Network tuning for zone transfers in (lossy) Long Fat Networks

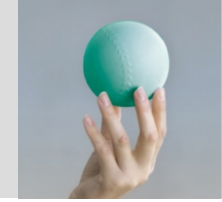
14.05.2015 – RIPE70 Amsterdam

Marco Prause <prause@denic.de>

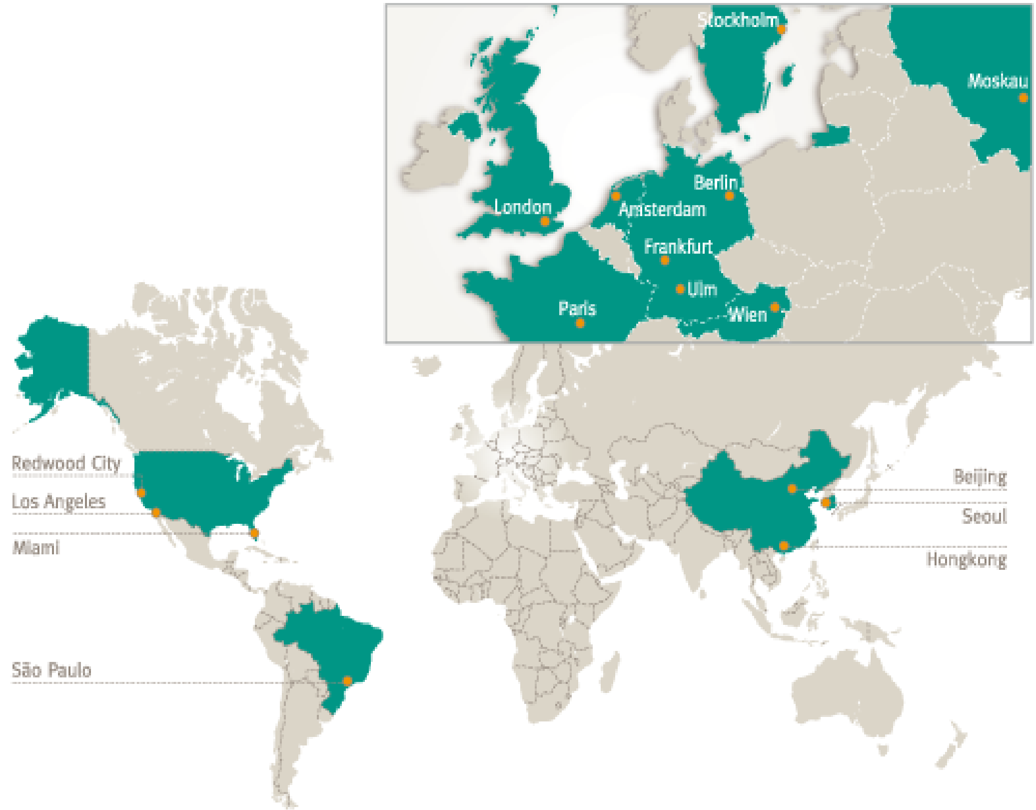
Agenda

1. Introduction
2. Path-MTU-Discovery and Maximum-Segment-Size
3. Having a short look at involved TCP Congestion Control algorithms
4. Changing the algorithm – changing the game ?

1. Introduction



- Regsitry for .de
- Domains : over 15 million
- Nameserver locations : 16
- Zonefile size : 1.5 GByte
- DNSSEC domains : 20.000
- Average IXFR size : 185 MByte

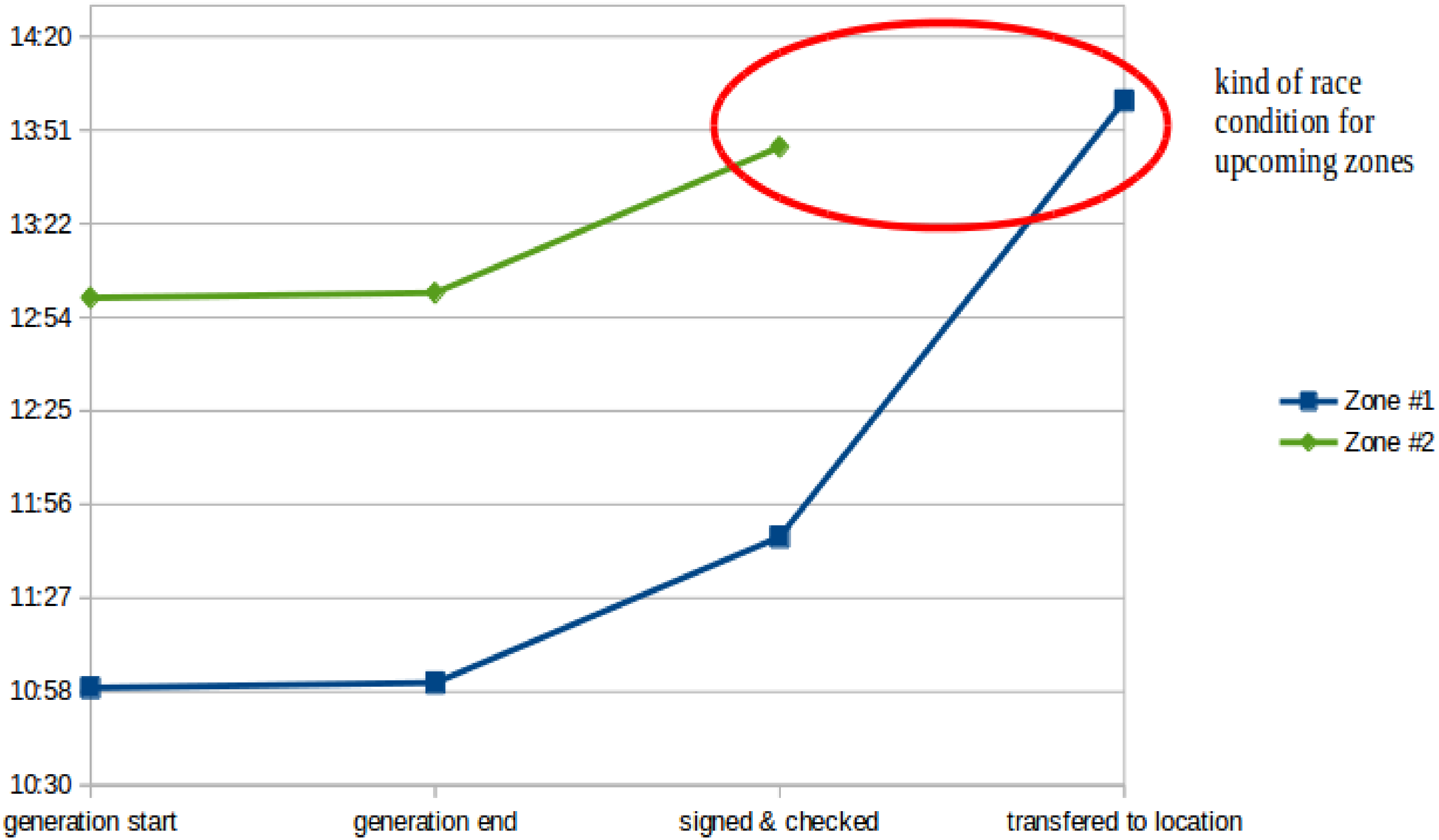


1. Introduction

- Why should we take a deeper look at the network ?
  - Increasing zonefile and dnssec = growing incremental zonetransfer
  - To locations far far away, we saw that the transfers last longer
  - In some cases the transfers
    - Didn't fit in our zone generation cycle
    - Or their incremental transfers were cancled and often an AXFR was started
  - Beside latency we also see packetloss on some paths, which is also decreasing our throughput

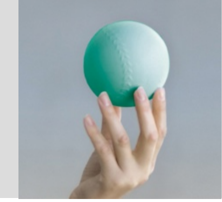
1. Introduction

- Why should we take a deeper look at the network ?



2. Path-MTU-Discovery and Maximum-Segment-Size

- Good news
  - PMTUD is working like a champ
  - also MSS is adjusted by the interface MTU
- BUT
  - Wireshark says : PMTUD is not influencing the MSS
  - only the fixed MTU of the interface is taken to compute the MSS



2. Path-MTU-Discovery and Maximum-Segment-Size

- So we had two possibilities to fix that issue
  - Fixed MTU of 1300 on the interfaces
    - Will also be used for LAN traffic and therefor also decrease the MTU on the LAN
  - Let our VPN-Concentrator change the MSS inside the flow
    - Thanks to MSS clamping we could rewrite the MSS during the initial TCP handshake
    - So both endpoints learn the correct Maximun Segment Size
- After enabling MSS clamping we saw a small improvement concerning fragmentation,
  but not enough to handle traffic to our locations with high latency and additional packetloss

3. Having a short look at involved TCP Congestion Control algorithms

- There are a few TCP-Algorithm in the wild, e.g. :
  - BIC
  - CUBIC
  - Veno
  - Illinois
  - Hybla
  - ...
- we focused at the most promising three – TCP-CUBIC, TCP-Illinois and TCP-Hybla

3. Having a short look at involved TCP Congestion Control algorithms

- TCP-Cubic

„TCP Cubic attempts, like Highspeed TCP, to solve the problem of efficient TCP transport when **bandwidth×delay is large**. TCP Cubic **allows very fast window expansion**; however, it also makes attempts to slow the growth of cwnd sharply as cwnd approaches the current network ceiling, and to treat other TCP connections fairly.“

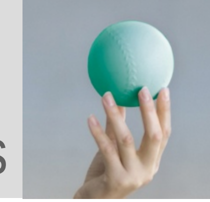
(http://intronetworks.cs.luc.edu/current/html/newtcps.html)

3. Having a short look at involved TCP Congestion Control algorithms

- TCP-Illinois

„TCP-Illinois is a variant of TCP congestion control protocol, developed at the University of Illinois at Urbana-Champaign. It is **especially targeted at high-speed, long-distance networks. ... achieves a higher average throughput** than the standard TCP, allocates the network resource fairly as the standard TCP, is compatible with the standard TCP...“

(http://en.wikipedia.org/wiki/TCP-Illinois)

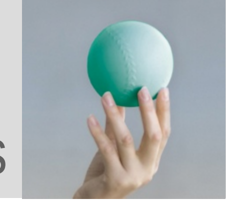


3. Having a short look at involved TCP Congestion Control algorithms

- TCP-Hybla

„TCP-Hybla was designed with the primary goal of counteracting the performance unfairness of TCP connections with **longer RTTs**. TCP-Hybla is meant to overcome performance issues encountered by TCP connections over terrestrial and satellite radio links. These issues stem from **packet loss due to errors** in the transmission link being mistaken for congestion, and a long RTT which limits the size of the congestion window“

(http://www.satnac.org.za/proceedings/2012/papers/2.Core_Network_Technologies/15.pdf)



3. Having a short look at involved TCP Congestion Control algorithms

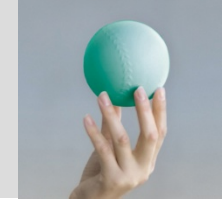
- The test setup for emulating the latency and packetloss...
  - RTT ~ 300 ms
  - Loss rate ~ 10 % averrage
- …was installed quite easy
  - 2 x Linux CentOS 6
  - 1 x FreeBSD 10
    - Dummynet/IPFW for simulation of latency and packetloss

And the winner is : TCP-Hybla

- Although they are quite close together, tcp-hybla did the best job at the simulated lossy LFN
  - Latency : 300 ms
  - Lossrate : 10 %

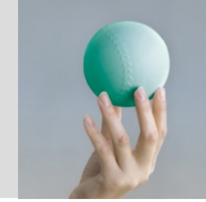
Algorithm	***Throughput***
Cubic	10 KByte/s
Illinois	15-20 KByte/s
Hybla	60-80 KByte/s

4. Changing the algorithm – changing the game ?



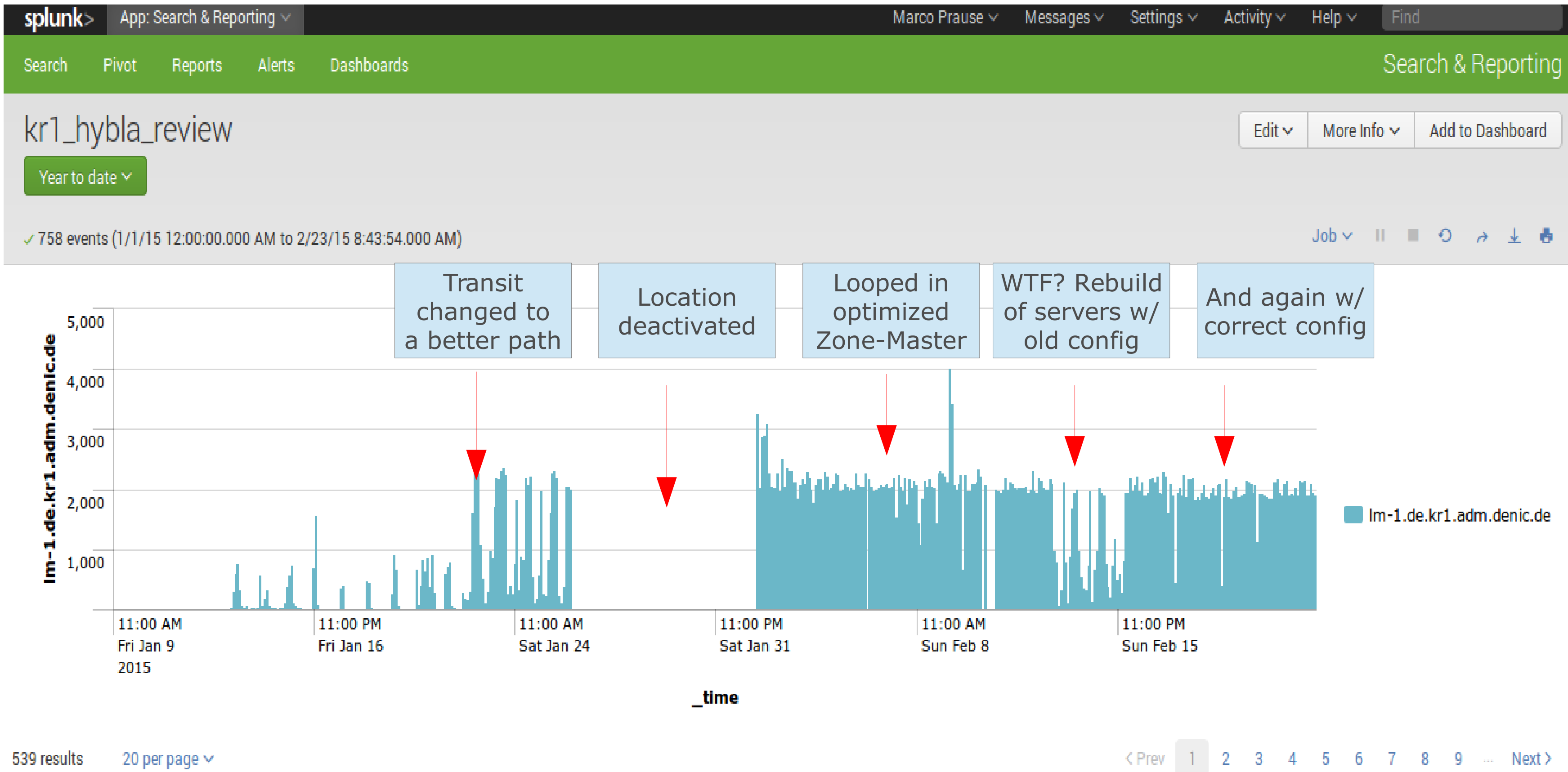
- Easy to activate at our Linux servers (sender)
  - # ls /lib/modules/`uname -r`/kernel/net/ipv4/
  - # modprobe tcp_hybla
  - # echo "hybla" > /proc/sys/net/ipv4/tcp_congestion_control

- On client's side (receiver)
  - net.ipv4.tcp_sack = 1
  - net.ipv4.tcp_timestamps = 1
  - net.ipv4.tcp_window_scaling = 1



4. Changing the algorithm – changing the game ?

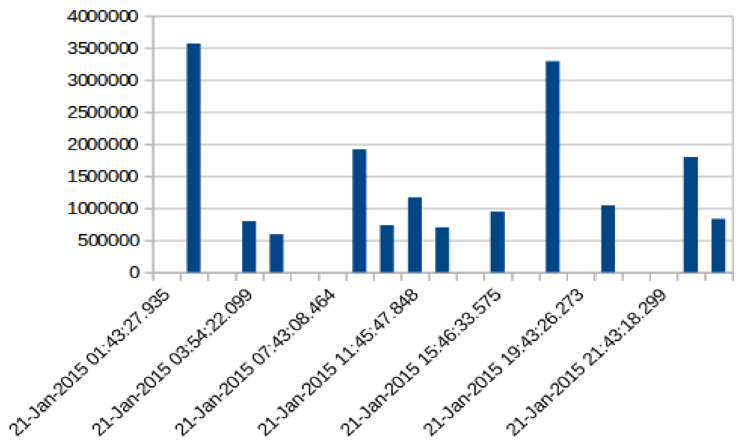
- And here we go...
- Zonentransfer-Rates in KByte/s (Location Seoul)

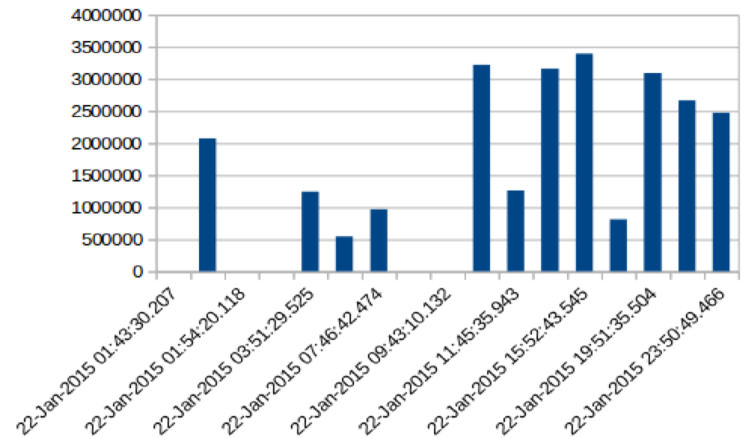


4. Changing the algorithm – changing the game ?

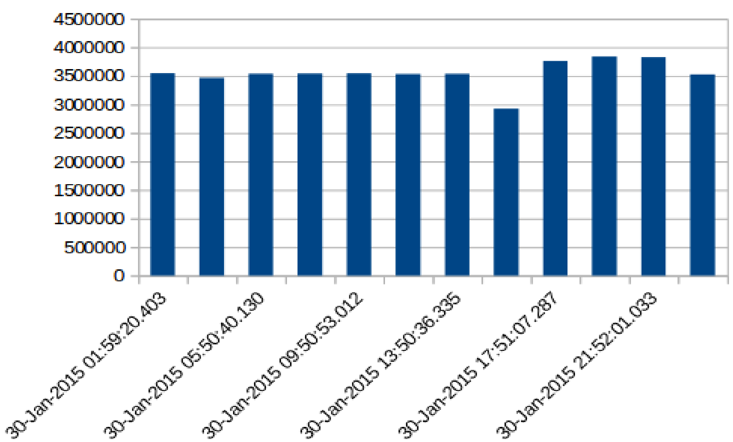
- Zonen transfer rates in Byte/s (Location Bejing)

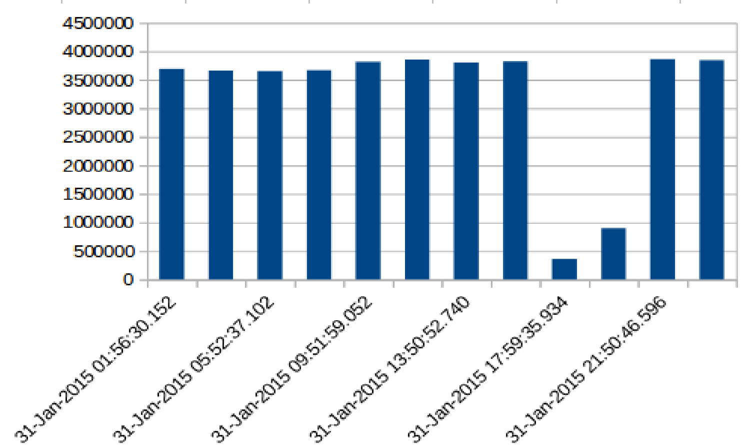
without Hybla

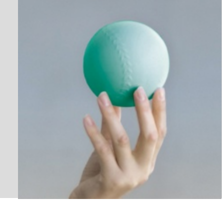




with Hybla

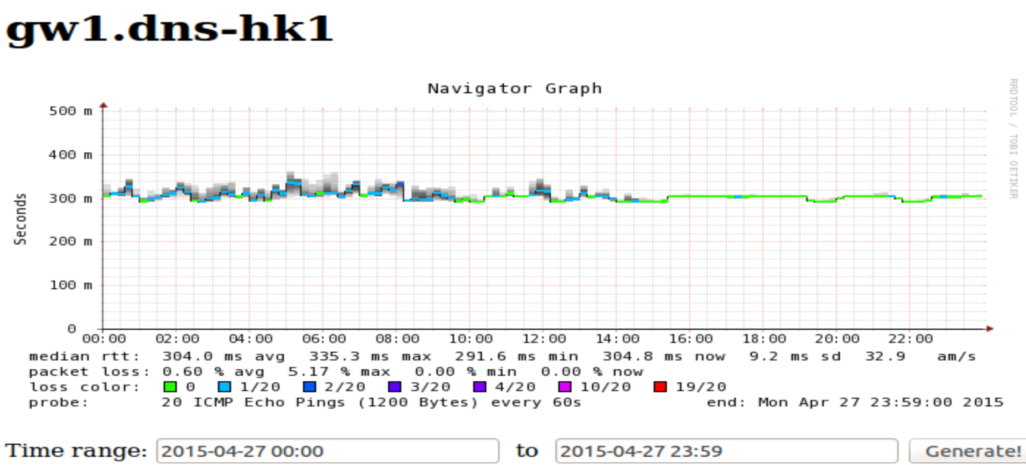


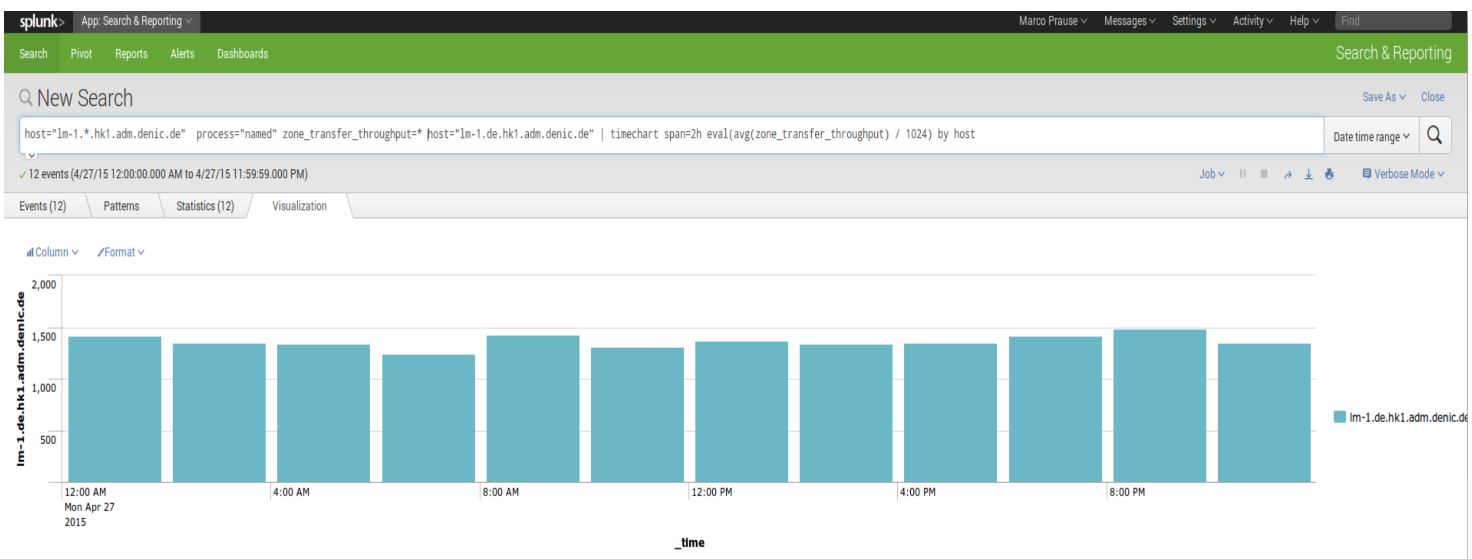




4. Changing the algorithm – changing the game ?

- Zonentransfer-times & lossrate (location Hongkong)





Thanks !

Questions ?

Marco Prause <prause@denic.de>