Next Level Ping Nerdery

Netzwerk-Monitoring mit meshping

github.com/Svedrin/meshping

Michael Ziegler

Chemnitzer Linuxtage

23.03.2025

Vorgeschichte

- heutzutage IT-Leiter Rechenzentrum bei einem regionalen Cloud-Provider in Osthessen
- arbeite seit 2011 als Systemadministrator von Virtualisierungsclustern

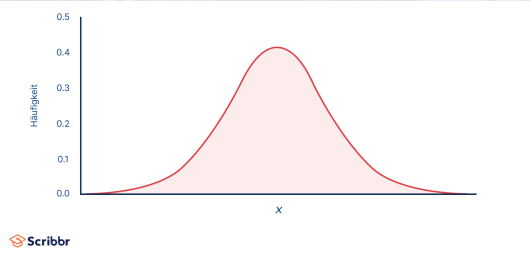
  (bevorzugt auf Linux-Basis)
- Monitoring als großes Steckenpferd – privat Munin, im Job damals Nagios
- Fokus auf Storage-Systeme und Disk-Latenzen
- "Kann ein Monitoring-System nicht von selbst lernen was normal ist und was nicht?"

- August 2013: Side-Project namens Fluxmon begonnen um dieser Frage nachzugehen
- Habe viel zum Thema Mittelwert, Standardabweichung und Konfidenzintervallen recherchiert (ich hatte Mathe-Grundkurs… aber Statistik mochte ich)
- 21.03.2015: CLT-Vortrag “Daten superschnell analysieren” von Hans Schönig gab mir das Datenbank-Handwerkszeug für meine Berechnungen
- auf keinen grünen Zweig gekommen :(
- es gab schlicht kein Konfidenzintervall, an das die Daten sich gehalten haben
- oder es war so groß dass ich keine Alerts mehr bekommen habe

- YouTube: Theo Schlossnagle - “The Math Behind Bad Behavior”
- Theo kennt sich mit Mathe tatsächlich *aus* :)
- er hat dasselbe versucht und weiß warum es nicht klappt

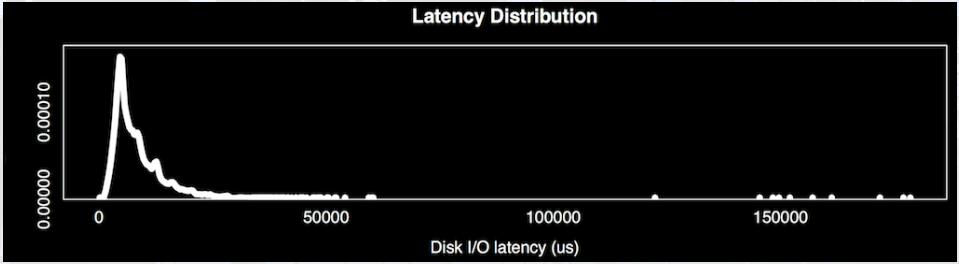
"All the statistics you're trying to apply – doesn't apply."

- Monitoring-Tools geben fast immer Mittelwerte aus
- Mittelwert und Standardabweichung sind nur auf normalverteilte Daten anwendbar
- Normalverteilt heißt so:

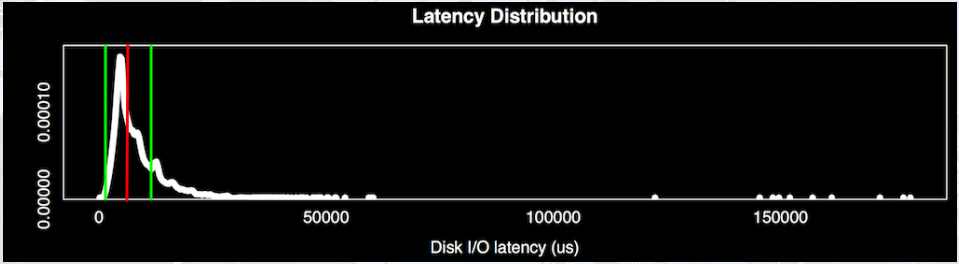


- man *denkt* intuitiv, Latenzen wären das - sind sie aber nicht

- Brendan Gregg (Kernel-Entwickler): Frequency trails
- Latenzen sind eher so verteilt:

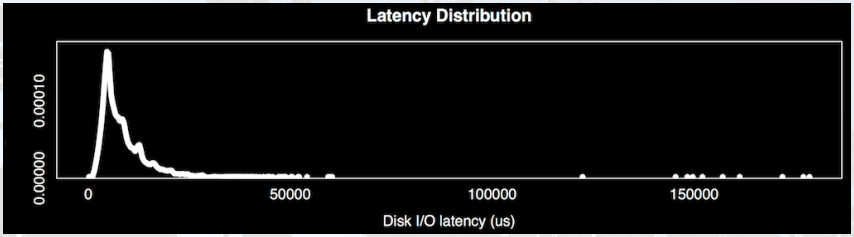


- “intuitive Erwartung” für Mittelwert und Standardabweichung:

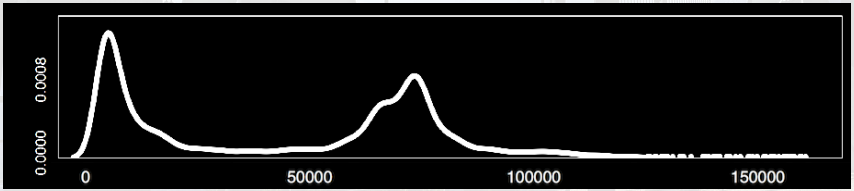


- passt offensichtlich nicht zur Verteilung
- es gibt kein aussagekräftiges Konfidenzintervall
- aber es gibt was besseres

- Modality Detection. Wenn ein Graph, der immer so aussah:



- plötzlich so aussieht:



- dann ist was faul. (Vielleicht.)

September 2015: meshping

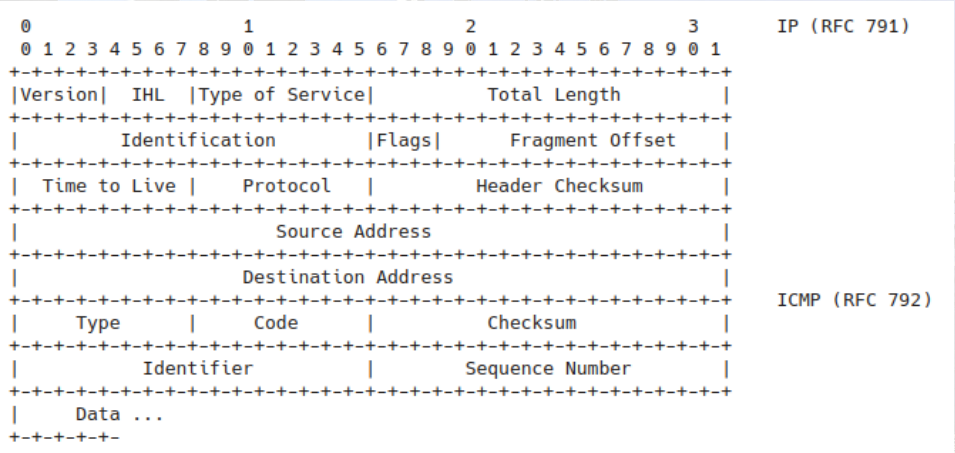
- Ich betreibe privat ein kleines verteiltes Netzwerk, damals zwei Server und drei Standorte über VPNs verbunden
- ich wollte ein Monitoring, ob alles erreichbar ist
- Anforderungen:
  - einfach, modernes UI (mobile-friendly)
  - schlank $\rightarrow$ geringe Hardware-Anforderungen
  - keine Mittelwerte

- Ihr alle kennt wahrscheinlich Ping:

```
# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=60 time=13.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=60 time=13.4 ms
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 13.398/13.498/13.599/0.100 ms
```

- prüft ob ein Host erreichbar ist, und mit welcher Latenz

- Internet Control Message Protocol Type 8/0



- ich wollte ein Tool haben, das das für viele Hosts effizient kann

- habe eine Library namens oping gefunden
- diese erlaubt viele Ziele mit einem einzelnen Aufruf anzupingen
- nettes Tool namens `noping` (aka “ncurses oping”):

```
56 bytes from 8.8.8.8 (8.8.8.8): icmp_seq=80 ttl=60 time=26,62 ms
56 bytes from 8.8.4.4 (8.8.4.4): icmp_seq=80 ttl=60 time=28,46 ms
56 bytes from 1.1.1.1 (1.1.1.1): icmp_seq=80 ttl=56 time=39,83 ms
56 bytes from 9.9.9.9 (9.9.9.9): icmp_seq=80 ttl=60 time=28,68 ms
```

8.8.8.8 ping statistics
80 packets transmitted, 65 received, 18,75% packet loss, time 2108,6ms
RTT[ms]: min = 27, median = 29, p(95) = 41, max = 196

8.8.4.4 ping statistics
80 packets transmitted, 69 received, 13,75% packet loss, time 2343,5ms
RTT[ms]: min = 27, median = 30, p(95) = 44, max = 196

1.1.1.1 ping statistics
80 packets transmitted, 80 received, 0,00% packet loss, time 3693,9ms
RTT[ms]: min = 40, median = 42, p(95) = 57, max = 196

9.9.9.9 ping statistics
80 packets transmitted, 80 received, 0,00% packet loss, time 2754,4ms
RTT[ms]: min = 28, median = 31, p(95) = 42, max = 196

Meshping: meshping

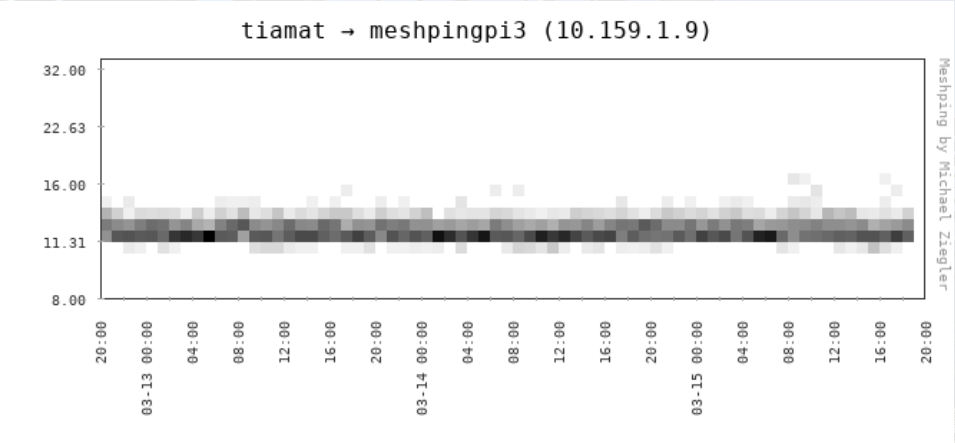
Map | Metrics | Clear statistics | Compare

Search | Name or IP

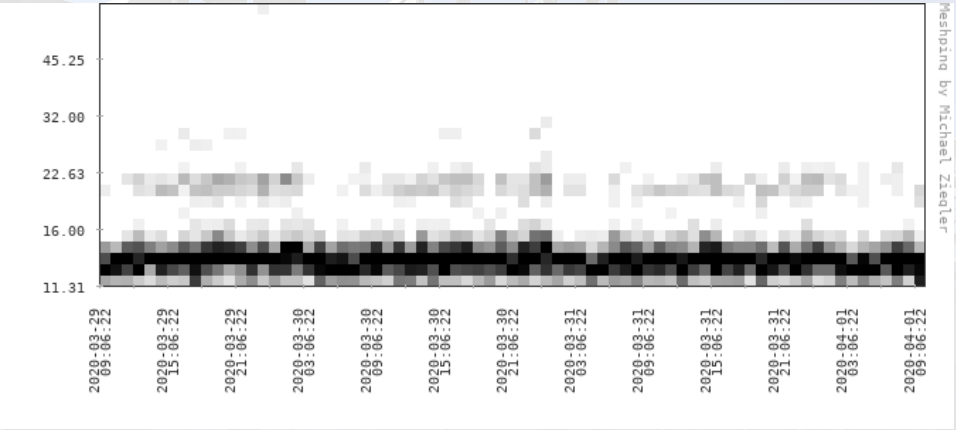
	Target	Address	Sent	Recv	Succ	Loss	Min	Avg15m	Avg6h	Avg24h	Max	Last	
✓	1.0.0.1	1.0.0.1	11614	11601	99.89	0.11	3.31	25.51	27.92	27.63	4194.11	17.50	
↗	1.1.1.1	1.1.1.1	11614	10888	93.75	6.25	3.00	25.32	25.27	23.94	2463.50	17.04	
✓	dns.google	8.8.4.4	11614	11588	99.78	0.22	2.55	18.17	20.80	21.11	2043.20	13.99	
✓	dns.google	8.8.8.8	11614	11599	99.87	0.13	2.32	22.39	24.53	23.48	2352.95	16.01	
✓	dns.google	2001:4860:4860::8844	11614	10815	93.12	6.88	9.21	16.65	19.51	19.67	1939.59	13.26	
✓	dns.google	2001:4860:4860::8888	11614	10825	93.21	6.79	7.81	20.60	21.62	20.09	1223.02	14.74	

example.com | 1.2.3.4 (optiona | +

- Graphen: Darstellung als Heatmap. Jede Spalte entspricht einer Stunde

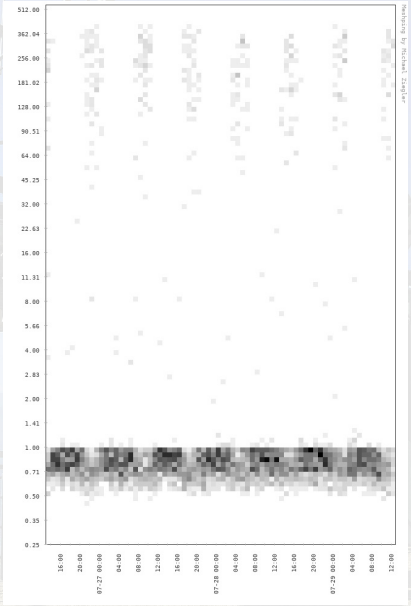


- bei meinem ISP sah es damals so aus:

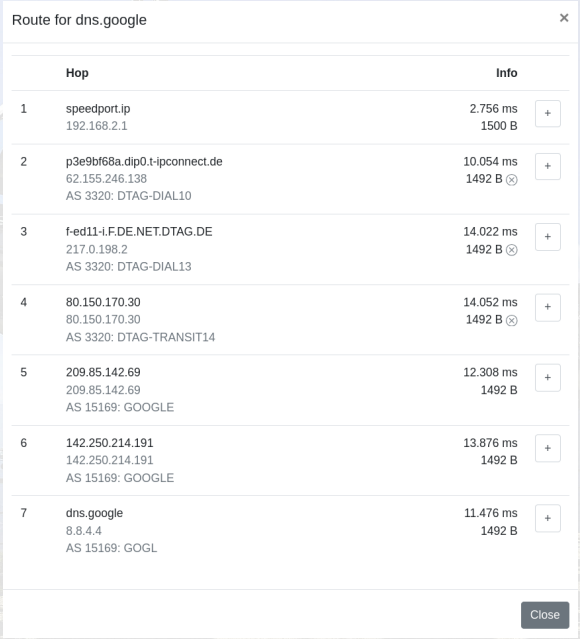


- vermutlich Load-Balancing über zwei Router

- bei einem Kunden im RZ:
- die Firewall hatte ein Problem und hat Pakete verzögert
- Mittelwert war 7ms – eigentlich unauffällig
- Kunde hatte ständig RDP-Verbindungsabbrüche und konnte kaum arbeiten
- der Graph verdeutlicht, dass Handlungsbedarf besteht



- welchen Weg nehmen unsere Pakete?

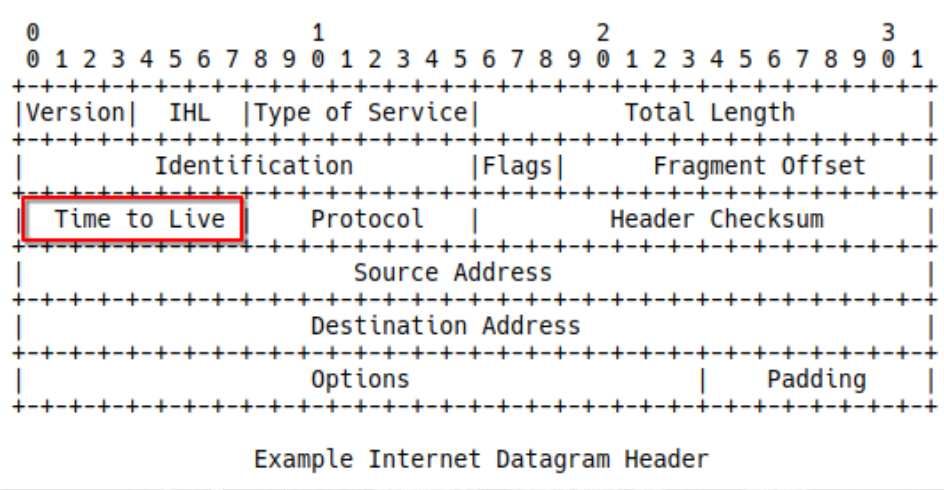


- woher weiß meshping das? → traceroute (oder tracert)

```
# traceroute 8.8.4.4
traceroute to 8.8.4.4 (8.8.4.4), 30 hops max, 60 byte packets
1  192.168.2.1      2.982 ms   2.741 ms   2.686 ms
2  62.155.246.138  11.318 ms  11.585 ms  11.625 ms
3  217.0.198.10    15.629 ms  15.524 ms  15.419 ms
4  80.150.170.30   14.512 ms  15.207 ms  15.106 ms
5  * * *
6  8.8.4.4         12.011 ms  11.719 ms  10.994 ms
```

- woher weiß traceroute das?

- im IP-Header (RFC 791):



- traceroute kann man mittels ping simulieren:

```
# ping 8.8.4.4 -t 4
PING 8.8.4.4 (8.8.4.4) 56(84) bytes of data.
From 80.150.170.30 icmp_seq=1 Time to live exceeded
From 80.150.170.30 icmp_seq=2 Time to live exceeded
^C
--- 8.8.4.4 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss
```

4	80.150.170.30 80.150.170.30 AS 3320: DTAG-TRANSIT14	

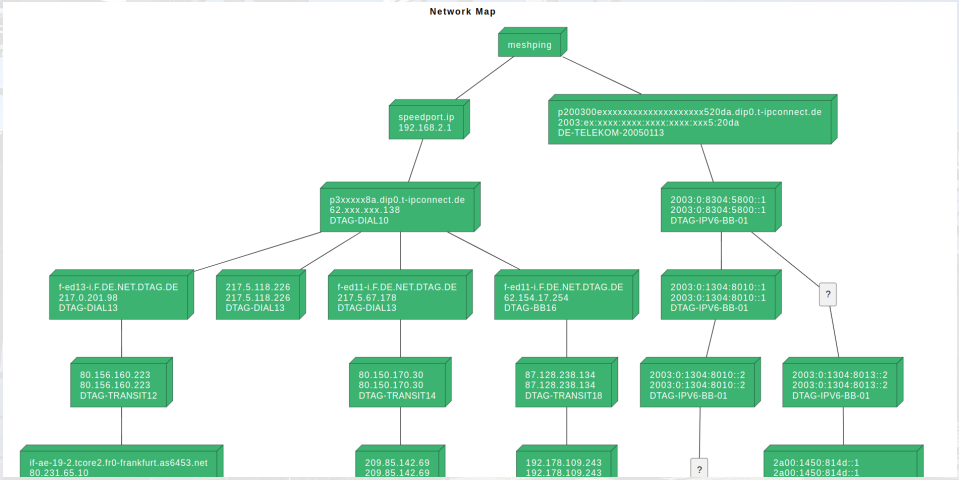
- wiederholen mit `-t 5`, `-t 6`, `-t 7` etc, bis eine Antwort kommt

- meshping zeigt bei der Route auch die AS-Nummer an
- AS = Autonomes System - sozusagen ein Provider
- wird aus whois-Datenbank abgefragt

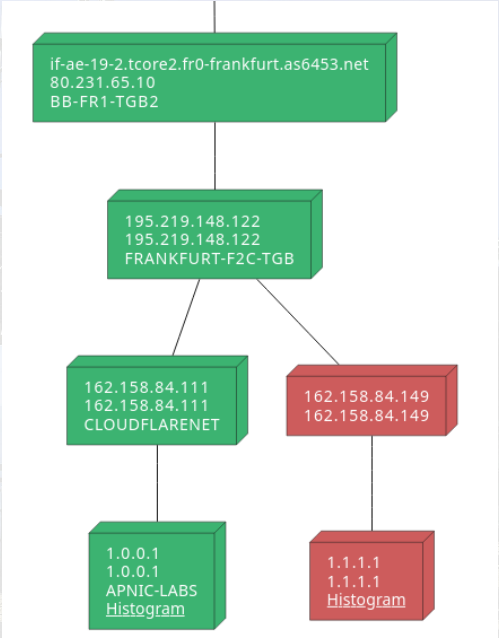
Route for dns.google

	Hop
1	192.168.2.1 192.168.2.1
2	p3e9bf68a.dip0.t-ipconnect.de 62.155.246.138 AS 3320: DTAG-DIAL10
3	f-ed11-i.F.DE.NET.DTAG.DE 217.0.198.2 AS 3320: DTAG-DIAL13
4	80.150.170.30 80.150.170.30 AS 3320: DTAG-TRANSIT14
5	209.85.142.69 209.85.142.69 AS 15169: GOOGLE
6	142.250.214.191 142.250.214.191 AS 15169: GOOGLE
7	dns.google 8.8.4.4 AS 15169: GOGL

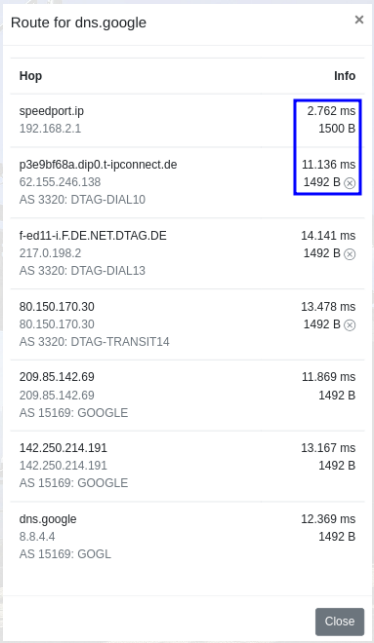
- Alle traceroutes zusammen ergeben eine Karte des Netzwerks:



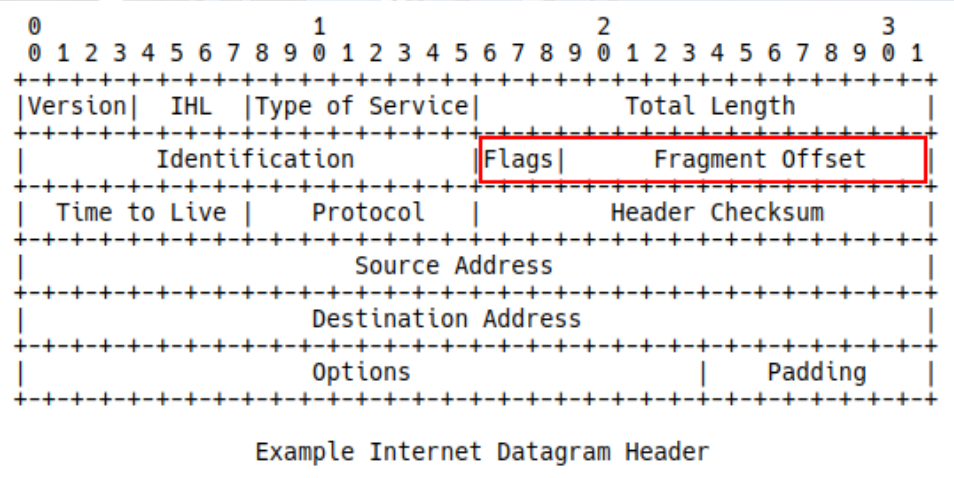
- wenn ein Ziel down geht, zeigt die Karte an, ab welchem Hop



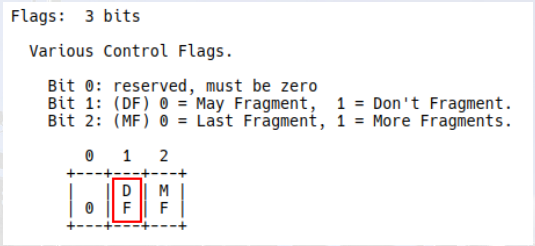
- die Route zeigt auch die MTU an
- MTU = Maximum Transfer Unit
- Maximal erlaubte Paketgröße
- beeinflusst durch DSL und VPNs
- kann Downloads oder RDP-Verbindungen stören



- woher weiß meshping das? → Path MTU Discovery. IP-Header:



- Flag "don't fragment" verbietet Fragmentierung
- Ping kann ein beliebig großes Datensegment mitsenden
- wenn wir das Paket so groß machen wie die MTU, und ein Router es fragmentieren muss, bekommen wir eine Nachricht:




```
# ping 8.8.4.4 -M probe -s $((1500-28))
PING 8.8.4.4 (8.8.4.4) 1472(1500) bytes of data.
From 192.168.2.1 icmp_seq=1 Frag needed and DF set (mtu = 1492)

# ip route show cache 8.8.4.4
8.8.4.4 via 192.168.2.1 dev wlp4s0
  cache expires 523sec mtu 1492
```

Geht auch unter Windows:

```
PS C:\> ping -f -l (1500-28) 8.8.4.4

Ping wird ausgeführt für 8.8.4.4 mit 1472 Bytes Daten:
Antwort von 192.168.2.1: Paket müsste fragmentiert werden, DF-Flag ist
                                                          jedoch gesetzt.

PS C:\> netsh interface ipv4 show destinationcache address=8.8.4.4
Ziel      : 8.8.4.4
Pfad-MTU  : 1492
```

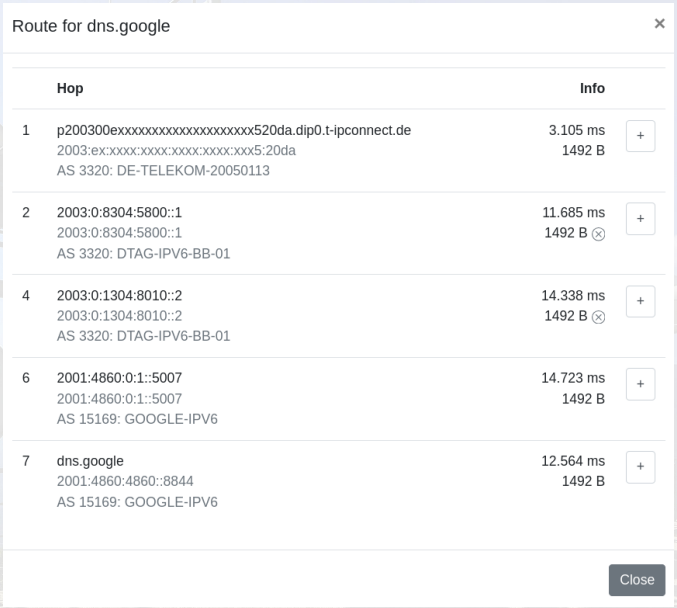
Weitere Features

- Vergleichsfunktion für bis zu drei Targets im selben Graphen:

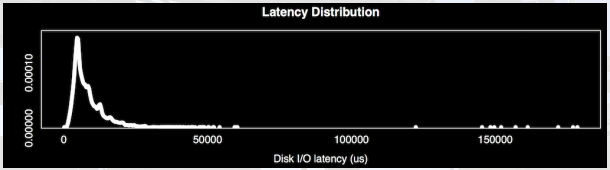
- Integrierter Prometheus-Endpoint (unterstützt natürlich Histogramme):

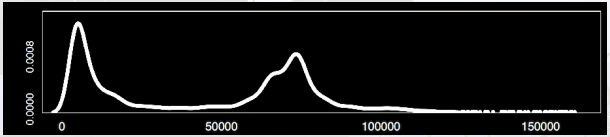
```
# curl http://localhost:9922/metrics
meshping_sent{name="dns.google",target="8.8.4.4"} 11794
meshping_recv{name="dns.google",target="8.8.4.4"} 11768
meshping_lost{name="dns.google",target="8.8.4.4"} 26
meshping_max{name="dns.google",target="8.8.4.4"} 2043.20
meshping_min{name="dns.google",target="8.8.4.4"} 2.55
meshping_pings_sum{name="dns.google",target="8.8.4.4"} 232453.875000
meshping_pings_count{name="dns.google",target="8.8.4.4"} 11768
meshping_pings_bucket{name="dns.google",target="8.8.4.4",le="12.13"} 1
meshping_pings_bucket{name="dns.google",target="8.8.4.4",le="13.00"} 13
meshping_pings_bucket{name="dns.google",target="8.8.4.4",le="13.93"} 29
meshping_pings_bucket{name="dns.google",target="8.8.4.4",le="14.93"} 34
meshping_pings_bucket{name="dns.google",target="8.8.4.4",le="16.00"} 38
meshping_pings_bucket{name="dns.google",target="8.8.4.4",le="17.15"} 39
meshping_pings_bucket{name="dns.google",target="8.8.4.4",le="18.38"} 43
```

- IPv6 ist voll unterstützt

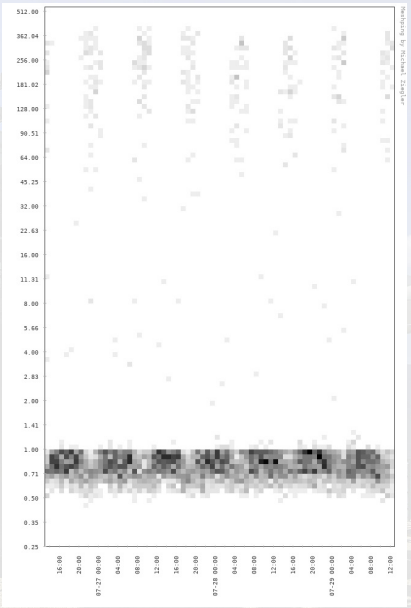


- Modality Detection: Schwieriges Thema

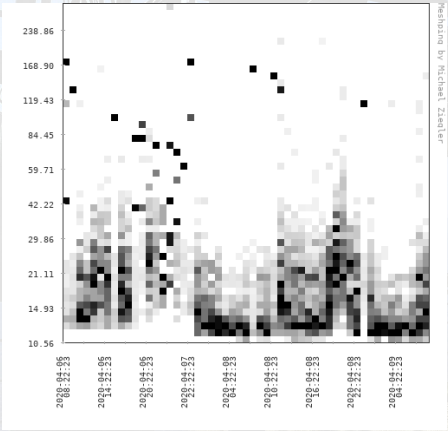


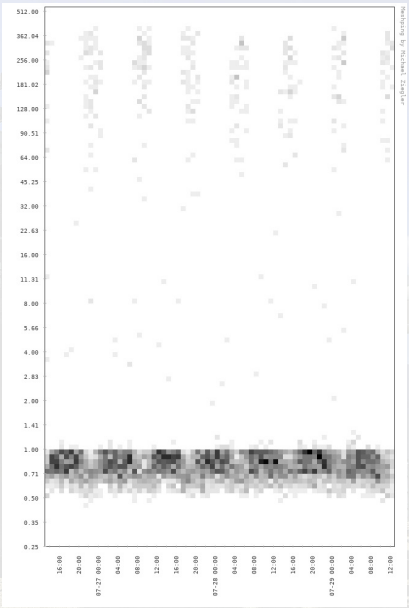


- könnte in diesem Fall helfen $\rightarrow$



- aber wie unterscheidet sie diese Fälle?
- Rechts RZ, Links WLAN über ca. 5km
- Sieht für die MD gleich aus, aber der Mensch weiß es besser – hohes Potenzial für Fehlalarme

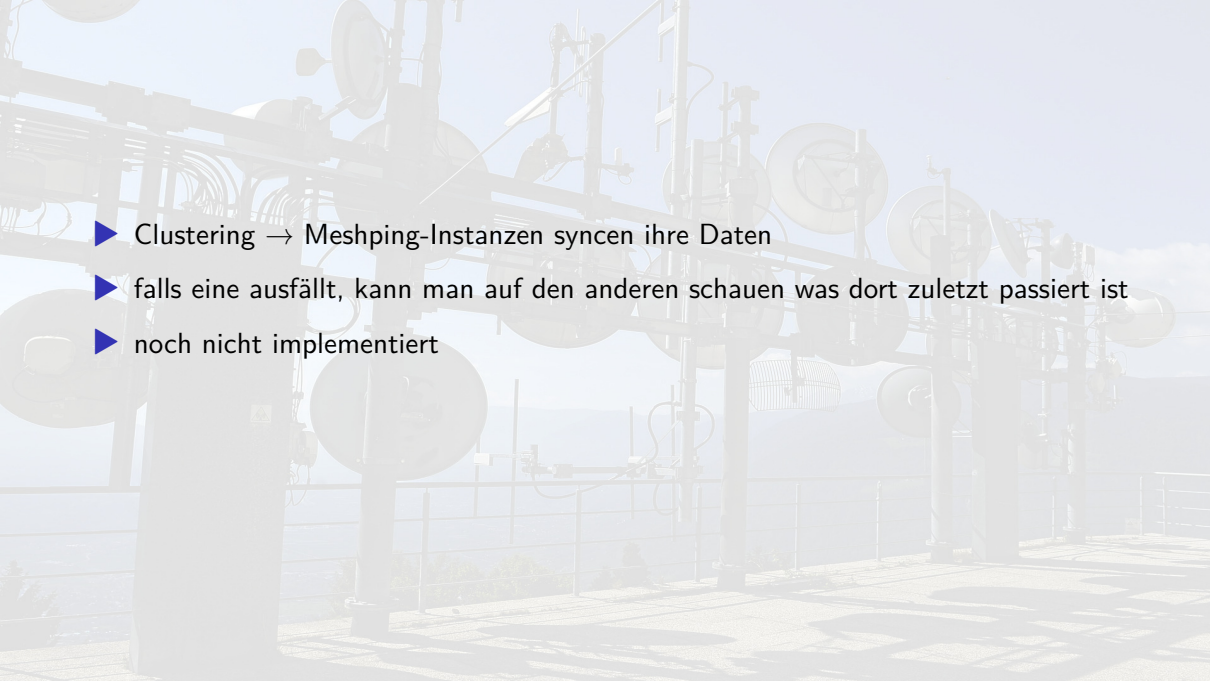




- Modality Detection aktuell nicht implementiert
- meshping an Prometheus anbinden, wichtige Links auf diese Weise monitoren
- Abfrage:

```
histogram_quantile(
  0.95,
  rate(meshping_pings_bucket{instance="RZ",target="8.8.4.4"}[5m])
)
```

- ergibt die Zeit, in der 95% der Pings beantwortet werden
- wenn >5ms, Alarm



- Clustering $\rightarrow$ Meshping-Instanzen syncen ihre Daten
- falls eine ausfällt, kann man auf den anderen schauen was dort zuletzt passiert ist
- noch nicht implementiert

- Installation:

```
docker run -d --name meshping \
  --net host \
  -v meshping-data:/opt/meshping/db \
  svedrin/meshping:latest
```

```
http://localhost:9922
```

- Linux amd64, armv7l, arm64
- Windows mit WSL2 und Docker Desktop
- Wichtig: Traceroute-Features laufen nur alle halbe Stunde

Fragen?

github.com/Svedrin/meshping

Hintergrundbild: pixabay – Walter Wiedenhofer, Südtirol

- kennt ihr schon IPv2?

```
# ping 1.1
PING 1.1 (1.0.0.1) 56(84) bytes of data.
64 bytes from 1.0.0.1: icmp_seq=1 ttl=59 time=14.7 ms
^C
--- 1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.665/14.665/14.665/0.000 ms
```

Vielen Dank!

github.com/Svedrin/meshping