OPERA software

# NTP, a misunderstood protocol

designing an efficient NTP subnet: the Opera case

# Who is this guy?

- Marco Marongiu
- Currently working as Senior System Administrator for Opera Software in the Company Headquarters, located in Oslo;
- Working steadily as a System Administrator since 1999:
  - Sardegna IT
  - Tiscali
  - CRS4
- Co-founder and currently President of the “Apriti Software!” association, promoting Open Standards and Data Formats;
- Co-founder of the GULCh Linux Users Group, the first one in Sardinia, Italy (1996)

# What is Opera Software?

- Known worldwide for its web browser, Opera Desktop;
- Headquarters in Oslo, offices worldwide
- Opera invented a number of features that are common in today's browsers:
  - Tabbed browsing
  - Sessions
  - Mouse gestures
  - Speed dial
- Although Opera's main products are not Free Software, the company is well known for actively promoting Open Standards (e.g.: CSS, HTML5, WebM...)

# NTP: common beliefs and mistakes

Let me start with a simple question...

# What do you know about NTP?

# What do you know about NTP?

- General understanding of NTP is probably inversely proportional to its adoption;
- What people (and, unfortunately, many sysadmins) commonly think:
  - Ntp is a protocol designed to synchronize computers' clocks
  - You use it you by configuring ntpd on your machines, pointing it to a one (or maybe more) “upstream” servers out there, and syncing their clocks to the servers'
  - This magically synchronizes the clock on **any** possible system
- If, for any reason, ntpd can't be used, then it's OK to run ntpdate in cron once every minute/hour/day
- **Do you know how do I call this?**

# What do you know about NTP?

- Another common misbelief about NTP is that it aims to synchronize a computer's clocks to another computer's
- The real thing is that NTP aims to synchronize each computer's clock with UTC
  - What's the difference? Think about an orchestra: all instruments are tuned to a well-defined reference (e.g.: for a guitar, the 5$^{th}$ string is tuned to the “A” tune at 440Hz)
  - If the instruments were tuned with each other in a cascade fashion, the tuning's quality would be rather poor...
- This, and other protocol's “tricks” allow for the best results
  - Unfortunately, the quality of NTP on virtual machines is still rather poor...

# Bare-bones NTP

# Back to basics

The NTP protocol is defined in RFC 1305 (v.3) and 5905 (v.4)

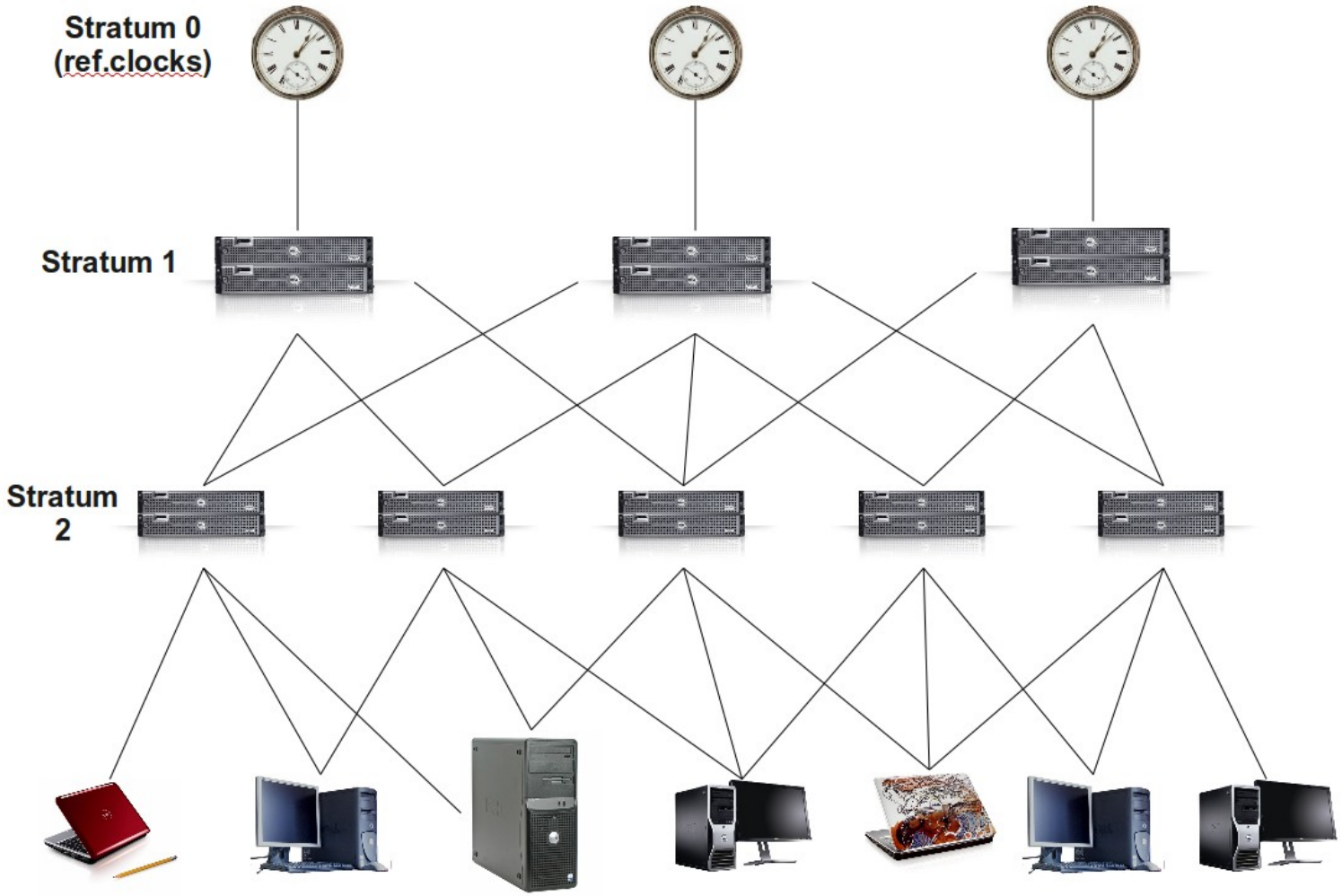
> *This document defines the Network Time Protocol version 4 (NTPv4), which is widely used to synchronize system clocks among **a set** of distributed time servers and clients. […] The NTP **subnet model** includes a number of widely accessible primary time servers synchronized by wire or radio to national standards. The purpose of the NTP protocol is to **convey timekeeping information** from these primary servers **to secondary time servers** and clients via both private networks and the public Internet.*

Doesn't this ring a bell?

# The NTP subnet model

- The RFC talks about a **subnet** model
  - We have **primary references** available on the Internet, and **secondary servers**
  - Secondary servers should be used to **synchronize clients on a LAN**
- It's a hierarchical, tree-like model where a small number of servers synchronizes a large number of clients
- Servers and clients are layered in **strata**, with primary servers sitting at **stratum 1**. Stratum number increases as we go down the tree up to the leaves, or stratum 15.
  - Reference clocks are said to sit at **stratum 0**

# The NTP subnet model (simplified...)



# The NTP subnet model

- Using public stratum 1 servers to synchronize single clients is both an abuse and a bad practice
  - An **abuse**, because you are really abusing a service that someone provides for free, and degrading the quality of that service
  - A **bad practice**, because **NTP is not meant to be used that way!**
- In the following slides we'll examine how NTP works and we'll see the implementation we adopted in Opera
- At the end of the presentation, we'll also see some special configuration, and a real debugging case

# Clients, servers, peers, *cast...

- If you read the RFC, you'll find information about *modes of operation* and *protocol modes*. I'll leave the gory details to you;
- For the scope of this presentation, it will suffice to say that:
  - a node participating in an NTP subnet can “be” any combination of: **client, server, peer**;
  - a node could use NTP in: **unicast, broadcast, multicast** and **manycast** modes;
  - We are **not** going to cover manycast here; and we'll cover NTP security in little detail
- The first design choice is the “casting” mode of the subnet

# Unicast

- In unicast mode, you point a node to a number of upstream servers, specifying their DNS name or IP address
- From a configuration point of view, it may be the simplest way
- What if you have 100 or more nodes configured already and:
  - you need to change one of your servers and you can't use the same IP or DNS name (for any reason);
  - your network is heavily loaded, and this is impacting the quality of the time service, or conversely:
  - you have so many clients that NTP adds a sensitive load to the network

# Unicast

- Unicast is often the first choice for sysadmins: it should actually be the **last** choice
- Cases where unicast is a good choice includes:
  - Secondary servers
  - Machines that you can't synchronize in any other way (for any reason)
  - You don't have any other option
- The advice here is: **avoid unicast as much as you can**

# Broadcast

- In broadcast mode, your servers broadcast an NTP packet every 64 seconds to the subnets it belongs to;
- After initialization, broadcast clients listen “passively” for these packets
- This means that the network traffic is quite limited
  - Maths say that each day has 24×60×60=86400 seconds
  - One packet every 64 seconds means 86400/64=1350 packets per day
  - Since each packet is about 100 bytes long, this amounts to about 135000 bytes per server per day
    - e.g.: four servers would consume about 527kB per day
  - Clients initialisation would add some more, but in normal cases this should be feasible

# Broadcast

- The problem with broadcast packets is that they are subnet limited
- If you have a huge number of subnets, a decent setup with broadcast could be difficult to achieve
  - e.g.: VLANs could help, but if you have 100 subnets, do you really want to have a few machines that have 100 (virtual) interfaces?
- **In large environments, broadcast is a no-op.**

# Multicast

- In multicast mode, your servers broadcast an NTP packet every 64 seconds to a multicast address;
  - The “well known address” for NTP is 224.0.1.1, aka ntp.mcast.net; you may use any multicast address anyway
- After initialization, multicast clients listen “passively” to these packets
- Network traffic is quite limited in this case, too
- The notable advantage is that **multicast is not subnet limited!**
  - Provided that your network equipment can be properly configured, NTP packets will reach all of your subnets

# Multicast

- What's bad about multicast? It is not in widespread use, so:
  - It's not well known by many network admins
  - It is not well implemented in many network devices (routers, switches, firewalls)
  - It may be badly implemented even on operating systems
  - It may require “additional” software to work properly
- The upside is that, once in place, it scales quite well and needs little maintenance.

You will find a good description of pro's and con's of all these configurations (and more) in Brad Knowles' article. See bibliography.

# Which solution?

- My advice is to **use multicast whenever possible**
- I used multicast in three different environments;
- Each time it required a lot of support from the network admins to have it started, but once done, it was a great experience for everybody:
  - For me, because I like teamwork, and learned something new each time;
  - For the network admins who, after some initial resistence, were quite happy to learn and apply something they only learnt on the books (ok, that wasn't always the case!)
  - For users, who got a **great** NTP implementation

# What about manycast?

- I never tried manycast. It looks quite interesting, but it's also quite new.
- I guess it would take much more effort to implement a manycast solution, for the same reasons it is difficult to implement the multicast solution, and more;
- Nevertheless, I would be damn happy to try it! Anyone in? :)

manycasting is an **automatic dynamic discovery and configuration** paradigm. It is distinct from anycasting, where a single service provider is selected from a number that may respond to a multicast invitation. Manycasting is designed for highly robust services where **multiply redundant respondents are continuously evaluated** and quasi-optimal subsets mitigated using engineered algorithms.[...]

The NTP Manycast scheme uses an expanding-ring search with pruning and variable poll rate in order to minimize network overhead. [...] **A client trolls the nearby network neighborhood looking for available manycast servers,** authenticates them [...] and then evaluates their time values with respect to other servers [...]. The intended result is that **each manycast client mobilizes client associations with the "best" three nearest available manycast servers**, yet automatically reconfigures to sustain this number should one or another degrade, fail or become compromised

http://www.ece.udel.edu/~mills/autocfg.html

# What about security?

- NTP implements **symmetric cryptography and authentication** since at least v3 (1992)
- Initially based on the old MD5 checksums, it has added some more over time
- **v4 provides for public key cryptography and auth**
  - I did some research on that, and I got it working but not "properly"
  - The status at the time was that implementation details could vary between **minor** versions of the ntpd (e.g., the implementations in version 4.x.0 and 4.x.1 could be different – and work differently)
- My **suggestion** is to stay on the symmetric, and watch for a better future implementation of pubkey

# Subnet architecture

- There are not many suggestions for good architectures out there. If you want to start from scratch, you'll probably need to read a lot to learn who's authoritative and who's not
  - e.g., Prof.D.Mills probably is :)
- My suggestion is to take a good read of the documents in bibliography
- The Sun blueprints mentioned are quite old, but they still offer a very nice introduction to the full subject: protocol, implementation, and debugging.

# Choosing the hardware

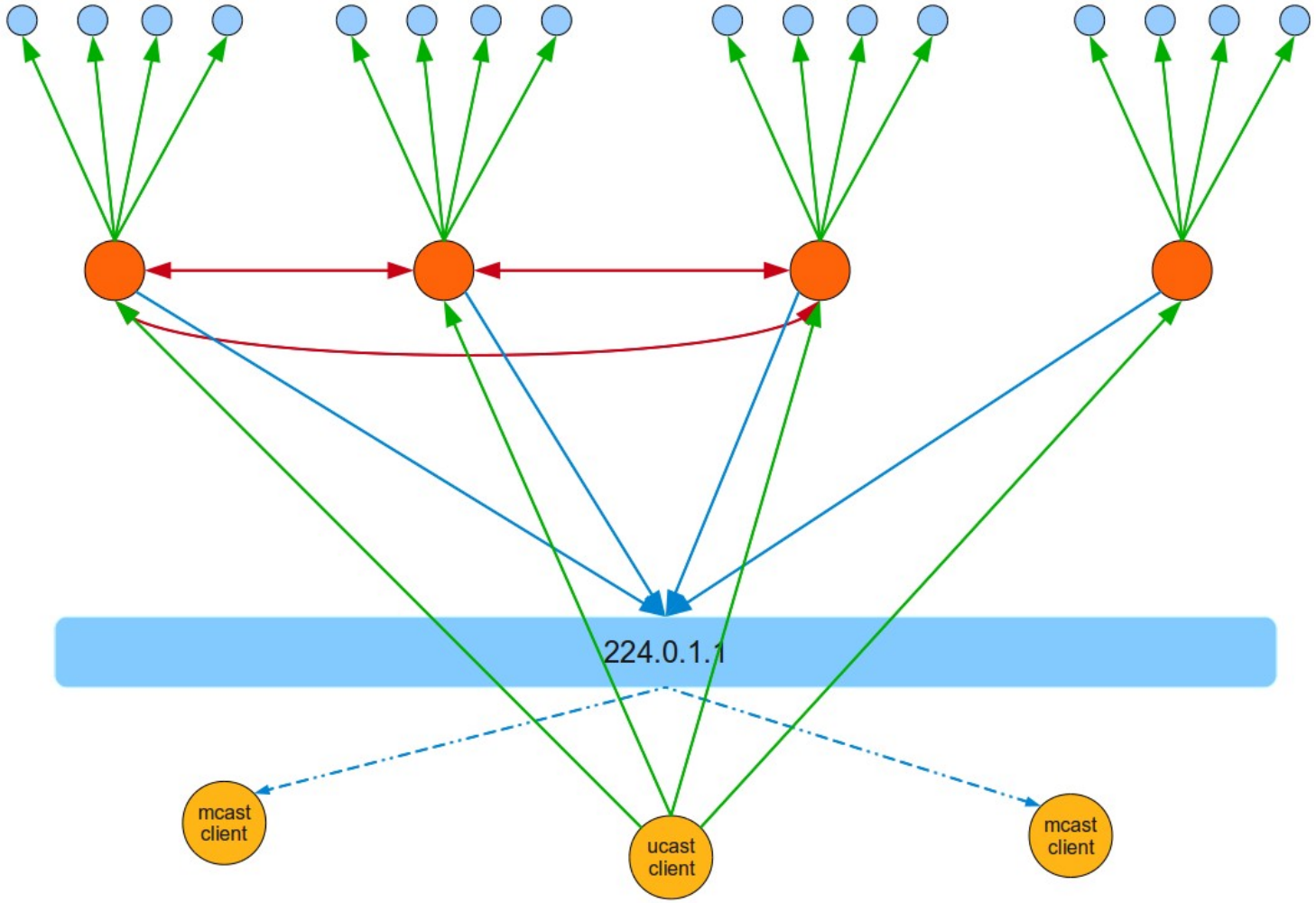
- You need either **lightly loaded servers** that are already in use, **or old, decommissioned, unused servers** that you wouldn't use otherwise
  - You don't need bleeding-edge, super-multicore servers for NTP; in fact, ntpd is single-threaded, and wouldn't gain anything from that
  - You don't want to mask it with NAT, or load balancing solutions, or any other trick like that; they would actually confuse your clients
  - You don't want to put it on task-specific hardware (e.g. Routers and switches): they are very good for the task they were designed for, but poor at NTP
  - You don't want virtual machines: ntpd assumes a stable CPU, and virtual CPUs aren't

# Opera's NTP subnet

# Opera's subnet architecture

- Our architecture is a 3+1
- Each one of the four secondary servers will get its time from (at least) two different primaries
  - You will find an always updated list of public primaries on support.ntp.org, along with geographic location and rules of engagement for each server (check bibliography)
  - Primaries are chosen as closest as possible
- None of the four will have any primary in common
  - Primaries' reference clocks should be of different type as much as possible
- Three servers will be peers, while the other one will run solo
- Servers will use shared key cryptography

# Opera's subnet architecture



# What if...?

- ...the “solo” server looses reach from its upstreams and starts drifting?
  - Clients will see the other three running smoothly together, and follow one of them
- ...one of the three peering looses reach from its upstreams?
  - It shouldn't drift because of the peering; and if it drifts, it will be discarded by the clients for the same reasons above
- ...all servers lose reach to their upstreams?
  - They will drift differently, the three peering will go together, and the solo will go... solo :)
  - The clients will collect statistics and decide which one is the best for them

# Configuring cryptography and auth

- It is as simple as:
  - Generate a keyfile using **`ntp-keygen -M`**
  - Copy the generated file on all servers and clients
  - Choose (at least) one key and tell the servers to propagate packets “signed” with that key
    - I personally prefer using one key per server, so that I can “identify” the server with the key number
  - Tell the clients to trust the key(s) used by the servers
- Using an automated installation and/or a configuration system can help a lot (e.g.: FAI, cfengine, puppet...)

# Sample key file

```
# ntpkey_MD5key_cooper.3494425072
# Sat Sep 25 19:37:52 2010
 1 MD5  aI4Iqym@L}n;fe: # MD5 key
 2 MD5  9J'%p_AFQ23mwK! # MD5 key
 3 MD5  {6+L~+QljbAk[m9 # MD5 key
 4 MD5  "ga{mtas{QC*c:c # MD5 key
 5 MD5  "<VquB5aJ7.H+o= # MD5 key
 6 MD5  -o,1R6ya$ok6oGE # MD5 key
 7 MD5  ]U$"s6XlM(*C-Z" # MD5 key
 8 MD5  V2QT*QsC&Q~7r*} # MD5 key
 9 MD5  q/(MYy*ai5\2Bua # MD5 key
10 MD5  ))mvcG00k+n]ibi # MD5 key
11 MD5  'n_a8j|^m=Q:dTq # MD5 key
12 MD5  U+D/8LuWtQOZei\ # MD5 key
13 MD5  a48&$"LrhXgze(@ # MD5 key
14 MD5  ~2KA{YaL_BU;V"p # MD5 key
15 MD5  Ua{=/y>wOK\Yk3> # MD5 key
16 MD5  1Q^J6'OP[[4D-OS # MD5 key
```

# Sample peering server configuration file

```
driftfile  /var/lib/ntp/ntp.drift
keysdir    /etc/ntp
keys       /etc/ntp/ntp.keys

trustedkey 4

server stratum1-1.xmp iburst dynamic
server stratum1-2.xmp iburst dynamic
peer peer1
peer peer2

broadcast 224.0.1.1 key 4 ttl 7

restrict -4 default kod notrap nomodify nopeer
restrict -6 default kod notrap nomodify nopeer
restrict 127.0.0.1
restrict ::1
```

# Sample solo server configuration file

```
driftfile  /var/lib/ntp/ntp.drift
keysdir    /etc/ntp
keys       /etc/ntp/ntp.keys

trustedkey 1

server stratum1-3.xmp iburst dynamic
server stratum1-4.xmp iburst dynamic

broadcast 224.0.1.1 key 1 ttl 7

restrict -4 default kod notrap nomodify nopeer
restrict -6 default kod notrap nomodify nopeer
restrict 127.0.0.1
restrict ::1
```

# Sample multicast client configuration file

```
driftfile /var/lib/ntp/ntp.drift
keys /etc/ntp/ntp.keys

trustedkey 1 2 3 4

multicastclient 224.0.1.1

restrict -4 default kod notrap nomodify nopeer noquery notrust
restrict -6 default kod notrap nomodify nopeer noquery notrust

restrict 127.0.0.1
restrict ::1
```

# Summing up

- This architecture should guarantee for good time accuracy:
  - For very long-lasting outages impacting many (not all) of them
  - For short- to medium-lasting outage impacting all of them
- It provides **high redundancy**: a single server full outage doesn't significantly impact the whole service
- Thanks to the multicast implementation, a server can be replaced **transparently** for the clients
- Changing the solo server has **no impact**; changing one of the peering **may** require reconfiguration and very short downtime for each one

# Summing up

- None of the secondary servers in use in Opera is **dedicated** to NTP
- Thanks to a careful choice of the servers, it is providing a very good service (up to the millisecond offset, or less)
- Multicast is not reaching all our subnets yet, we are adding more and more over time.
- “External” users of the service helped us testing it, and they were quite satisfied.
- **Any question before we move to more complex setups?**

# More complex setups: repeaters

# Reaching unreachable subnets

- There may be subnets of your sites that can't be reached by the multicast packets propagated by your NTP servers, but allow multicast on the inside
- In this case, you may want to configure further secondary servers in that subnet (at least two) that we call *repeaters*.
- E.g., you will configure two machines, each one using two non-overlapping secondaries as upstreams, and propagating multicast in their subnet or “island”
- If you need more redundancy or accuracy, you may want to add more repeaters, or set-up separate secondaries for this “island”

# More complex setups: clusters

# Syncing cluster members

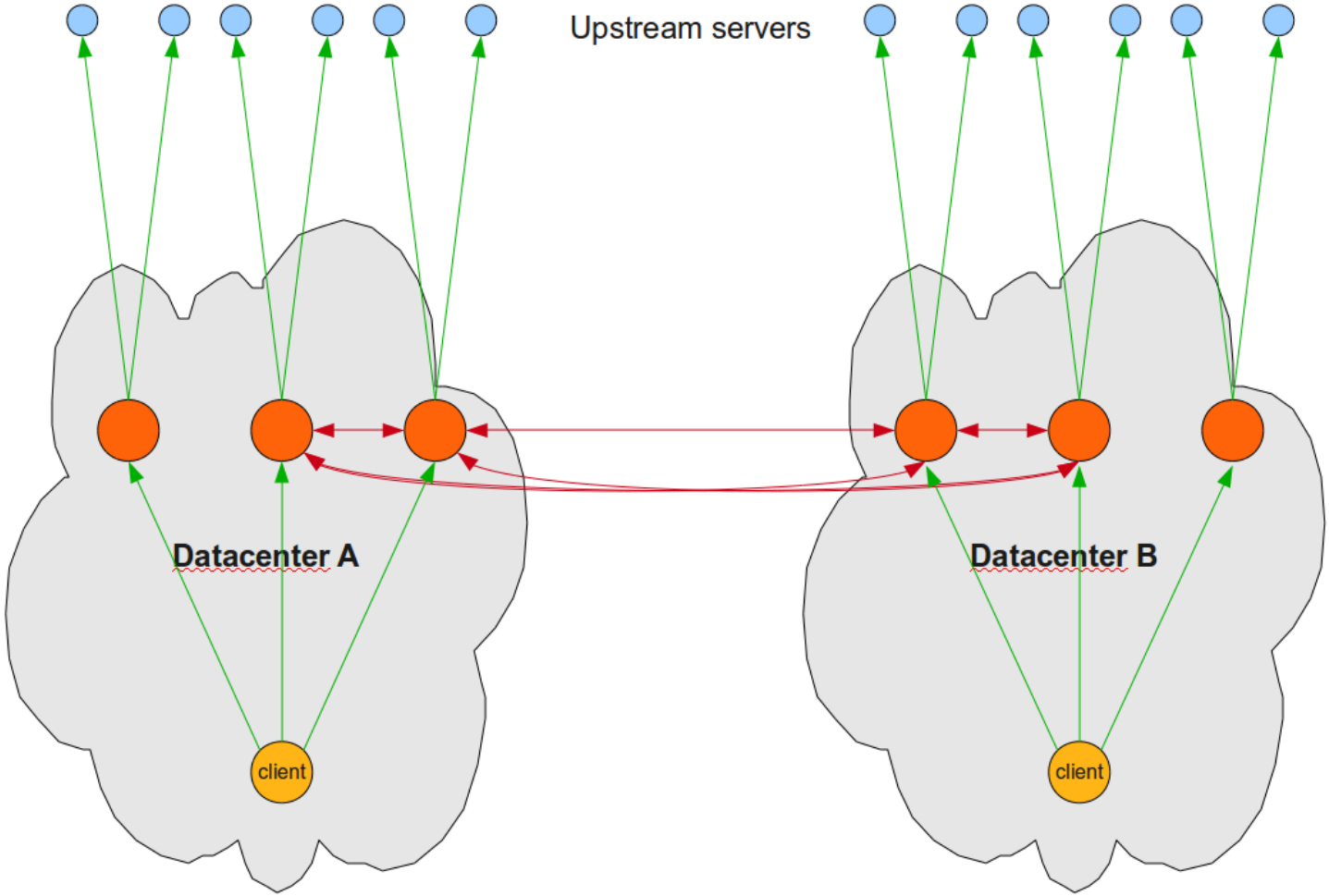
- It's always important that cluster members have their clocks in good sync
- For small clusters (e.g.: two to four machines) a solution is to add a peering relation between cluster members
  - You configure cluster members as “regular” clients, adding a “peer” directive for all other cluster members
- **Take good care at not creating loops!** (NTP has no loop detection mechanism)

# More complex setups: geographic clusters

# Syncing datacenters

- You may have a service that is served in different, geographically-dispersed datacenters
- In this case, it may be important that the clocks in the two datacenters evolve accordingly
- If there is a chance that the two datacenters can reach each other in adverse conditions, a **possible** implementation could be the following

# Syncing two datacenters



# When things go wrong...

# Debugging NTP

- Things may go wrong sometimes
  - that's one of the reasons why System Administrators have a job, after all...
- There are no general rules to debug a system problem; NTP is no exception
- In the next few slides I'll show a concrete example; for general advice and tools, please have a look at the bibliography

# The Xen server and the “resounding” offset

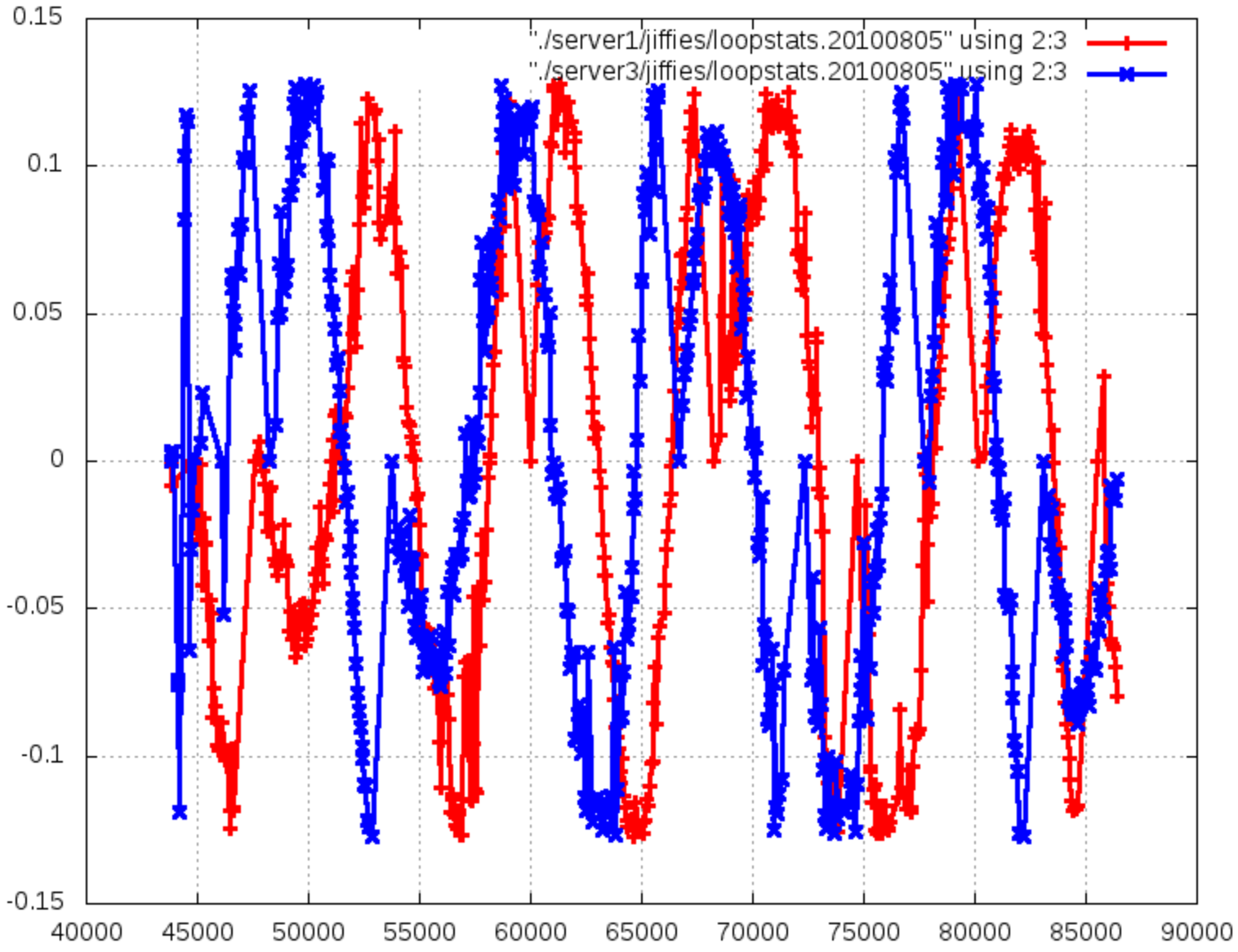
- Two pairs of Xen servers in two different locations: same OS (Debian 5 “Lenny”), same hardware, same configuration, same everything!
- The Debian wiki suggested two different configurations for Xen servers, and I tried them, but without success
- scanning the logs revealed the time resets
- A number of sources revealed this kind of problem on Xen is quite common
- Setting a “normal” configuration on both pairs worked for one, but didn't for the other:

# Step 1: collect statistics

- How was the offset evolving? It was time to activate **statistics**
- The first pair synced perfectly, the second had the offset oscillating up and down in wider and wider waves, until ntpd reset the clock
- This cycle repeated over and over
- All known/documented solutions failed, or made the problem worse
- What to do?

# Step 1: collect statistics

- You don't need extra tools to gather ntp statistics: ntpd provides them for free
  - We activated two types: loopstats and peerstats
    - Loopstats tell us how the local clock is evolving, peerstats record information each time a packet is received from a source
  - See ntpd documentation to make sense of the collected data
- Using gnuplot, it was easy to compare the four Xen servers over a long interval, and...


"./server1/jiffies/loopstats.20100805" using 2:3
"./server3/jiffies/loopstats.20100805" using 2:3
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
40000
45000
50000
55000
60000
65000
70000
75000
80000
85000
90000

# Step 2: analyse statistics

- To activate some stats:

```
statsdir /var/log/ntpstats/
statistics loopstats peerstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
```

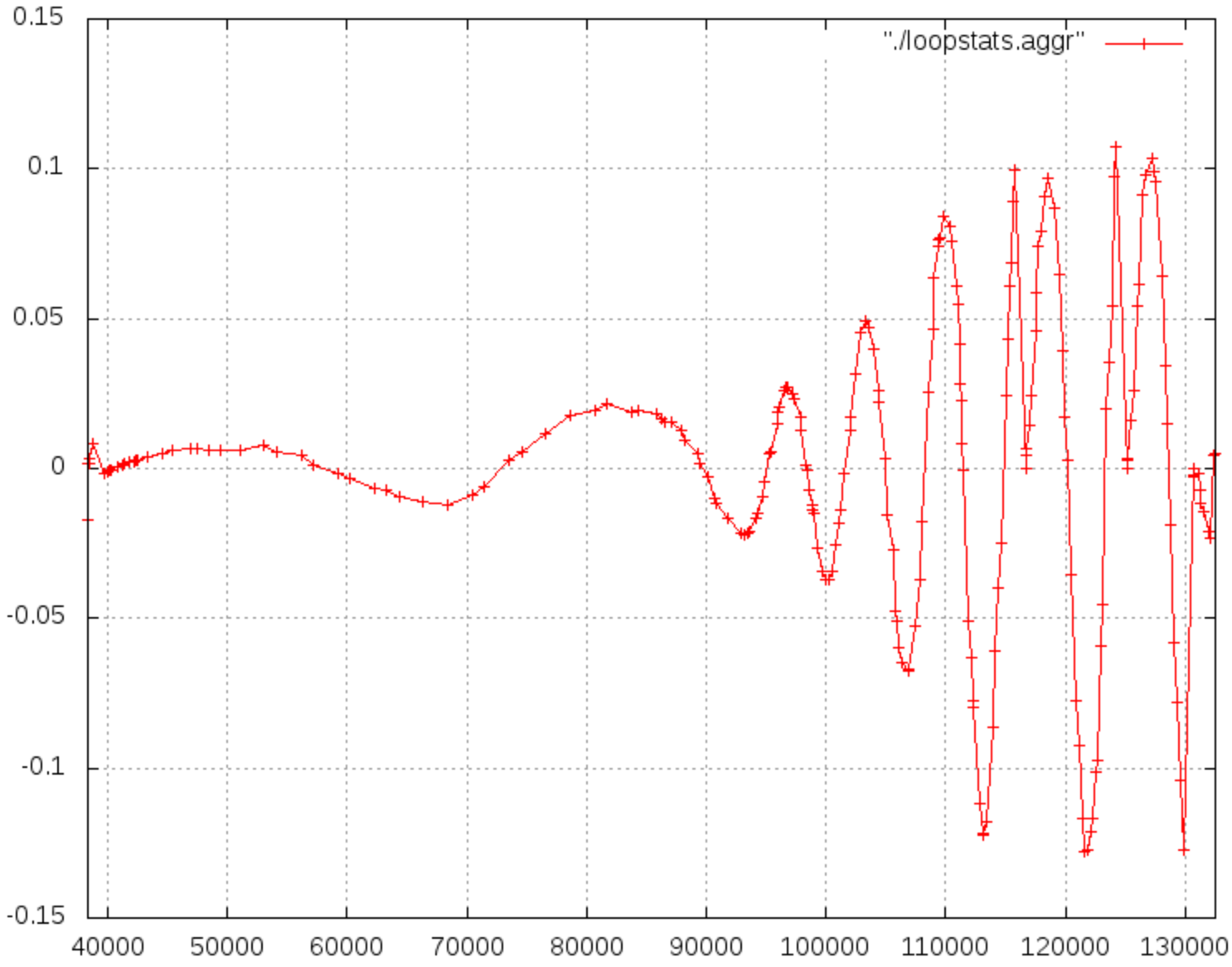
- To plot files in gnuplot:

```
plot "./server1/loopstats" using 2:3 with
linespoints, "./server2/loopstats" using 2:3 with
linespoints
```

- This will show a plot that will help you compare how the offset is evolving

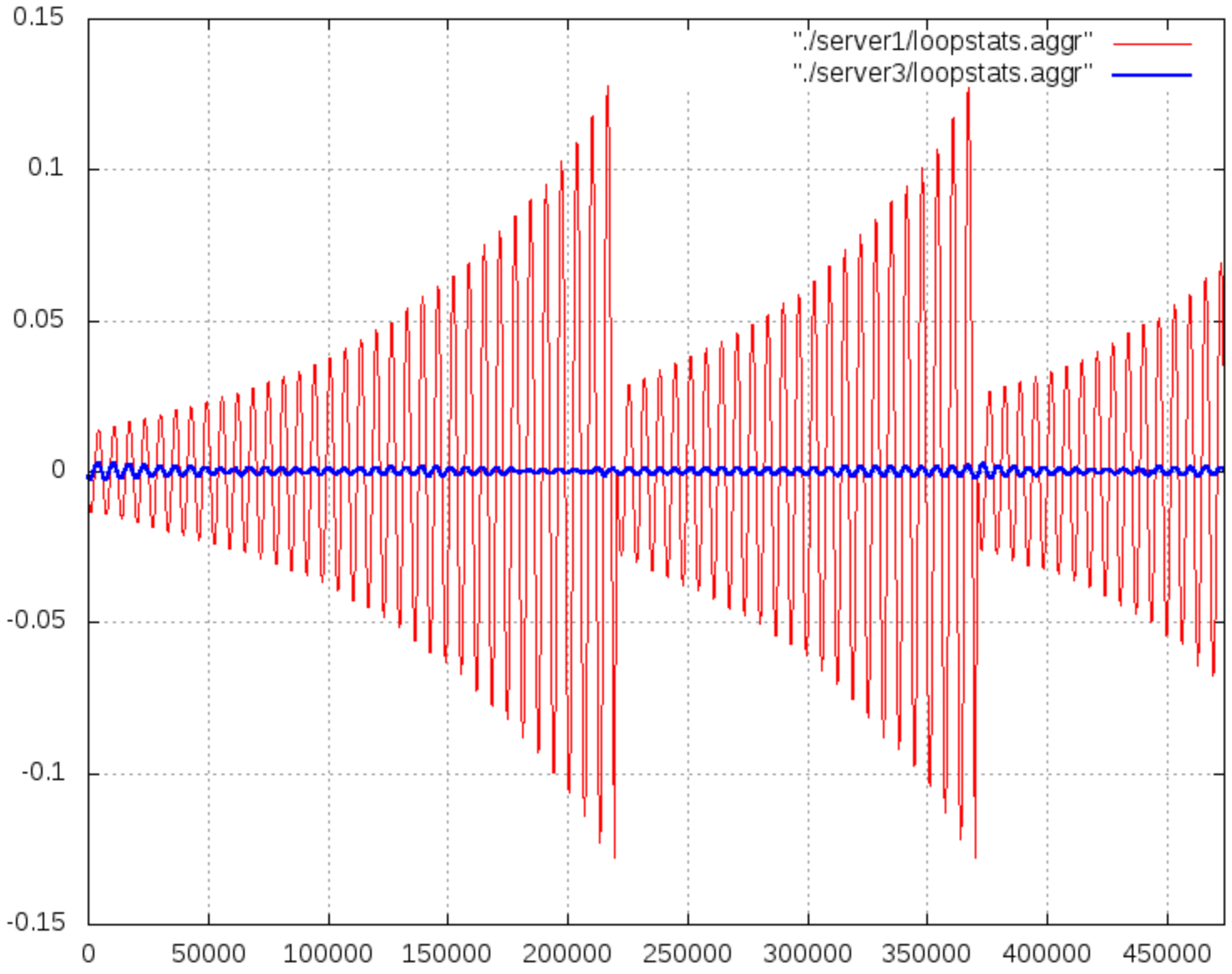
# Step 3: check for differences

- Bad behaviour indeed...
- Maybe switching to unicast and letting the client choose when to poll its upstream could help...
- Well... it didn't
- More searches for solutions resulted in 99% crap and 1% irrelevant information
- A difference in /etc/adjtimex raised some hopes, but they were short lived...


"./loopstats.aggr"
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
40000
50000
60000
70000
80000
90000
100000
110000
120000
130000

# Step 4: ask for help

- It was time to ask for help. There are a lot of knowledgeable people in SAGE, and I am a SAGE member...
- Unfortunately, no one answered...
- While waiting for an answer, I restored the same configuration on all servers and kept looking for a solution
- When I was almost decided for manual calibration, I found there was an ntp IRC channel. Why not to ask there?
- In the meanwhile...


"./server1/loopstats.aggr"
"./server3/loopstats.aggr"
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
0
50000
100000
150000
200000
250000
300000
350000
400000
450000

# Step 5: experts to the rescue: #ntp

mlichvar: bronto: i think i have seen the same problem some time ago

bronto: mlichvar: good... erm... sort of ;) How did you manage to solve it?

mlichvar: bronto: i was just helping one guy and he didn't solve it :)

mlichvar: bronto: it looked like broken PLL in the kernel

bronto: mlichvar: erm... what's a PLL? :(

mlichvar: the thing that adjusts offset and frequency

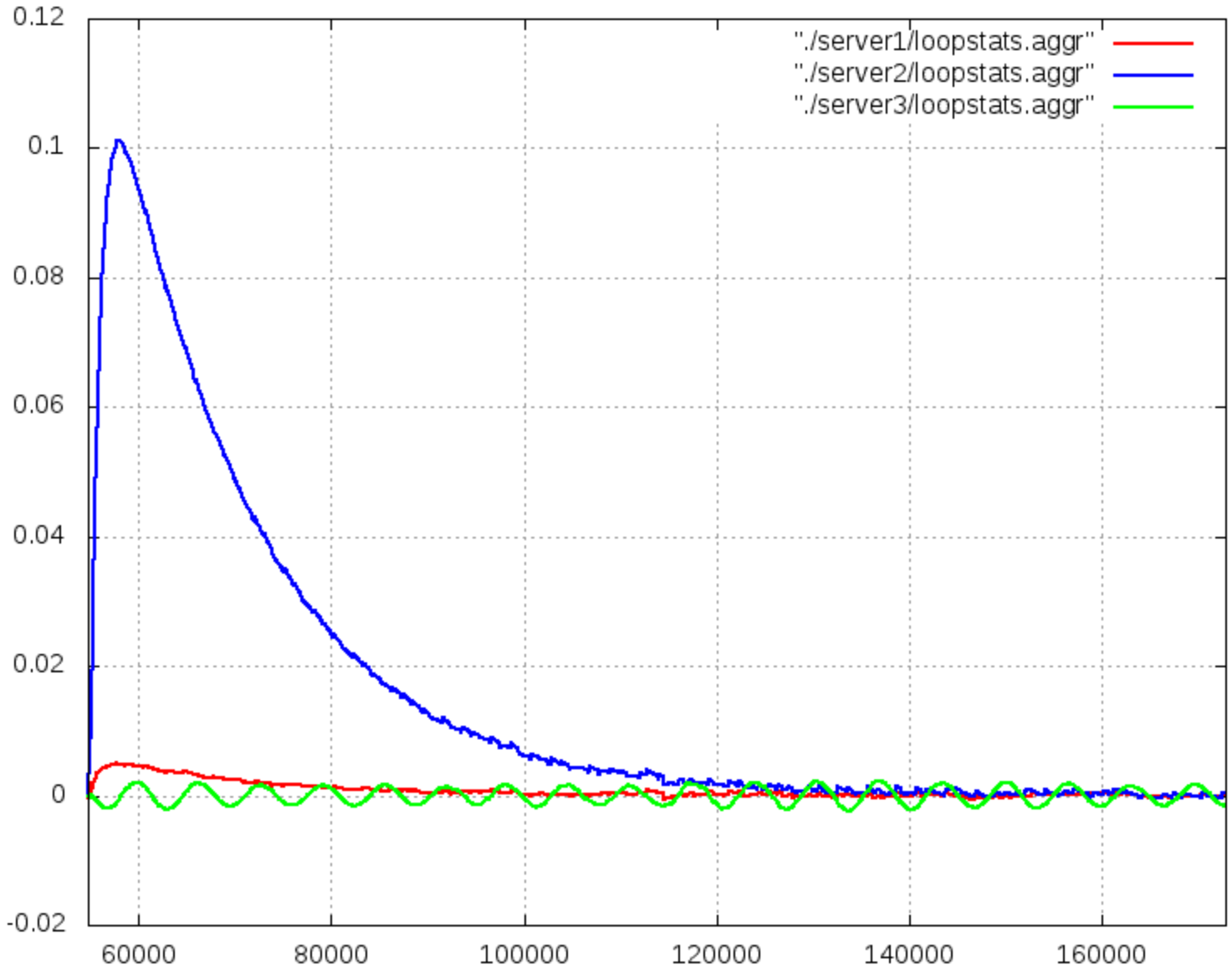
mlichvar: in the offset plot it looked like the time constant was too short

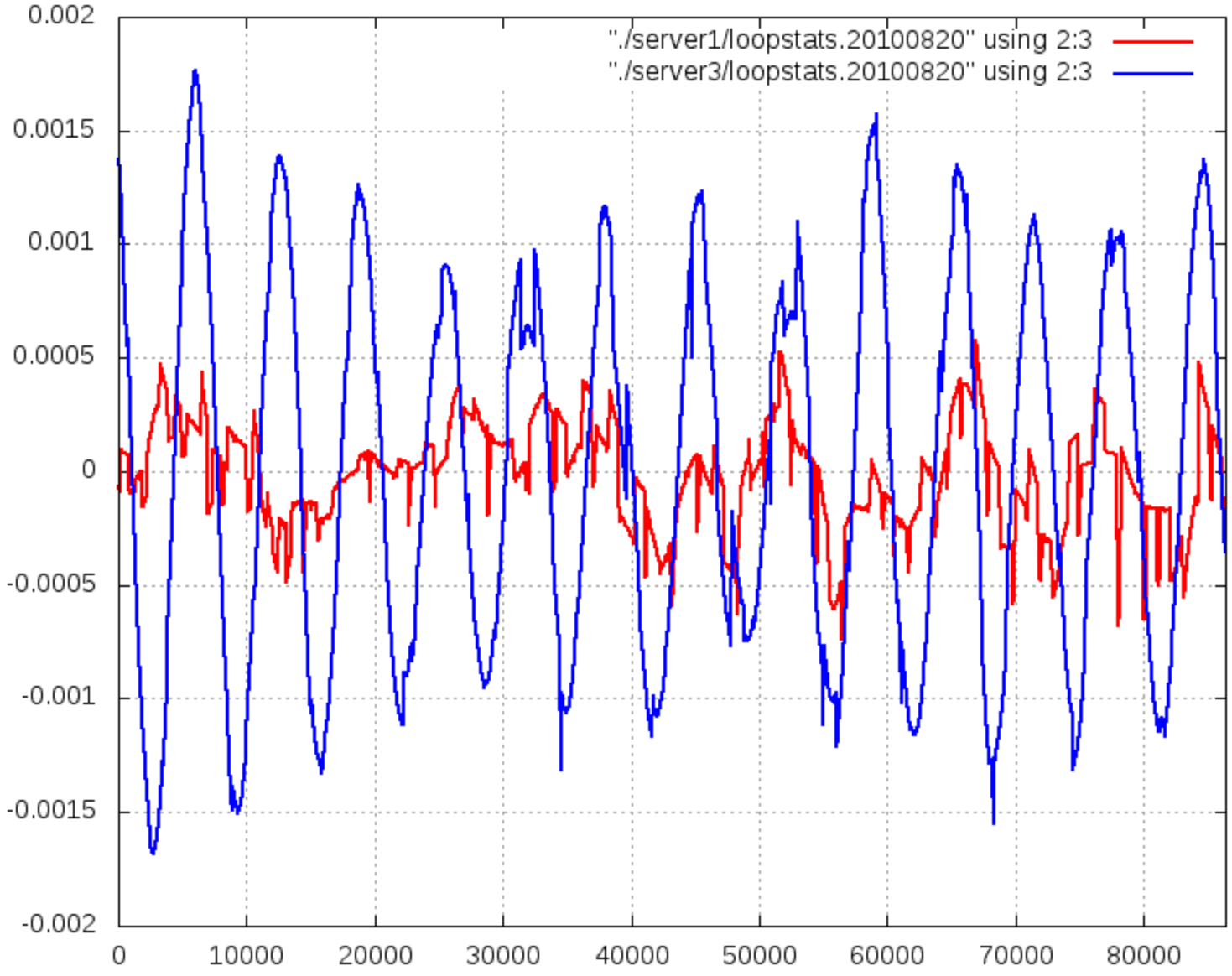
mlichvar: there is one easy thing you could try first

mlichvar: disabling kernel discipline by adding "disable kernel" to ntp.conf

mlichvar: if that works, it's definitely a kernel bug

***bronto adding disable kernel to ntp.conf on the Xen servers


"./server1/loopstats.aggr"
"./server2/loopstats.aggr"
"./server3/loopstats.aggr"
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
60000
80000
100000
120000
140000
160000


"./server1/loopstats.20100820" using 2:3
"./server3/loopstats.20100820" using 2:3
0.002
0.0015
0.001
0.0005
0
-0.0005
-0.001
-0.0015
-0.002
0
10000
20000
30000
40000
50000
60000
70000
80000

# Some bibliography

# Bibliography

RFCs are **the** authoritative source of information for any protocol (assuming that such RFC exists), so:


- Mills, Martin, Burbank, Kasch: Network Time Protocol version 4: protocol and algorithms specification; RFC 5905; June 2010
- Mills: Network Time Protocol (version 3) specification, implementation and analysis; RFC 1305; March 1992

# Bibliography

Although a bit outdated, the following Sun BluePrints "Using NTP to Control and Synchronize System Clocks", by Deeths and Brunette, are an excellent summary:

- Part I: Introduction to NTP; July 2001
- Part II: Basic NTP administration and architecture; August 2001
- Part III: NTP monitoring and troubleshooting; September 2001

# Bibliography

An excellent article about the pros and cons of different solutions for an NTP infrastructure. It assumes some knowledge though:

- Brad Knowles: building scalable NTP server infrastructures; in “;login:”; October 2008
  http://www.usenix.org/publications/login/2008-10/pdfs/knowles.pdf

# Bibliography

Web sites at ntp.org are, of course, more than authoritative:

- Community support:
  http://support.ntp.org
- Public NTP services are available at:
  http://support.ntp.org/bin/view/Servers/WebHome
- ...but be sure to **read the rules of engagement** first!!!
  http://support.ntp.org/bin/view/Servers/RulesOfEngagement
- ...both the general ones (above) and those **specific to each server**, e.g.:
  http://support.ntp.org/bin/view/Servers/NtpOneRnpBr

# Bibliography

For ntpd, the reference implementation which is normally installed by default on all major Linux distributions, the authoritative sources are:

- Home of the Network Time Protocol:
  http://www.ntp.org/
- NTP Documentation archive:
  http://doc.ntp.org/
- NTP Debugging Techniques
  http://doc.ntp.org/4.2.6/debug.html
- Monitoring Options
  http://doc.ntp.org/4.2.6/monopt.html

# Bibliography

Do you want to know the full story about the debugging of the Xen servers?

- For the thread in the SAGE mailing list, check these:
  http://mailman.sage.org/pipermail/sage-members/2010/msg01058.html
  http://mailman.sage.org/pipermail/sage-members/2010/msg01057.html
  http://mailman.sage.org/pipermail/sage-members/2010/msg01069.html

- And if you have the same problem on any of your Xen servers running Lenny, check the workaround #3 at:
  http://wiki.debian.org/Xen#A.27clocksource.2BAC8-0.3ATimewentbackwards.27

# www.opera.com