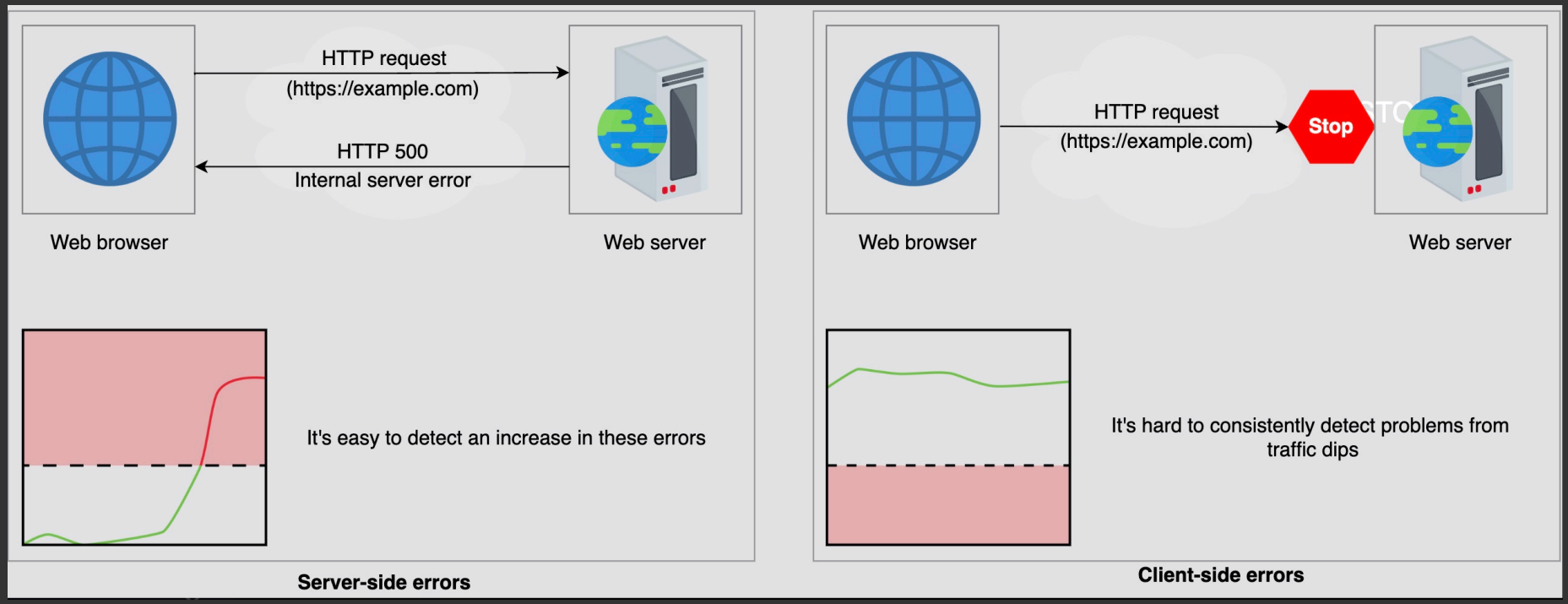
Client Side Errors

We can monitor our web and application servers' logs if a request fails to process. If multiple requests fail, we can observe a spike in internal errors (error 500).

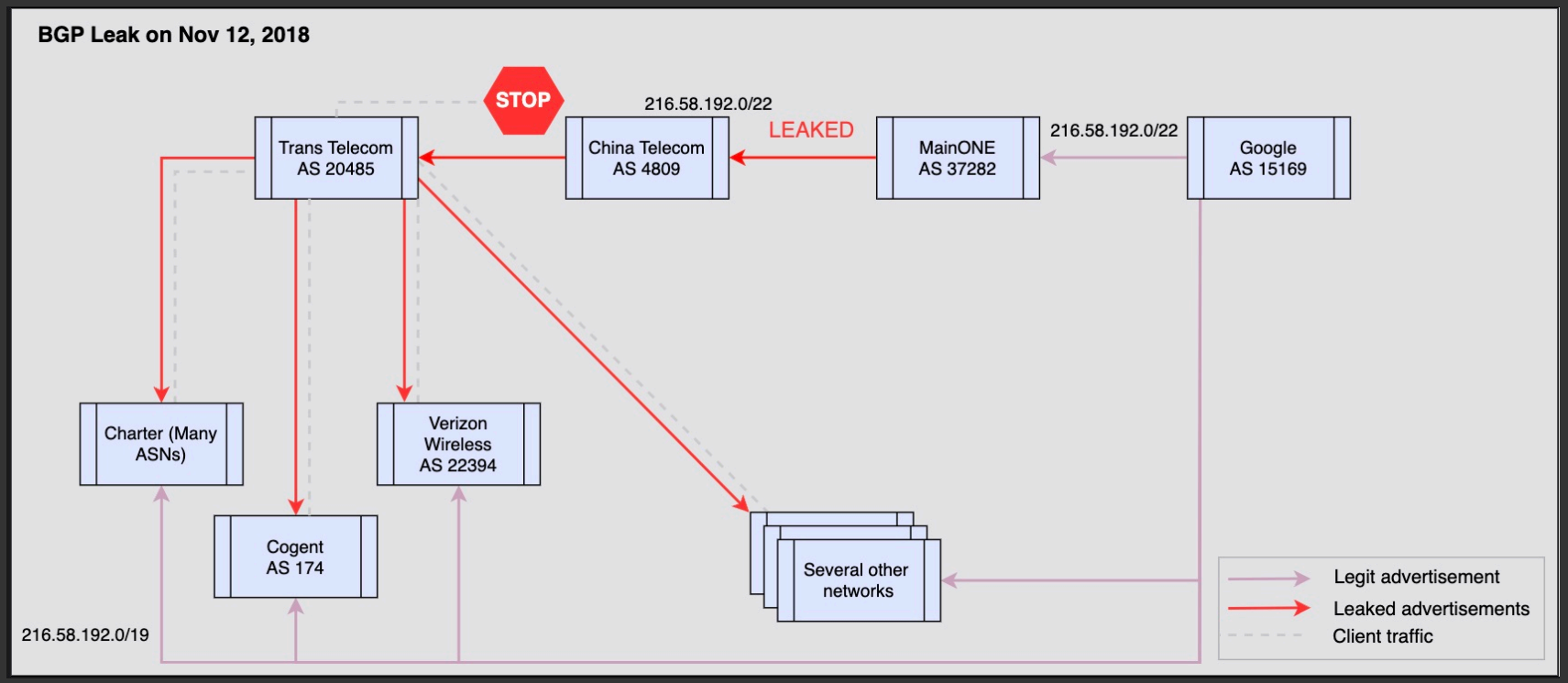
Those errors whose root cause is on the client side are hard to respond to because the service has little to no insight into the client's system. We might try to look for a dip in the load compared to averages, but such a graph is usually hard. It can have false positives and false negatives due to factors such as unexpectedly variable load or if a small portion of the client population is affected.

There are many factors that can cause failures that can result in clients being unable to reach the server. These include the following:

Failure in DNS name resolution.
Any failure in routing along the path from the client to the service provider.
Any failures with third-party infrastructure, such as middleboxes and content delivery networks (CDNs).



Failures due to a routing bug



Design of client-side monitoring System

Initial design

To ensure that the client's requests reach the server, we'll act as clients and perform reachability and health checks. We'll need various vantage points across the globe. We can run a service, let's call it prober, that periodically sends requests to the service to check availability. This way, we can monitor reachability to our service from many different places.

Issues with probers

We can have the following issues with probers:

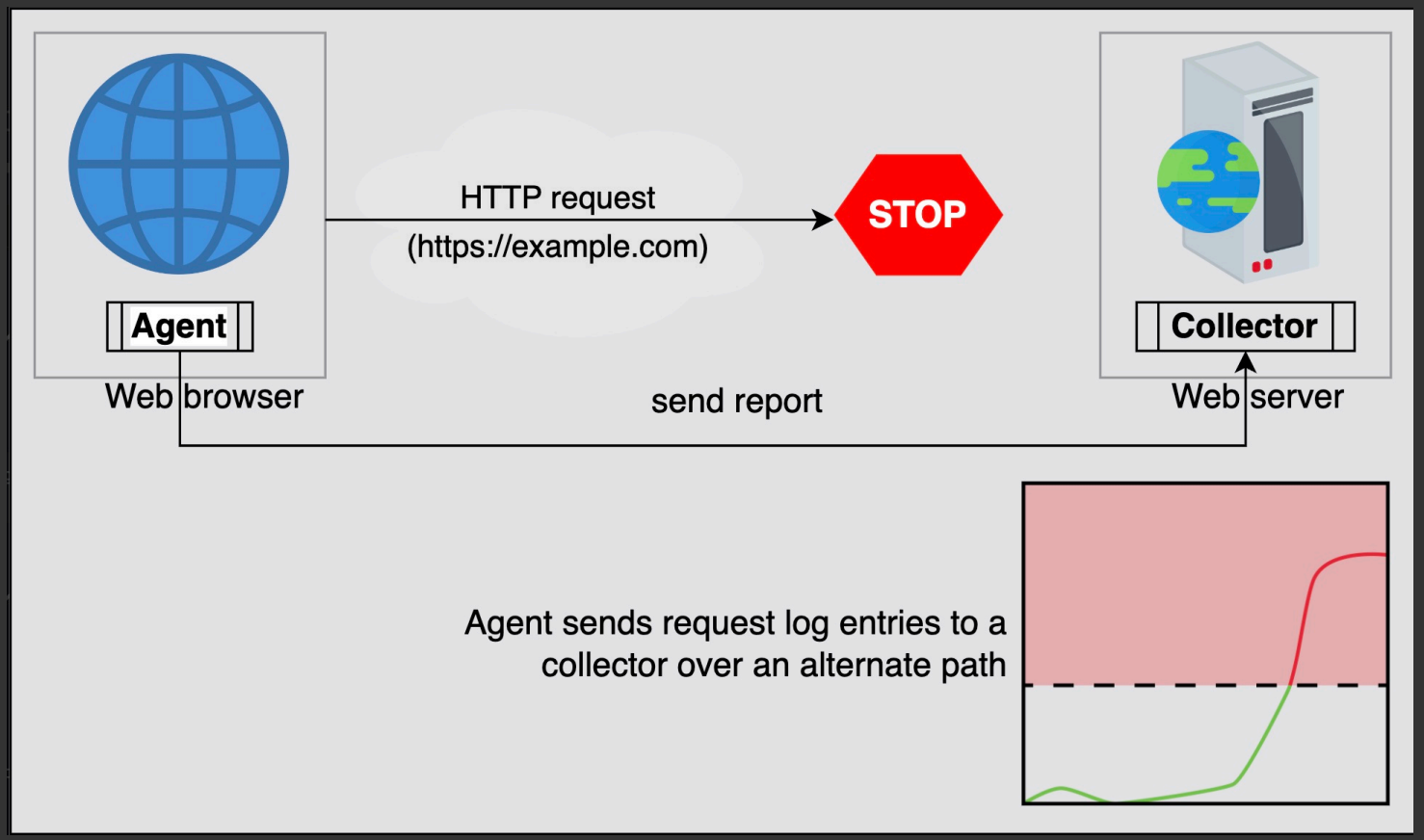
Incomplete coverage: We might not have good coverage across all autonomous systems. There are 100,000 unique autonomous systems on the Internet as of March 2021. It's not cost-effective or even possible to put those many probes across the globe. Country or ISP-specific regulations and the need for periodic maintenance are additional hurdles to implementing such a scheme.

Lack of user imitation: Such probes might not represent a typical user behavior to explain how a typical user will use the service.

Improve the design
Instead of using a prober on vantage points, we can embed the probers into the actual application instead. We'll have the following two components:

Agent: This is a prober embedded in the client application that sends the appropriate service reports about any failures.

Collector: This is a report collector independent of the primary service. It's made independent to avoid the situations where client agents want to report an error to the failed service. We summarize errors reports from collectors and look for spikes in the errors graph to see client-side issues.



Conclusion

In a distributed system, it’s difficult to detect and respond to errors on the client side. So, it’s necessary to monitor such events to provide a good user experience.

We can handle errors using an independent agent that sends service reports about any failures to a collector. Such collectors should be independent of the primary service in terms of infrastructure and deployment.