

Black-box monitoring on steroids

Giedrius Stasiulionis

White-box vs. Black-box monitoring

- White-box is when you can collect all the internal states and data of the monitored object
- Black-box is knowing nothing about the internals of monitored object (as a user would see it*)

* <https://sre.google/sre-book/monitoring-distributed-systems/>

Black-box monitoring

- You know nothing more but how to reach it as a user (e.g. URL of a website)
- Alchemy: turning nothing into something

What is typically monitored?

- HTTP response code
- Response time

How is it typically monitored?

- From some static datacenter probe
- Worst case – from same infra as service itself is hosted (no real-world network included into monitoring scope)

What will we monitor?

- HTTP response code
- Response time (TTFB)
- Number of retries
- Content / response headers length
- ICMP ping (min / max / avg / loss)
- Output of “mtr”
- Screenshots of website
- SSL cert expiry

How will we monitor it?

- **Combine data center-based checks with residential proxy-based checks**

How will we monitor it?

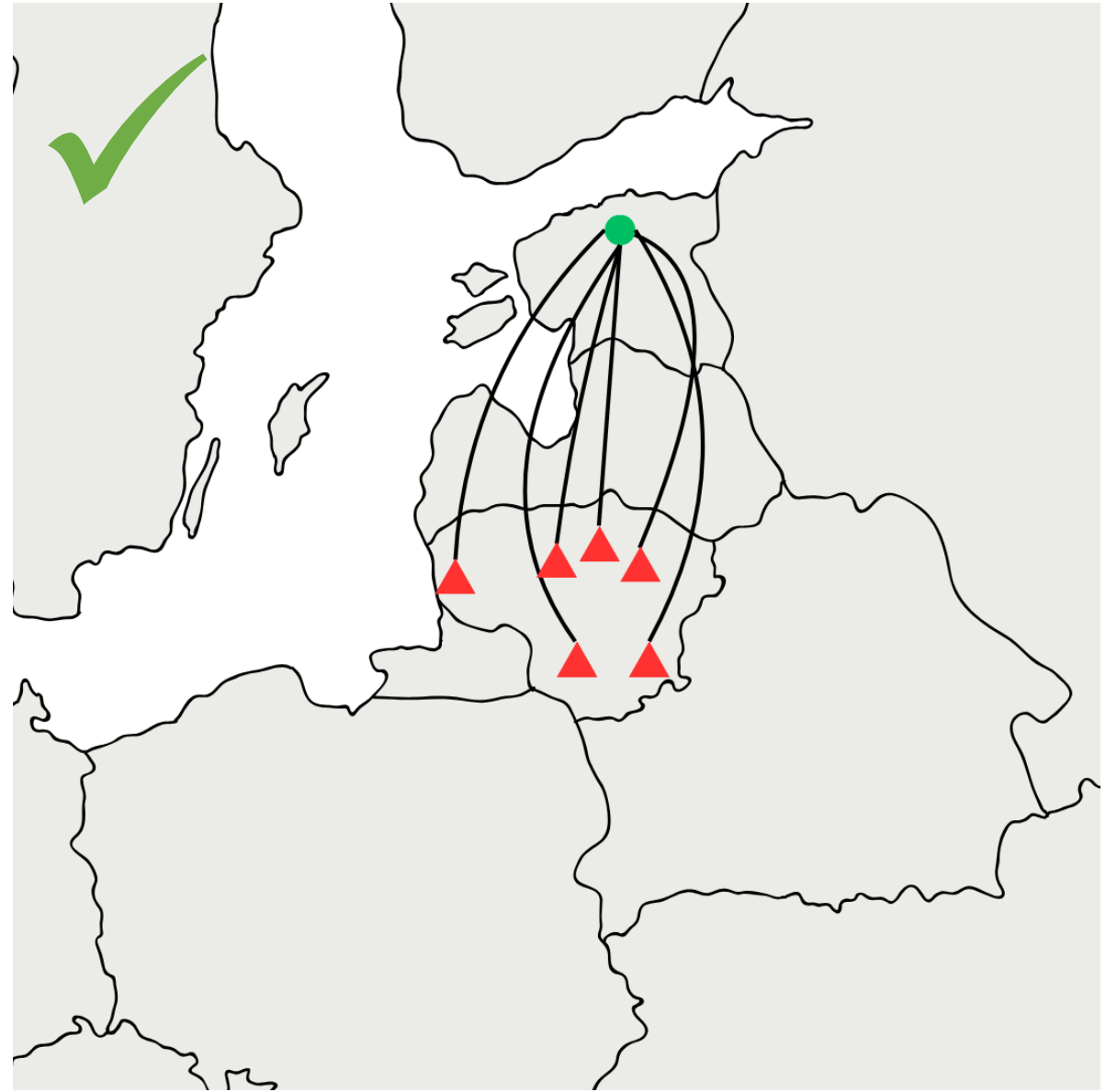
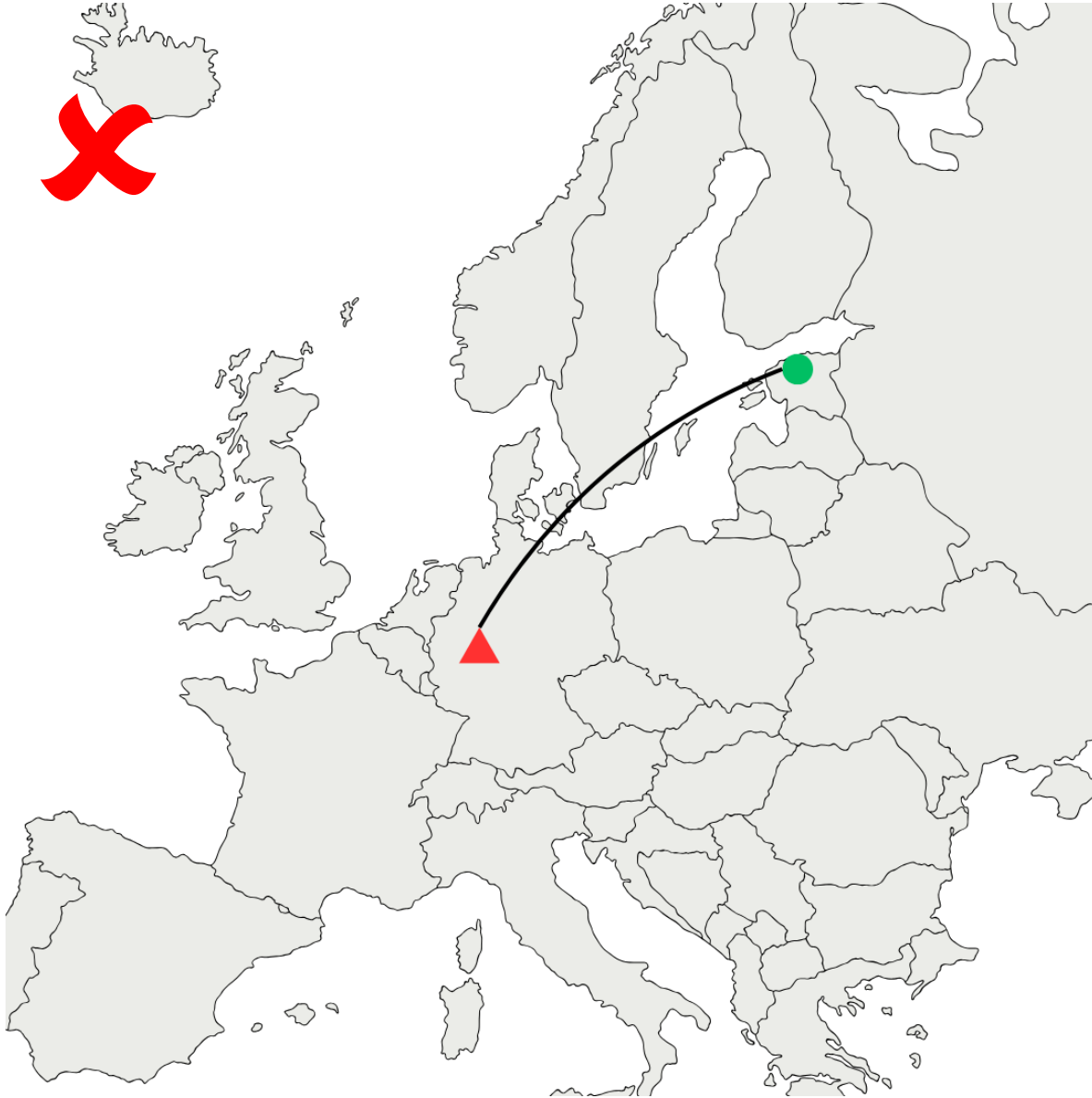
Property	Data center	Residential proxies
HTTP response code	✓	✓
Response time (TTFB)	✓	✓
Number of retries	✓	✓
Content / response length	✓	✓
ICPM ping (min / max / avg / loss)	✓	
Output of “mtr”	✓	
Screenshots of website	✓	
SSL cert expiry	✓	

How will we monitor it?

- Geographical awareness:
 - Datacenter probe located in same country(ies) as our target audience
 - For HTTP checks – using rotating residential proxies from same location as our target audience

Why geography and residential proxies matter?

- Say 90% of real visitors are from Lithuania
- We should not care if page is or is not reached (and how fast) from some static datacenter in Germany (or even worse, something more remote)
- We should care that real users from Lithuania can reach the page



Zabbix native features for datacenter checks

- Web scenario
- HTTP agent item
- ICMP ping items
- net.tcp* items
- Browser item

And of course...

- UserParameter feature – to tweak everything needed

For example...

With web scenario we can't:

- Implement dynamic timeout value for each retry / exponential backoff strategy
- Monitor number of retries
- Measure TTFB when proxies are used ($\text{time_starttransfer} - \text{time_pretransfer}$)
- Store values in Zabbix at exact same intervals between them if it takes long time to retrieve it (because timestamp is of “finish”, not “start”)

UserParameter

```
run_request() {  
    local timeout=$1  
  
    curl \  
        --proxy-header "Cache-Control: no-cache" \  
        -H "Cache-Control: no-cache" \  
        -L -v -s -S -m ${timeout} -I -X HEAD \  
        --no-keepalive \  
        -w '$Time total: %{time_total} seconds\nTime connect: %{time_connect} seconds\nTime pretransfer: %  
{time_pretransfer} seconds\nTime starttransfer: %{time_starttransfer} seconds\n' \  
        -x "http://${proxy_user}:${proxy_pass}_country-${proxy_country}_streaming-1@${proxy_host}:${proxy_port}" \  
        "${url}" 2>&1  
}
```

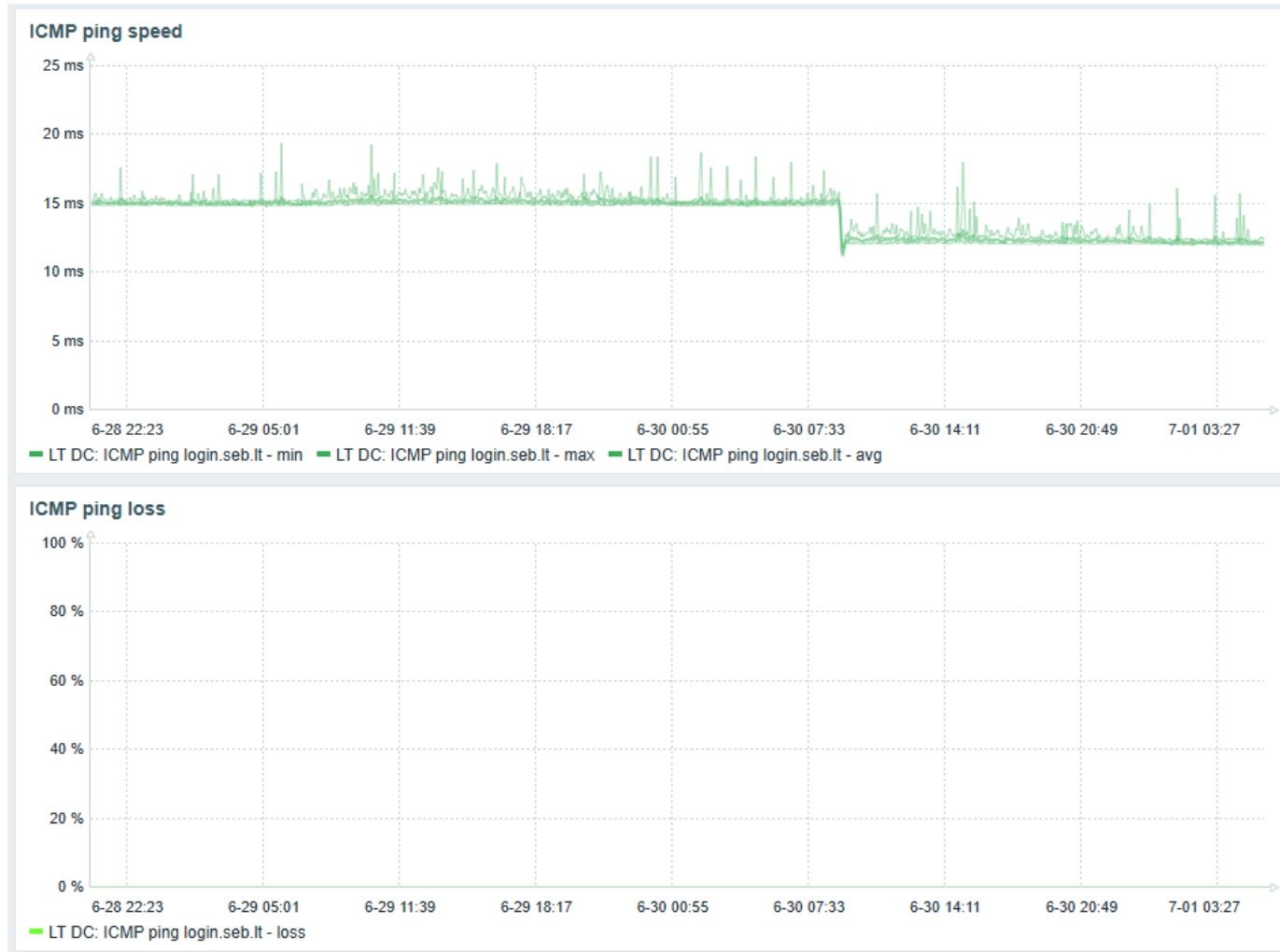
ICMP ping

- One master item and parse min / max / avg / loss from it



```
UserParameter=fping[*],fping -s -C 10 -t 100 -p 200 "$1"
```

ICMP ping



mtr

- This command combines ping and traceroute – we can see how do we travel from source to destination in details



```
UserParameter=mtr.raw.json[*],mtr -bw -r -c 5 -4 -T -P 443 -j "$1"
```

mtr

- LLD applied to be able to extract info (latency, loss) about each hop from source to destination
- Thanks to the concept of dependent items, mtr needs to be run only once – both LLD and items are dependent on it

mtr

Item prototype Tags 2 Preprocessing 1

Preprocessing steps ?

	Name	Parameters	Custom on fail
1:	JSONPath	<code>\$.report.hubs[?(@.host=="{#HOST}" && @.count=="{#HOP}")].Avg.first()</code>	☒
	Custom on fail	Discard value	

[Add](#)

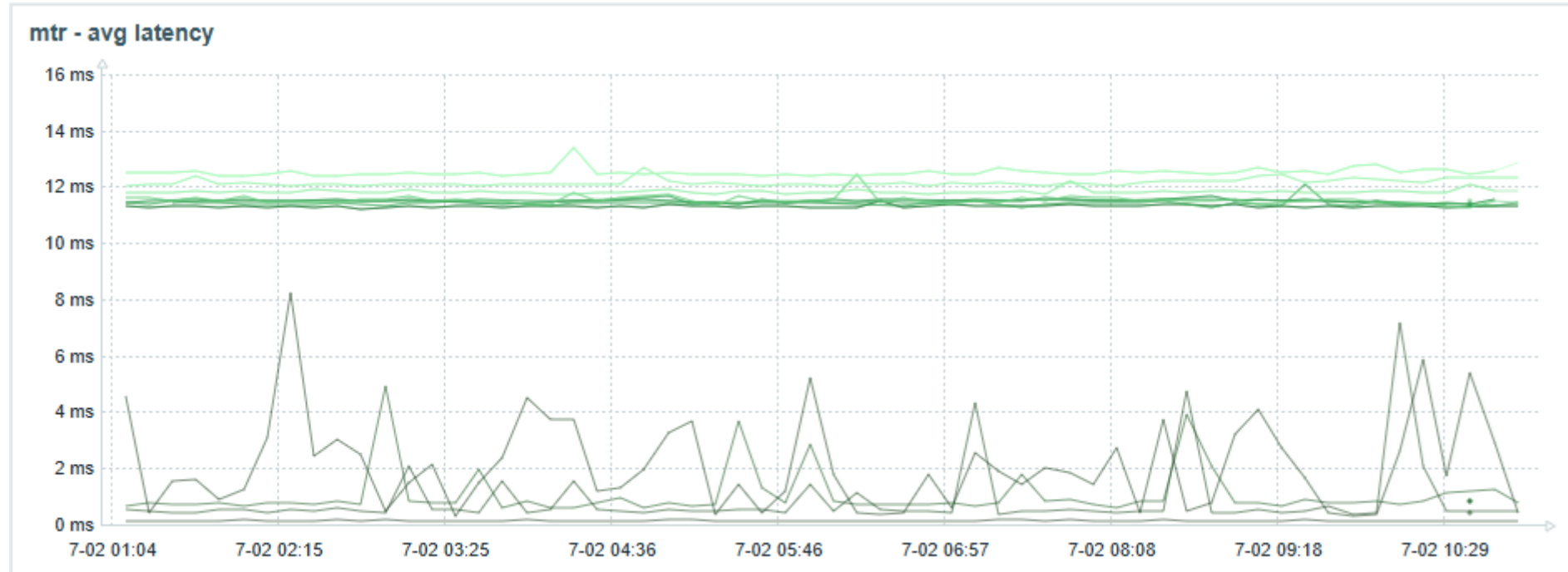
```
...  
  
{  
  "count": "14",  
  "host": "login.seb.lt (78.24.199.35)",  
  "Loss%": 0.0,  
  "Snt": 4,  
  "Last": 12.819,  
  "Avg": 12.744,  
  "Best": 12.56,  
  "Wrst": 12.955,  
  "StDev": 0.176  
}
```

mtr

2026-07-02 15:40:43

- LT DC: login.seb.lt - 01 - 169.254.0.1 - avg latency: 0.208 ms
- LT DC: login.seb.lt - 02 - 31.14.176.255 - avg latency: 0.515 ms
- LT DC: login.seb.lt - 03 - 81-16-238-52.datalogistics.lt (81.16.238.52) - avg latency: 0.592 ms
- LT DC: login.seb.lt - 04 - 185-84-46-76.datalogistics.lt (185.84.46.76) - avg latency: 0.778 ms
- LT DC: login.seb.lt - 05 - vlna00-agg-1.bundle-ether7.tele2.net (91.129.14.186) - avg latency: 11.475 ms
- LT DC: login.seb.lt - 06 - vlna00-agg-1.bundle-ether7.tele2.net (91.129.14.186) - avg latency: 11.493 ms
- LT DC: login.seb.lt - 07 - vlna00-core-1.bundle-ether111.tele2.net (130.244.130.22) - avg latency: 11.572 ms
- LT DC: login.seb.lt - 08 - ada345-bcore-1.bundle-ether2.tele2.net (130.244.130.2) - avg latency: 11.478 ms
- LT DC: login.seb.lt - 09 - ada345-bcore-1.bundle-ether2.tele2.net (130.244.130.2) - avg latency: 11.63 ms
- LT DC: login.seb.lt - 10 - mcf588-sn-1.bundle-ether1.tele2.net (212.151.28.135) - avg latency: 11.423 ms
- LT DC: login.seb.lt - 11 - static-212-107-37-29.cust.tele2.ee (212.107.37.29) - avg latency: 11.916 ms
- LT DC: login.seb.lt - 12 - jmt-g003.seb.ee (78.24.192.18) - avg latency: 12.458 ms
- LT DC: login.seb.lt - 13 - jmt-g003.seb.ee (78.24.192.18) - avg latency: 12.456 ms
- LT DC: login.seb.lt - 14 - login.seb.lt (78.24.199.35) - avg latency: 12.599 ms

mtr



SSL certificate monitoring



```
expires=$(echo | timeout 5 openssl s_client -connect $1:$2 -servername $1 2>/dev/null | timeout 5 openssl x509 -noout  
-dates 2>/dev/null | grep "After")  
  
[[ $(echo ${expires#notAfter=*}) != "" ]] && echo $(((date -d "${expires#notAfter=*}" +%s)-(date +%s))/86400)) ||  
echo "-1"
```

SSL certificate monitoring

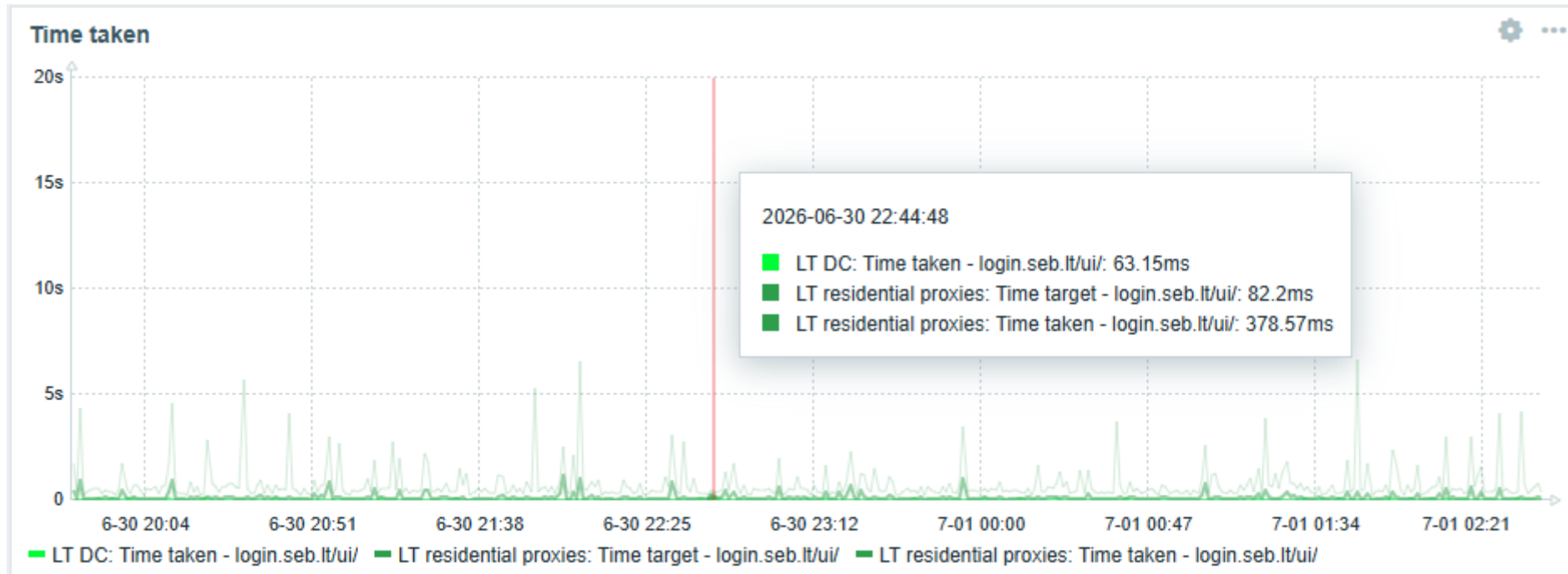


HTTP-related checks

- Since proxy traffic might become expensive, we perform HEAD instead GET
- From data center we can perform GET in parallel, if needed (and for example search for some string in return content)

HTTP-related checks

- Time to respond – in proxy case it is important to exclude time to connect to proxy itself



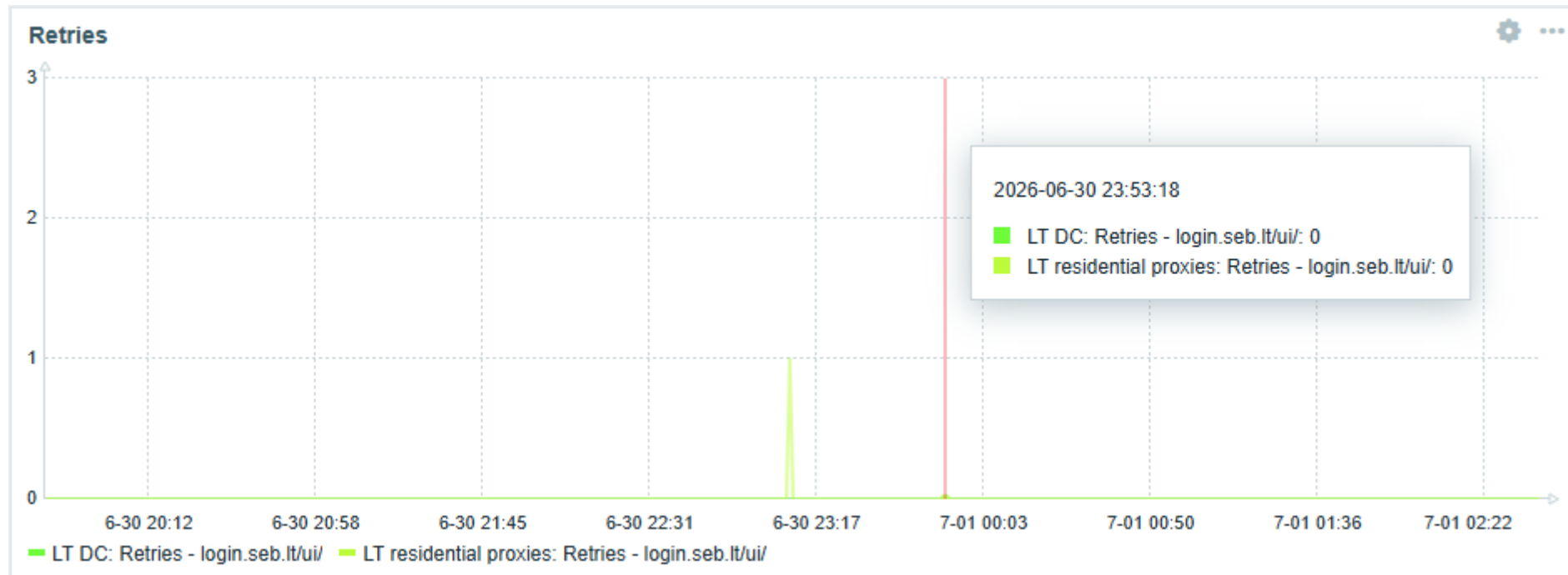
HTTP-related checks

- HTTP status code; in case of timeout – will drop to 0



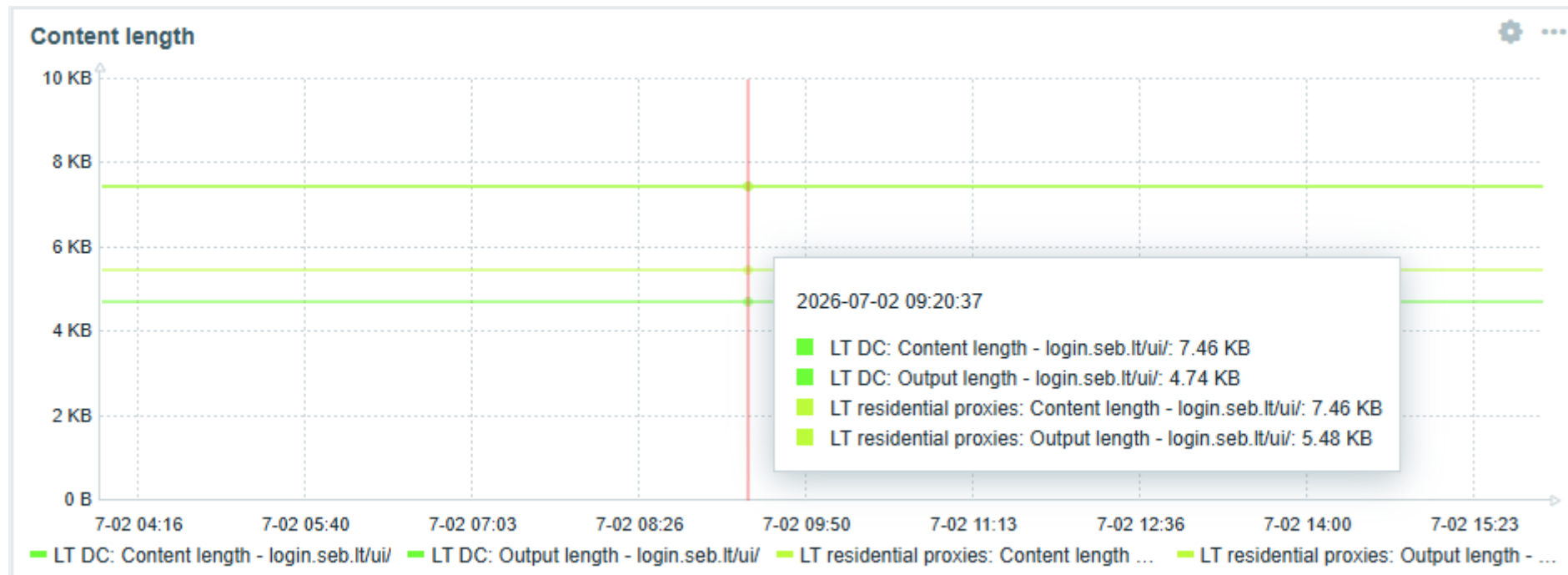
HTTP-related checks

- Number of retries – increasing / frequent retries indicate problem



HTTP-related checks

- Content length / response (full curl output) length – if changes, might be interesting, why



Browser item

- Allows you to create, store and display screenshots directly in Zabbix

Browser item

- However way of displaying is not entirely satisfying...

Item history		 
Name	Value	
login.seb.it: Website login.seb.it Get data		
login.seb.it: Website login.seb.it Get data		
login.seb.it: Website login.seb.it Get data		
login.seb.it: Website login.seb.it Get data		

Browser item

- So I created workaround to display it like:

Latest screenshot  

Pasirinkite atpažinimo priemonę

„Smart-ID“ „Mobile-ID“ SEB programėlė Generatorius

Atpažinimo kodas

Asmens kodas

☐ Įsiminti mane 
Kitų kartų nemokamai įvesti asmens kodą.
Pasirinkdami, jūs sutinkate su
funkciniais slapukais 

[Prijungti](#)

Naudinga

Finansinių sukčių prevencija: [kaip į jas neįkliūti?](#) 

Kaip mes tvarkome jūsų asmens duomenis? 

2026-07-02 12:45:09

Browser item

- Pros:
 - 100% width and height of widget
 - Incorporates smoothly into any dashboard
 - Includes timestamp when screenshot was created
- Cons:
 - It is a workaround

Browser item

- Custom endpoint
- Receives itemid as GET parameter



```
$itemid = $_GET['itemid'] ?? null;  
  
if ($itemid === null) {  
    exit(1);  
}
```

Browser item

- Queries Zabbix API (history.get) for latest timestamp (clock,ns) of this item

```
$params = [  
  'itemids'    => "$itemid",  
  'history'    => 5,  
  'output'     => ['clock', 'ns'],  
  'sortfield'  => 'clock',  
  'sortorder'  => 'DESC',  
  'limit'      => 1  
];  
  
$payload = [  
  'jsonrpc'    => '2.0',  
  'method'     => 'history.get',  
  'id'         => 1,  
  'params'     => $params  
];
```

Browser item

- Calls /zabbix/zabbix.php with action 'widget.itemhistory.image_value.get'



```
$query = http_build_query([
    'itemid'    => $itemid,
    'clock'     => $clock,
    'ns'        => $ns,
    'action'    => 'widget.itemhistory.image_value.get'
]);

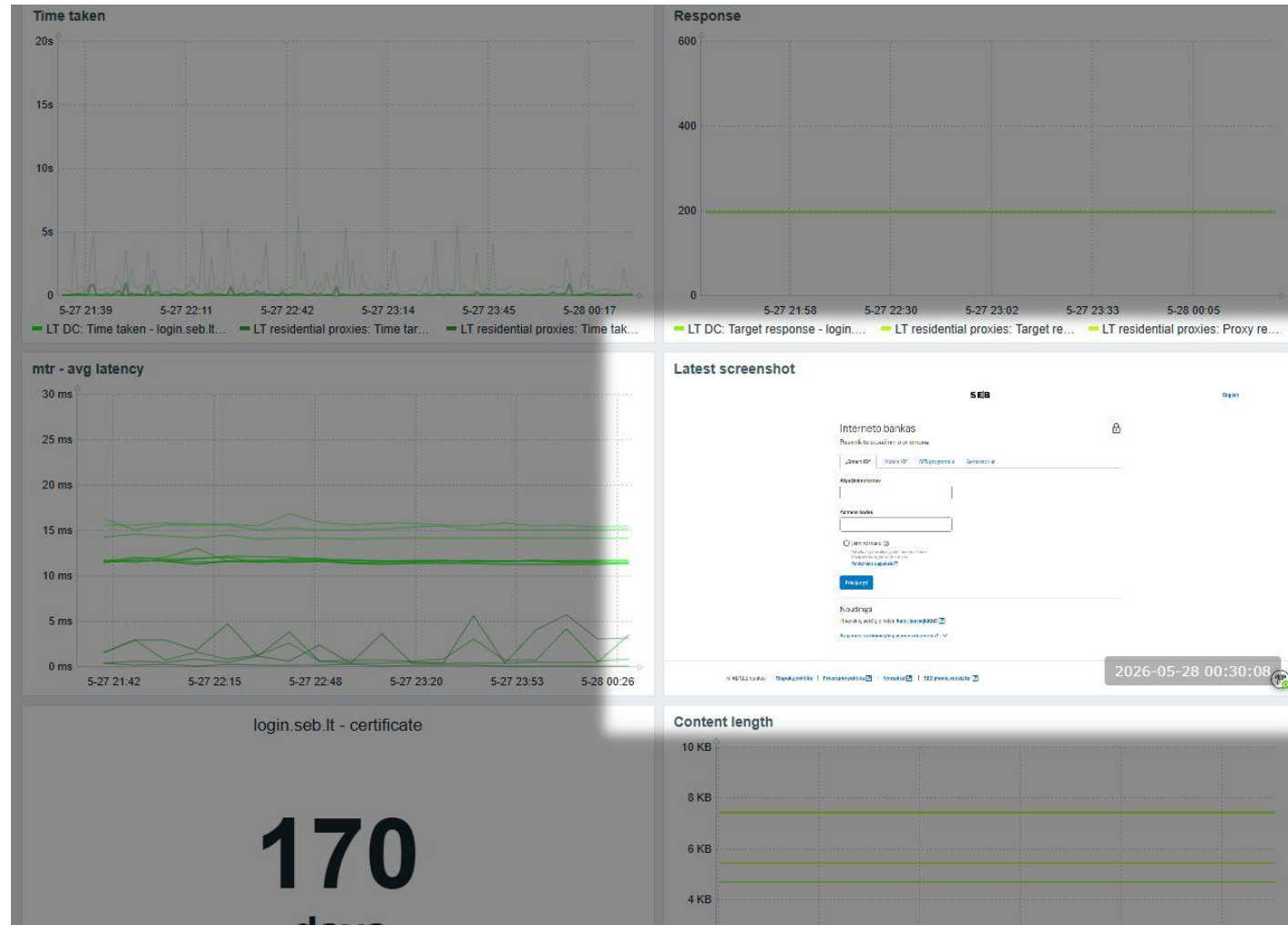
$url = "/zabbix/zabbix.php?$query";
```

Browser item

- Draws in full size the retrieved image – use it as URL widget

```
<html>
<head>
<meta charset="utf-8" />
<style type="text/css">
  html {
    background: url("<?= $url ?>") no-repeat center center fixed;
    -webkit-background-size: cover;
    -moz-background-size: cover;
    -o-background-size: cover;
    background-size: cover;
    font-family: Verdana, sans-serif;
    font-size: 48pt;
    color: #333;
  }
  .time-overlay {
    position: fixed;
    bottom: 10px;
    right: 15px;
    background: rgba(0,0,0,0.3);
    color: #fff;
    padding: 6px 10px;
    font-size: 14px;
    font-family: Verdana, sans-serif;
    border-radius: 4px;
  }
</style>
<title>Snapshot of <?php echo $itemid;?></title>
</head>
<body>
  <div class="time-overlay">
    <?= htmlspecialchars($time) ?>
  </div>
</body>
</html>
```

Browser item



Conclusion

Purpose of black-box monitoring is to tell that something is not working, not to tell why.

With Zabbix and all the presented techniques applied, you will have better chances to not miss those non-working moments.

Thanks!