System design: Problem Statement

1. 1 million backend server that serves search traffic (think it as search service by google).
2. 10k frontend servers – the FE servers intends to act as load balance the incoming traffic to the BE servers.
3. Incoming traffic to the 10k FE servers are not balanced: there can be cold FE servers, and hot FE servers
4. Design the system/algorithm, where the traffic to BE servers are balanced.
5. The design should be within today's computer architecture, ideally without centralized components.

Assumptions and NFR's To be Confirmed with Zhen, and design to be modified based on feedback.

Assumptions

- Requests are retried by clients when an FE server dies while processing.
- What happens if BE server dies while processing a request or midway ?
  - Do we expect FE to retry ?
- **Question** : Are the FE's and BE's in the Same DataCenter or is this a Per-Region Architecture Split across multiple Data Centres ?.
- BE's are used solely for this purpose (search application) and no other application is deployed on the BE's ?.

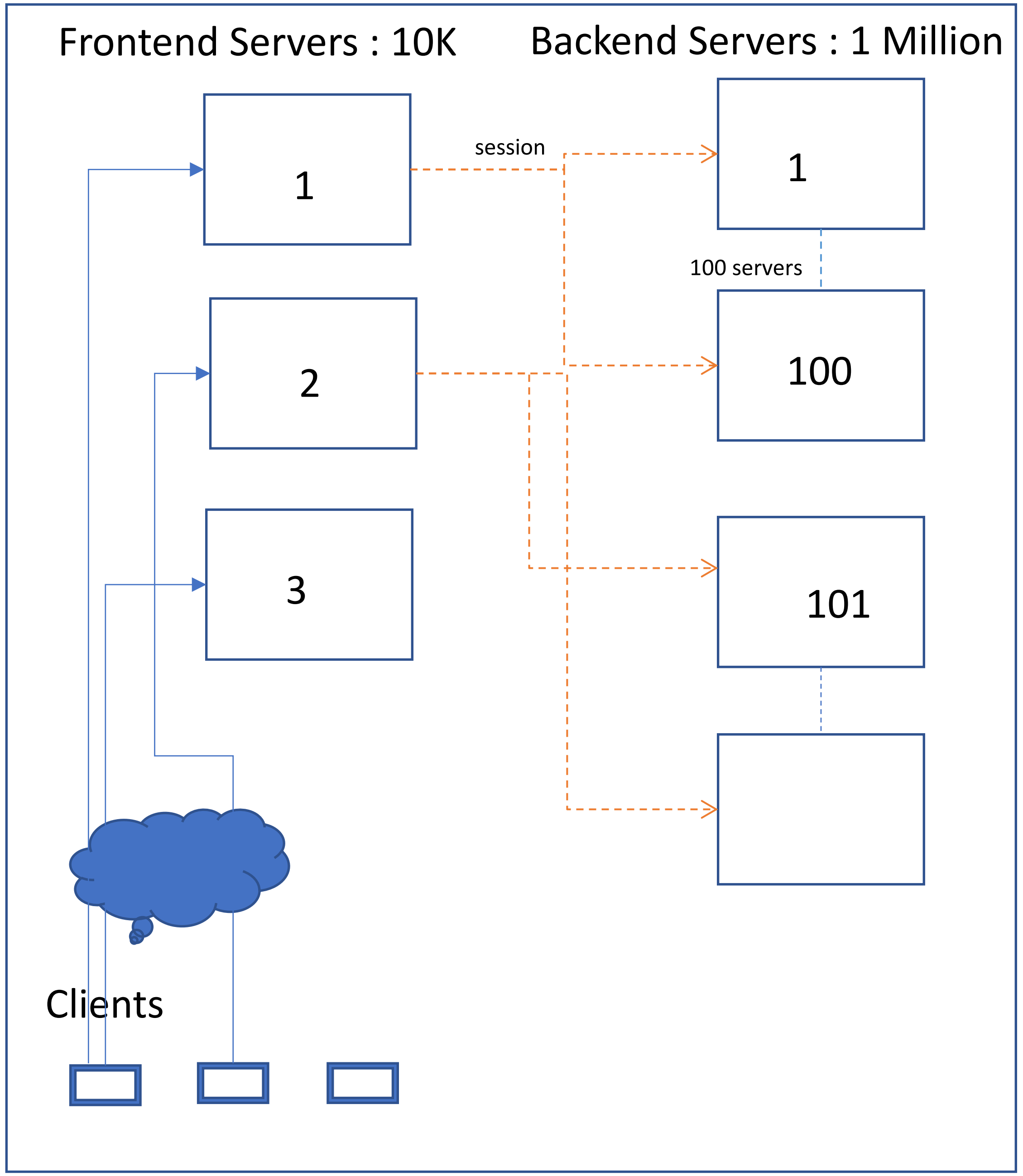
NFR’s

- **QPS :** 100,000 Google searches each second. That’s almost 8.4 billion Google searches per day.
- **Latency Requirements** : 99th percentile latency : 100 ms ?.
  - Google Speed Gospel : The entire process takes, in many cases, less than a tenth of a second – it’s practically instant.
- **QPS per Server** : 100K /10K = 10 requests per second on each FE Server in Ideal Scenario
  - Due to non uniform load on FE servers,
    we assume anywhere from 0 QPS on Cold Server to 100 QPS on FE Hot Servers
  - So overall need to plan for 100 QPS on each FE server
  - 10 requests per second however seems to indicate underutilized servers (depending on capacity of the commodity server deployed)
    - So should we bump up the Global QPS or do we assume the capacity is reserved for YOY growth ?
- **YOY projected growth in traffic**.
  - 10% growth ?.
- **Network Latency** : 1 ms average round trip latency between FE and BE servers.

Three possible approaches.

Approach 1 : **formulaic**

1. 10K FE server's and 1M BE servers so we split 1M /10K = 100 BE servers assigned to each FE Server (can be an initial configuration)
2. FE server has connection pool with upto n (1..n) connections to each of the 100 BE server's
3. Each unique client request is routed to a different BE server in a round-robin manner within the pool.
4. As BE server's go down, and comeback up the connection in the pool is refreshed
5. Detect If the network between the specific FE and BE is unreachable and mark the connection broken
    1. KeepAlive probes ..



Approach 1 : **formulaic**

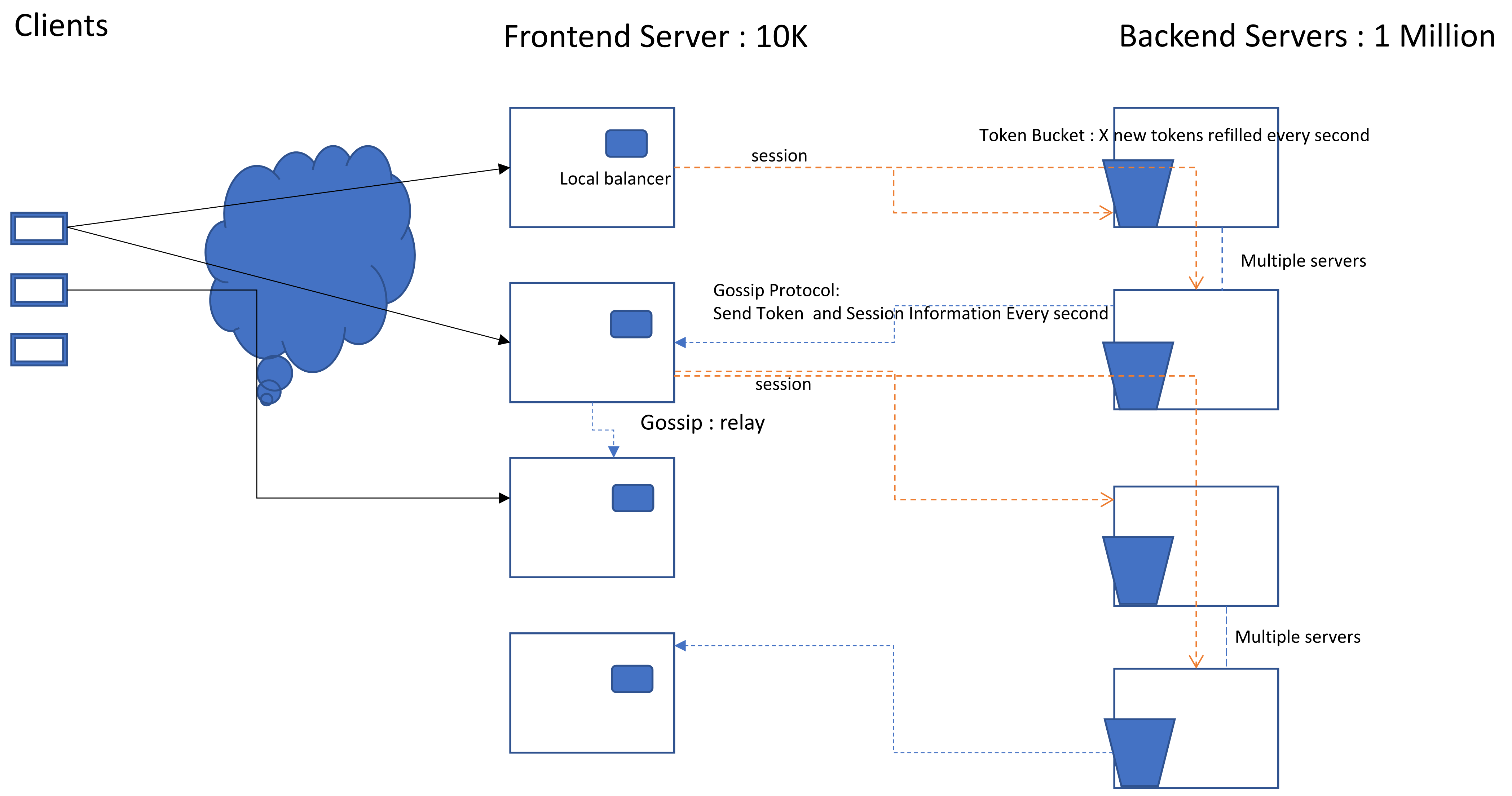
- **Pros**
    - Minimalist approach with no additional components and protocols needed for the implementation.
        - However we cannot claim that the load on BE servers is properly balanced
            - Given there are hot and cold FE's

- **Cons**

    - It assumes load is balanced on the FE Server, which in our case is not true. We have cold and hot FE's. Load on FE's not uniformly distributed.
        - If the FE is a hot server then it might need more BE servers to handle the load.
    - On the other hand the allocation of 100 servers to a Cold FE could be potentially going waste and in that sense the Load is not balanced on the BE servers.
    - If a BE server is permanently down/removed then we need a way to add any new BE replacement server to appropriate FE server configuration

Approach 2 : **Knotty**

- Points (1)(2)(4)(5) same as in *Approach 1 :* **formulaic**
    - Point (3) Client request routing explained in later slide below.
- (6) Each BE has a Token Bucket (of fixed capacity : C) that provides the ability to Limit the number of requests per unit time
    - This prevents callers from rendering the BE as a HotSpot.
    - Also acts as an upper-bound on the load on any BE server at any given point in time.
    - X tokens are refilled into the bucket every second
        - If the bucket is already full then extra tokens will overflow
- (7) The BE Servers use a Gossip Protocol to send information about number of outstanding tokens available in the Token Bucket as well as number of Active TCP Sessions on the BE Server at the current time.
    - The information is sent to FE Servers
        - FE Server's also relay them to other neighbouring FE servers
    - One Gossip message is sent out each from each BE every z seconds (z could be 0.5 or 1 or 2)
- (8) FE Server's consume gossip messages to updates an In-Memory table (BE-Stats) of information about the BE servers
    - Each FE server is interested in upto 100 BE servers only and not the full 1Million
        - Unlike *Approach 1:* **formulaic**, here the list of BE servers would change over time after the initial configuration
            - More on this in next slide.
    - When updating the entry for a BE server, the FE server also performs a latency probe and updates the observed latency in the entry.


Clients
Frontend Server : 10K
Backend Servers : 1 Million
Local balancer
session
Token Bucket : X new tokens refilled every second
Multiple servers
Gossip Protocol:
Send Token and Session Information Every second
session
Gossip : relay
Multiple servers

Approach 2: Knotty

BE server	Tokens	Sessions	Timestamp
w.x.y.z	3	20	xxxxxx

Gossip messages form BE servers would contain the following information.

Approach 2 : **Knotty** Continued…

- (9) We understand from problem statement that requests to FE servers are not Balanced and there can be Cold and Hot FE servers from time to time.
  - If an FE Server goes Cold then the BE's assigned to such as FE server (initial configuration) would show high Token count and Low Sessions
    - A HOT FE which observes this data (on receiving a Gossip message from a BE) can (instead of discarding) decide to use the cold BE as one of its new BE's. A session is established with that BE.
      - If there are Multiple Hot FE's in the system then each one of them might try to execute this step, however the randomness in Gossip will ensure that not all Hot FE's will receive information abot the same BE at the same time.
    - In turn the HOT FE can discard one of its own BE's which is showing high latency (L > L-Threshold) and/or session count beyond a configured threshold (SC > SC-Threshold)
      - The high latency or session count should be observed consistently for atleast n (=2 .. 5) cycles of checks before a Hot FE takes this decision to swap out.
        - Otherwise it could lead to thrashing where FE is frequently swapping BE's in and out.
- (10) Determination of a Hot FE : how does an FE determine its own HOT status
  - It could check its TCP Backlog
  - The TokenCount and Latency Status of all its BE's is below some threshold

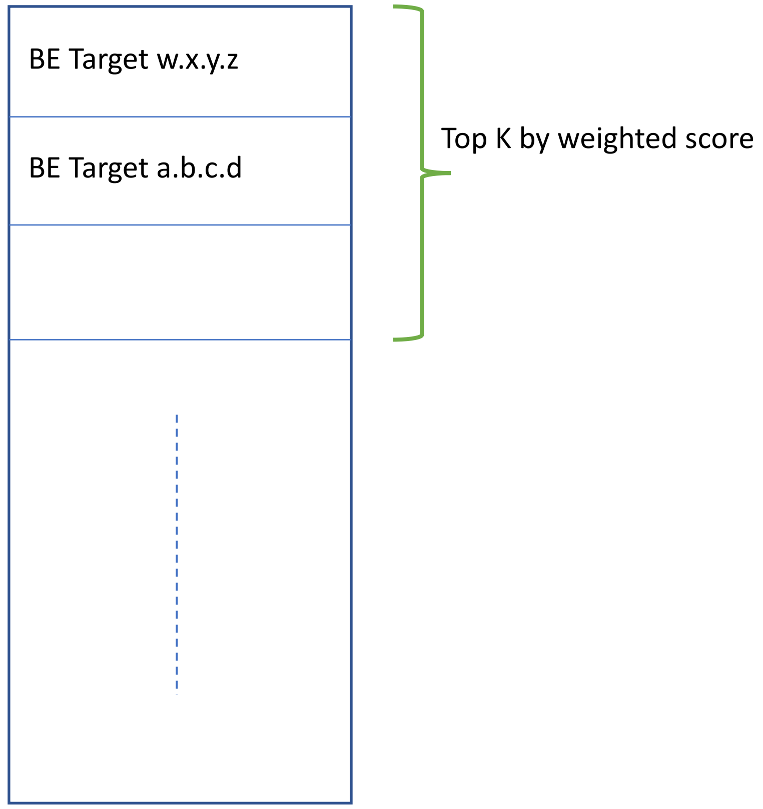
Approach 2 : **Knotty** Continued…

- (11) Client Request Routing: The FE Server maintains an In-Memory Priority queue of Top K target BE’s based on a weighted score.
    - The Score could be w1 * TokenCount + w2 * Latency + w3 * SessionCount (wi’s are weights that sum upto to 1)
    - The Priority Queue is updated every time the BE-Stats table is updated on the FE
        - Or the Queue update could be based out of a different Timer Task on the FE that fires every Ti (=1..5) seconds. But this can incur race conditions that would have to be handled.
    - The client is routed to one of the BE’s chosen randomly from the Top-K BE’s
    - When there are multiple (n) client requests waiting then, n > K then other BE’s at K+1…2K-n positions in the priority list are also considered as part of the Random Picking.
        - Attempt is made to choose as many unique BE’s as the number of unique client IP’s.
            - This allows for other FE’s to utilize the capacity on those BE’s
            - However Multiple requests from same client maybe sent to same BE for any locality and/or caching benefits.

BE-Stats In-Memory Table

BE server	Tokens	Sessions	Latency(ms)	Timestamp
w.x.y.z	3	20	100	xxxxxx

Priority Queue



Approach 2 : **Knotty** Continued…

Pros

- It provides an upper bound on the load on any BE server at any given time using a token-bucket
- By definition this approach will ensure BE's sitting idle become candidates for selection by Hot BE's.
  - In summary the approach attempts to set a non-zero lower bound and an reasonable upper bound on the requests being served by each BE at any given point in time.

Cons

- The approach requires additional protocols and algorithms to run in the FE's local load balancer component
  - (Gossip) for the FE's to learn about the status of BE's
  - BE's would need to have Token Bucket for Rate Limiting (setting an upper bound on rate of requests)
  - In-Memory DataStructures are used in each FE for dynamically updating its associated BE's
  - In-Memory DataStructures are used for computing the Top-K candidate BE's that can serve any Client Request
  - Multiple configurable parameters C, X, z, n, K, w(i), t, referenced in the algorithm need to be experimented with
- FE failure would cause it to loose all its in-memory information and it would then start fresh with its initial configuration of allocated BE's which may be used by other FE's in the mean time due to very definition of the algorithm.
  - The expectation is the FE would eventually drive itself into a state where it has the a good set of BE's to which it can route the requests.
  - Trying to persist the FE state to disk/DB does not make sense and could limit scalability

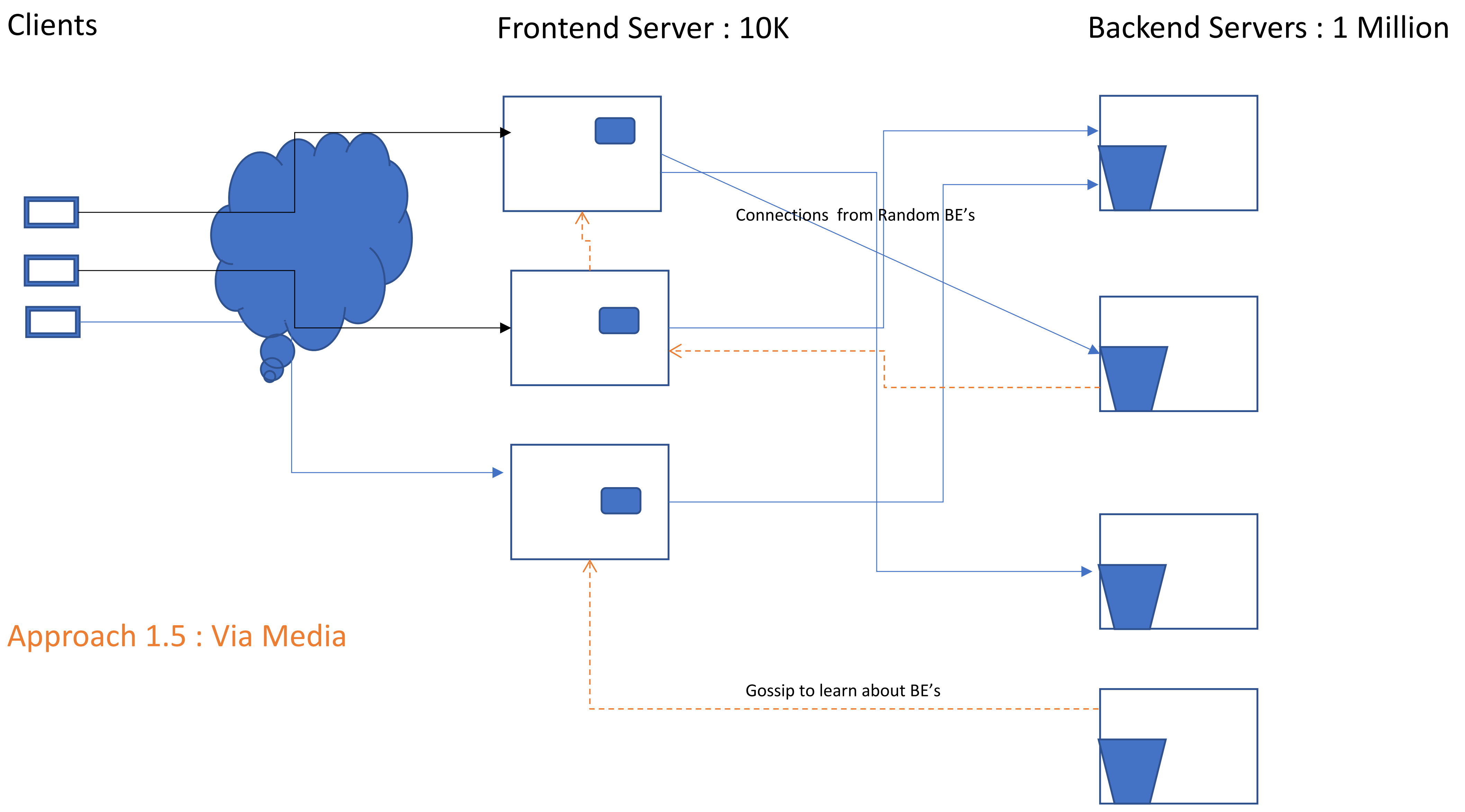
Approach 2 : **Knotty** Continued…

Cons

- Possibility of Rejected request due to Token Rate Limit
    - When a client request is routed eventually by the FE, it could so happen that the BE has no Token’s in the bucket
        - Multiple FE’s are contacting the same BE and the TokenCount for the BE stored in the FE is a delayed count.
        - This could result in a 429 HTTP status back to the client.
            - This could be undesirable.

Approach 1.5 : Via Media

- This approach does away with any initial configuration of BE’s for FE.
    - It will need information about potentially all the 1M BE’s.
        - Learnt through Gossip
- At first Connections are obtained at Random from the set of BE’s on any given client request
    - Connections thus obtained are pooled with a timeout of **N** minutes
    - When a pooled connection times out it is removed from the pool
- Any new client requests are first served from the Connection Pool
    - After which new connections are dynamically obtained at Random from the set of 1M BE’s
        - Subsequently added to the Pool for reuse
- The BE’s could still have their Token Buckets configured as in **Approach 2** to set an upper bound on the request rates


Clients
Frontend Server : 10K
Backend Servers : 1 Million
Connections from Random BE's
Gossip to learn about BE's

Approach 1.5 : Via Media

Approach 1.5: Via Media

Pros

- Simpler than Approach 2.
- Does not require any initial configuration (of BE List) as proposed in Approach 1 and 2
  - Tries to Randomly acquire connection to any of the 1M BE's everytime it needs a new one
- Less impacted by BE servers dying and new BE servers being added to the system
  - Connections are pooled only for N minutes
- Will try to obtain more connections for Hot FE
  - And with connections expiring every N minutes there could be a lot of activity
- The Cold FE does not impact this algorithm as it tries to randomly pick BE's on the Backend.

Cons

- The FE would not have information about the Token Counts and Active Sessions on the 1M BE's, it also does not have information on Latency of BE's.
  - The randomized connection requests by all the 10K FE's on getting connections to 1M BE's can hit more frequent Rate Limits on the BE's causing 429 errors to be returned to Clients.
- It opens new connections more often compared to Approaches 1 and 2
  - Performance can be worse, though how much worse needs to be assessed practically.
    - TCP connection handshake delays + TCP slow start

Estimations

Assumptions

- Round trip latency in data center = 1000 micro-second (1 millisecond)
- Search response payload upto 1Mb (1st page of many search results),
- Read 1Mb sequentially over network = 10ms
- FE Apache server supports 150 concurrent connections by default

Calculation

- FE Processing for selection of Target BE = Lookup K entries in Priority Queue (approach 2) + Lookup a connection from the pool and dispatch the request.
  - FE processing: (5 * 100) ns + Connection Lookup from pool: 500ns + Send request utpo 2K bytes: 20 microseconds + read 1Mb response sequentially over network : 10ms + Data center roundtrip 1ms
    - 1microsecond + 20 microsecond + 10ms + 1 ms = 11.021 ms
- 100ms 99th percentile latency requirement : so it leaves 88.979 milliseconds for BE processing logic
- Throughput of 10K FE servers (assuming all active) : 150 * 10K = 1.5M requests per second