# Introduction to Client-side Errors

* **Definition**: Client-side errors are issues that occur on the client side of a web application, which are often detected through monitoring HTTP requests that fail to process correctly. These errors are challenging to diagnose because they do not always originate from the server's infrastructure.
* **Types of Errors**: The document discusses several common causes of client-side errors, including DNS resolution failures, routing errors along the path to the server, and failures within third-party services like CDNs.

### Impact and Diagnosis

* **Diagnosing Errors**: Client-side errors can manifest as sporadic spikes in internal server errors (HTTP 500), which may not necessarily indicate a server-side issue. Monitoring these requires careful analysis to differentiate between genuine issues and noise in the data.
* **Challenges**: The transient nature of client-side errors and their dependence on external factors (like network paths and third-party services) make them difficult to track and resolve. Additionally, these errors often produce ambiguous symptoms that can lead to false positives or negatives in monitoring systems.

### Real-world Example: BGP Leak

* **Case Study**: The document details an incident where a peer ISP of Google accidentally announced incorrect Internet routes, causing traffic for many of Google's services to be misrouted through unintended ISPs. This event, known as a BGP leak, was not immediately detectable by Google’s internal monitoring tools because the disruption occurred outside its network infrastructure.
* **Resolution and Lessons**: The incident underscores the importance of having robust monitoring tools that can detect and alert on anomalies not only within but also outside the organization’s direct control. It also highlights the need for proactive monitoring strategies that can adapt to the complex nature of distributed network environments.

### Monitoring Client-side Errors

* **Monitoring Strategies**: Effective monitoring of client-side errors requires tools that can analyze traffic patterns and identify anomalies in data flows, including unexpected route changes or failures in external dependencies.
* **Tool Recommendations**: The course suggests integrating advanced diagnostic tools and adopting comprehensive logging strategies to capture detailed information about client interactions, which can help in tracing the root cause of client-side errors.

# Detailed Design of a Monitoring System

### Overview

* **Purpose**: The aim is to design a monitoring system that can detect and report errors occurring on the client-side, which do not reach the server due to various failures in the connectivity chain.

### Initial Design and Challenges

* **Probers**: The initial approach involves using "probers" that act as clients from various global vantage points to periodically send requests to the service to check availability.
* **Issues with Probers**:
  + **Incomplete Coverage**: Due to the vast number of unique autonomous systems (over 100,000 as of March 2021), it's neither cost-effective nor feasible to have comprehensive coverage using probers.
  + **Lack of User Imitation**: Probers may not accurately mimic typical user behavior, which can lead to gaps in understanding real user experiences.

### Improved Design

* **Integration in Client Application**: The improved system embeds probers within the client application itself, now called "agents," which report failures directly.
* **Collectors**: These are independent entities designed to receive error reports from agents. This separation ensures that even if the primary service is down, the collectors can still gather data.

### Mechanisms and Functionalities

* **Activating and Deactivating Reports**: The document describes methods to enable or disable error reporting, ensuring that users can control their participation.
* **Reaching Collectors Under Faulty Conditions**: It details strategies to ensure that agents can reach collectors even when typical pathways are compromised. This includes hosting collectors on different domains, IPs, or even autonomous systems to circumvent network failures.

### Privacy Considerations

* **User Privacy**: The design emphasizes user privacy, ensuring that only minimal and necessary data is collected, with clear user consent. It avoids sensitive data that could reveal user location or network configurations.

### Conclusions and Benefits

* **System Reliability**: By implementing a client-side monitoring system with independent collectors and embedded agents, the service can quickly detect and respond to client-side issues, improving overall reliability and user experience.
* **Cost-Effectiveness and Scalability**: The system is designed to be cost-effective by reducing the need for extensive global coverage through probers and instead relying on embedded agents within client applications.