**Shield the Service from failures in downstream dependencies – Downstream Resiliency**

1. **Timeouts:** Fail the request if the there is no response for the timeout duration. Without timeouts, it can lead to resource leaks, for example thread may not get returned to the thread pool of the server if it makes a synchronous network call without timeout and the call never returns. Also important on client side. For example, there is a limit to number of sockets the browser can open for a particular host. The socket pool can get exhausted if the network calls never return, so it may not be possible to connect to a host even if the problem at the remote host side is fixed at a later point of time.
2. **Retries:** If the request times out and is caused by a short lived connectivity issue, retry after a backoff time has a high probability of succeeding. But if the server is overwhelmed, retrying may further aggravate, so it needs to be slowed down with increasingly longer delays or till a maximum number of attempts. It’s important to not retry non idempotent calls (for example calls to a payment service) and error situations (401 not authorized) since the call will fail again.
   1. **Exponential Backoff:** delay = 𝑚𝑖𝑛(cap, initial-backoff ⋅ 2attempt) . Problem: Multiple clients may retry at the same time, hitting the downstream service with load spikes that can further degrade it’s performance. To avoid this herding behaviour, a random jitter can be introduced. delay = 𝑟𝑎𝑛𝑑𝑜𝑚(0, 𝑚𝑖𝑛(cap, initial-backoff ⋅ 2attempt)) .
   2. **Retry amplification:** Having retries at multiple levels of dependency chain can amplify the number of retries and the service deepest in the chain will be exposed to the most load. If the pressure increases, this can bring down the entire system. You should have retries in the single level of chain and fail fast in others.Diagram

      Description automatically generated
3. **Circuit Breaker:** If the downstream service is permanently degraded, unnecessary retries can make your service appear slow to the customers. The slowness can cascade to other components of the system. Solution is to not send requests to such downstream dependencies and fail them. Retries are useful when the expectation is that next network call will succeed. Circuit breakers are useful when the expectation is that the next network call will fail.
   1. **State Machine.** Circuit Breakers are implemented as state machines. In Closed state, circuit breakers monitor the errors and timeouts. If it exceeds the threshold, it trips open and blocks the network calls. After some time, circuit breaker transitions to half open and next network call is allowed. If it succeeds, transitions to closed state. It’s important that if the downstream dependency is non critical, it’s better for the service to degrade gracefully than to stop entirely. For example, for amazon front page, if the downstream dependency (recommendations) is unavailable, it’s better to render the page without recommendations than to fail rendering the home page entirely. Otherwise it can have business implications.

**Diagram

Description automatically generated**