, we need to filter the replies we get
  - "clock filter", implementing an algorithm by David Mills
  - basically, from 8 replies received from a peer, use the one with the lowest delay, and invalidate all older replies
- bad network connection results in poor accuracy
  - punish peers with bad network connection - currently only based on packet loss
  - once punished, a peer needs to get a number of replies to us that we consider good before the peer is marked valid again and affects the total offset calculation

Implementation: Client side

- in the query, we set the "transmit timestamp" to a random 64-bit cookie, and store both our cookie and the real transmit timestamp locally
- servers are required to copy that timestamp verbatim into the "originate timestamp" in the reply
- upon receival of the reply, we check that the originate timestamp matches the cookie
- It is a really cool hack, extending NTP security without any drawbacks

Implementation: Falsetickers

- What if some server deliberately sends us wrong time?
- there is an incredibly complicated falsetickers detection in the ntp.org implementation
- it can of course only work with a reasonable big set of servers
  - if you only query 2, no way to detect a falseticker

Implementation: Falsetickers

- we can filter away falstickers much simpler
- after the clock filter we have one reply per server
- to get the local clock offset, we take the median offset from all replies - not the average

Implementation: Falsetickers

- Lets look at median.
- basically, you order all offsets by value, and take the middle one.

```
        12
        14
        1024

average: (12 + 14 + 1024) / 3 = ~350
median: 14
```

Implementation: cope with big offsets at startup

- If the local clock is waaaaayyy off at startup, adjtime() will need ages to cope with that
- usually this is coped with by running something before ntpd startup that sets the clock hard at boot
- OpenNTPD 3.6.1 can do that itself. No second thing to configure.
  - -s command line switch for that, added unconditionally by our rc scripts
  - -S to override -s

Getting started, howto style

- sync your OpenBSD 3.6 box's clock to a set of random public timeservers

```
# echo 'ntpd_flags=""' >> /etc/rc.conf.local

# reboot
    -- or --
# ntpd
```

- that's it.

Configuration

```
# $OpenBSD: ntpd.conf,v 1.7 2004/07/20 17:38:35 henning Exp $
# sample ntpd configuration file, see ntpd.conf(5)

# Addresses to listen on (ntpd does not listen by default)
#listen on *

# sync to a single server
#server ntp.example.org

# use a random selection of 8 public stratum 2 servers
# see http://twiki.ntp.org/bin/view/Servers/NTPPoolServers
servers pool.ntp.org
```

Configuration

- listen on: tell ntpd to listen on a specific IP or all IPs
  - listen on *
  - listen on 127.0.0.1
  - can occur multiple times
- server: sync to a single server
  - if given as hostname that resolves to more than one IP, use the first one. If we don't get a reply from that, pick the next one and retry
- servers: sync to a set of servers (pool.ntp.org)
  - if given as hostname that resolves to n IPs, treat as if n "server $ip" statements were given

status quo

- 3000 lines of code, with only a tiny fraction running as root
- accuracy typically around 50ms
  - good enough for most uses - this is the system clock's accuracy limiting us...
- performance is very good
- everybody loves how easy to use it is ;)

future ideas and ongoing work

- permanent tick frequency adjustment
  - needs kernel support
- better filtering
  - detect outliers and punish peers omitting those
- maybe support GPS clocks and such

Thanks

- Claudio Jeker <claudio@openbsd.org> and Andre Oppermann <andre@freebsd.org> for working on bgpd with me
- Alexander Guy who worked on ntpd with me in the early days
- Theo de Raadt for kicking my lazy butt, lots of design help and many many many McNally's we had while discussing bgpd and ntpd
- Wim Vandeputte, for his continued support and beer supply
  - (don't ask him about the hotel minibar please)

The unavoidable last page, 2004 edition

- We have cool shirts and posters for sale outside, as well as OpenBSD CDs
- Money is running out, donations can be made at http://www.openbsd.org/donations.html or outside at our booth
- Beer donations for the hackers are always welcome!