## "Structured Service Addressing" using IPv6 unicast addresses to enable controlled client side load sharing and content/direct addressable storage in a “client – cluster communication paradigm”.

**Problem statement**: With the proliferation of clustered services, the traditional client-server communication pattern with a very simple (naïve?) client implementation and small number of VIP's to represent the service, causes the need for state heavy or synchronized coordination mechanisms. Such might include sticky load balancers, monitoring nodes with cluster topology maps and load balancers with consistent hashing tables to map client-access to a clustered resource. This approach imposes limitations to scalability and efficiency by requiring a lot of cluster-side compute and memory resources to maintain, and in some cases creates aggregation points in the infrastructure which eventually becomes bottlenecks. This approach also discourages the use of stateful service frontends in modern scale out architectures. Stateful service frontends can have significant efficiency and latency advantages over stateless frontends with stateful backends.

**Proposition**: *“Structured Service Addressing”;* Using a large number of network addresses, mapping one or many addresses to each actual service endpoint, the client computes the destination address or addresses needed to reach the resource based on a DNS APL RR with a large address range describing how to reach the service and optionally; a destination address mapping pattern and “hard to tamper” client state like its own network-source address used to communicate with the service. The client then calculates the destination address to reach the service. This could be as simple as picking destination address(es) at random to get naïve load sharing or computing the destination address based on the source address, predefined rules and the service IP-range to allow for stateless L4* DDoS protection at the cluster side or computing destination address based on the content the client needs form the service to directly address a scaled content service or a combination of these three methods.

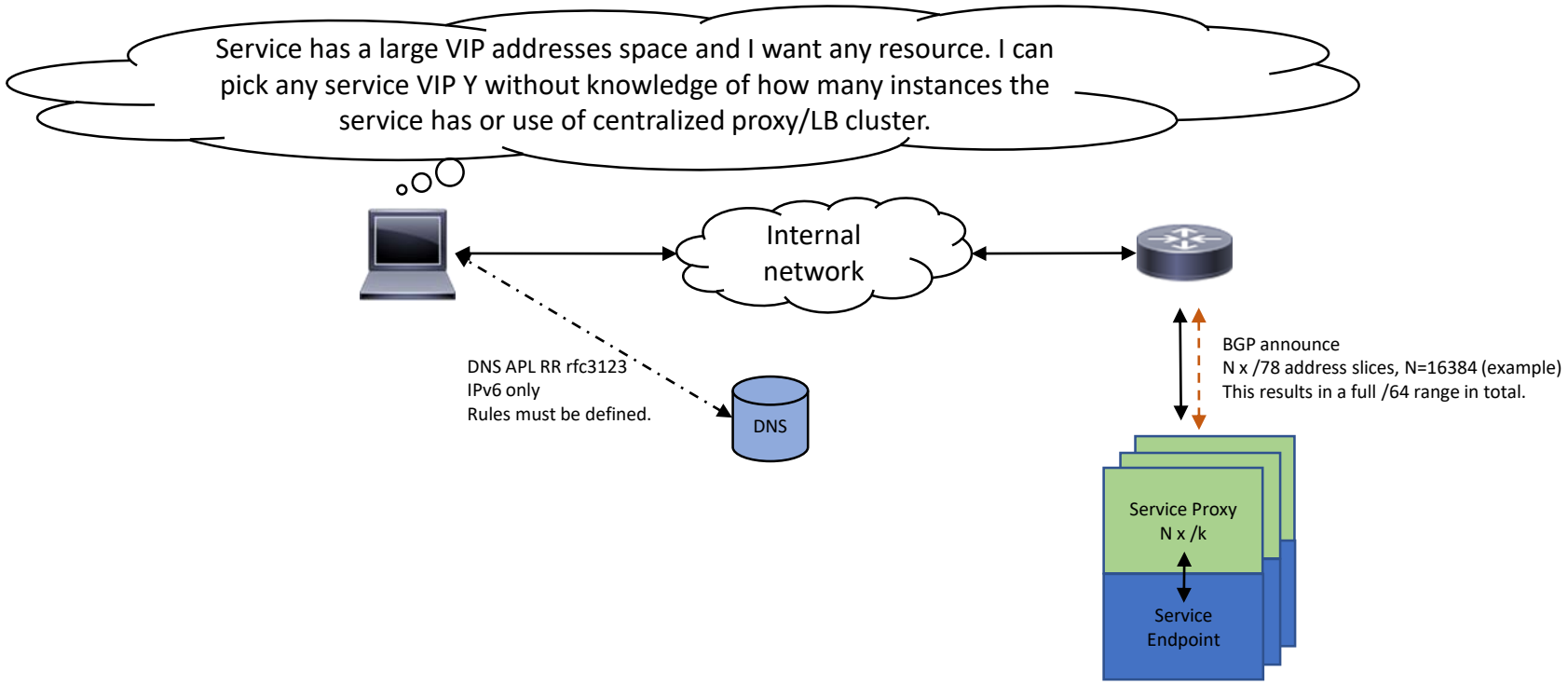
**Caveats**: NAT and IP spoofing will at least complicate implementation and operation.

**Resources**:

https://tools.ietf.org/html/rfc5375 "IPv6 Unicast Address Assignment Considerations“
https://tools.ietf.org/html/rfc3123 "A DNS RR Type for Lists of Address Prefixes (APL RR)“
https://tools.ietf.org/html/rfc3587 "IPv6 Global Unicast Address Format"

Anders Jensen
v1.1

# Use case 1: Client-cluster communication – Internal known client, naïve load sharing




For this example N x /128 subnets is also possible. It is important to have many more subnets (N) than service proxies to be able to scale up the number of service proxies and endpoints without adding addresses or getting imbalanced load. If you do not use /128 subnets you could also split the subnets into smaller slices to enable more scaling when needed without client visibility. Adding addresses is not desirable because scaling now becomes a client visible operation and may require DNS refresh to balance load correctly.

Example:
Service address space is a single /64 subnet and we want to scale dynamically between 10 and 1000 instances.
Having at least 1000 address slices is needed, and to get full uniform coverage of the /64 address space that would require at least 1024 subnets of size /84, however this would lead to great imbalance as the number of service instances passes 512 and some have 2 address slices and some have only one giving them only half the traffic. Using 16386 subnets of size /78 would eliminate this imbalance all the way up to the maximum anticipated scale of 1000. This will require the perhaps extreme example at minimum scale of each service proxy handling up to 1639 subnets each. One cloud of course have a process to split and combine subnets as the service scales up and down without the clients ever noticing the topology change as long as the whole APL RR is covered at all times and the RR is not changed. It is not necessary to use a /64 for a service, but it plays nice with routing and the APL record.

# Structured Service Addressing - Operation

1) Client asks DNS for IP address.
2) DNS returns a list of subnets (APL – Address Prefix List).
3) Client picks destination address(es) to be used based on predefined rules. NOTE: Rules and their format has not been described.
4) The address will lead to one of many service endpoints and may depend on one of these example rules;
    1) "**Naïve load sharing**". The client picks a random destination address from supplied prefix list for efficient client side load sharing without any knowledge of number of endpoints, anycast, proxies or load balancers. This will not give any advantage over a service mesh in most cases, but can perhaps be used as a means to more efficiently handle extremely scaled services or more likely in combination with 2) and 3). Can also work with round robin DNS. This scheme will be vulnerable to DDoS attacks.
    2) "**Controlled load sharing by calculating destination address based on source address**". The cluster resource must enforces adherence to rules. This can be done as a scaled out stateless operation in front of the actual resource, comparing source and destination addresses using the predefined rules. Example: source address 64 MSB must match destination address 64 LSB. Computation based on source address may be used to harden against DDoS attacks such that a large group of clients may not coordinate to overwhelm a small number of service endpoints.
    3) "**Directly addressable storage service**". The client computes the destination address to the endpoint holding the content based on (parts of) the contents address. Typically this would be the object address in an object store, but schemes to support some kind of file- or block-storage should be possible.
    4) Some combination of the above methods or other methods, perhaps based on hashing algorithms.
    5) Other schemes might be possible perhaps in combination with a L4 protocol carrying a token and/or "proof of work" concepts.
    6) An analog to frequency hopping could be done by requiring the client to know a combination of time and valid address. This might be useful when deploying military applications on an unrestricted network like the Internet.

# Structured Service Addressing - **Advantage**

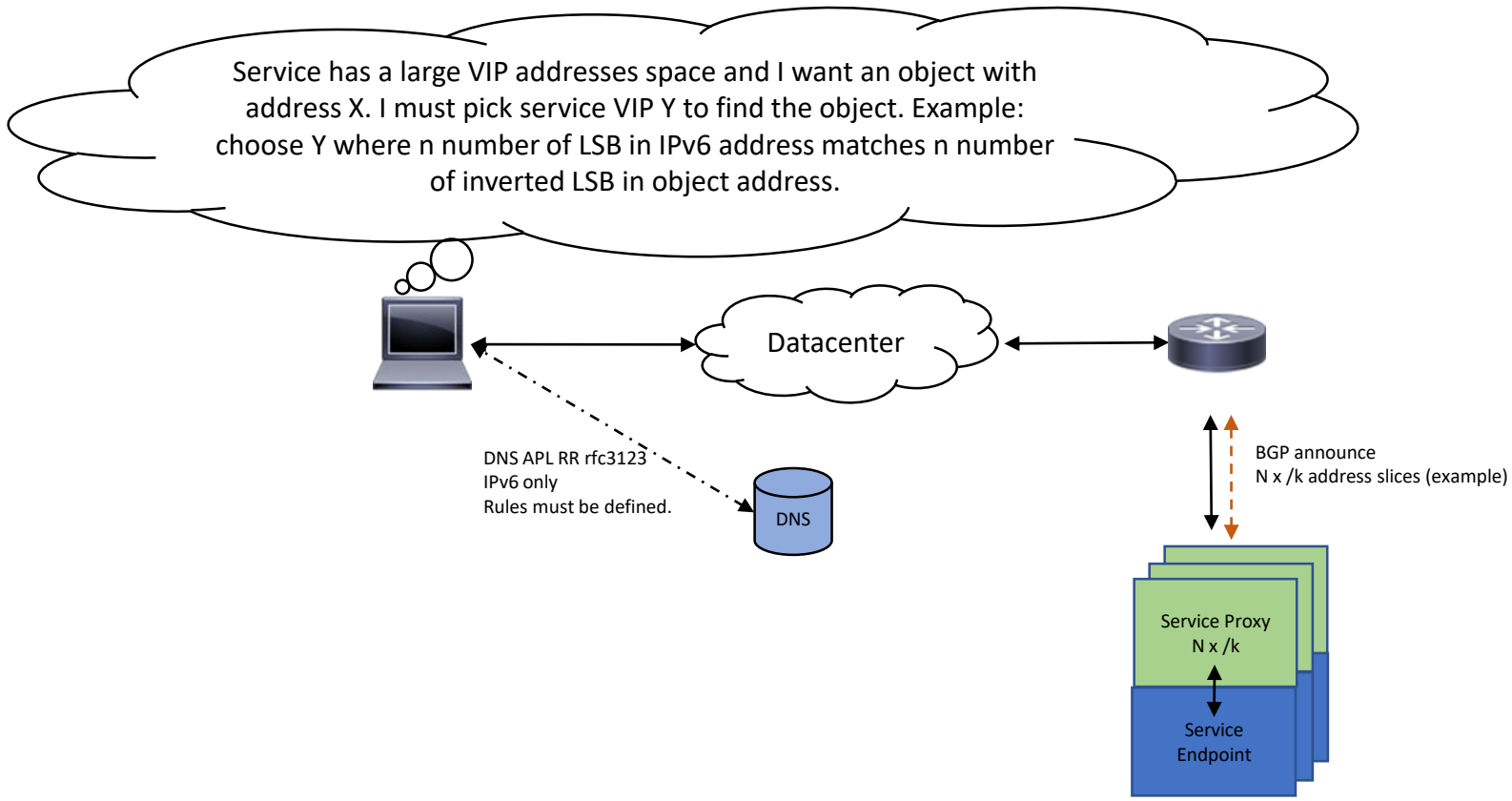
- Clients may directly connect to endpoints of a massively scaled out resource without having any information about the topology of the cluster (naming abstraction). This may include information like; how many endpoints are actually in use and which IP addresses are mapped to each endpoint.
- Clients may directly connect to endpoints without having to go through Stateful proxies accessing topology mapping services, as routing to endpoints is standard (IP) packet routing.
- Groups of clients can be forced to share its load on to the cluster by various methods where logic is applied to generate the correct service address to pick. Non-conforming packets would typically be dropped or put on a low priority queue. Stateless access control can be done on a per packet basis without knowledge of a connection. This makes it possible to divide the traffic into many streams before doing stateful inspection or termination.
- State replication for highly available endpoints may be limited to an absolute minimum for Stateful protocols or sessions by having only one or a small sett of endpoints responsible for tracking the state of each connection. Client could be aware of how to reach all relevant state replicated endpoints without any cluster topology knowledge. This could make Stateful services useful in a cloud based environment again as the responsibility for state replication is deterministically distributed.
- No client side visibility into how or when the cluster side is scaled.

# Structured Service Addressing - **Requirements**

- **Client side**
  - Client side capability to handle lists of subnets (DNS APL “address prefix lists” Resource Record).
  - Predefined rules for the client to pick addresses from the address space provided with APL. Not aware of any such implementation today. The need for this capability is also mentioned in the RFC for APL. In chapter 7. Applicability Statement.
- **Cluster side**
  - Many cluster side loopback interfaces for each endpoint (POD, VM) capable of handling large (IPv6) subnets to terminate TCP or other protocol requiring a Stateful endpoint. Handling many subnets per endpoint serves to hide the topology from the client while scaling the resource by allowing the resource to reallocate subnets to other endpoints. If the resource is not content addressable the subnets can be of size /128. For a content addressable resource the large subnets can be used to map a portion of a storage address space to an IP address range, allowing the client and network to cooperate to locate the endpoint(s) that is holding the content.
- **Network**
  - Large address space, to “carry more information in the address fields”, IPv6 in practice.
  - IPv6 ('globals') [RFC3587] reachability from client to all service endpoints. NAT will at least complicate operation.
  - Capability to move many small subnets inside the cluster between endpoints. (L3 SDN and service mesh)
  - More anti-spoofing capabilities on the Internet?

- Possible starting point for a roadmap
  - Implement some simple client and cluster side rules in a sidecar proxy to test functionality inside a datacenter for service to service connections.
  - Probably need a “connection upgrade” mechanism (DNS) that legacy devices will ignore. Changing the DNS behavior of end user devices will be a very slow process.

# Use case 2: Client-cluster communication – Internal known client, content addressable storage




**Notes**:

- Allow client to directly connect to the instance holding an object or block with a known address by implementing a distributed addressable memory where part of the IPv6 address matches the object address such that it is possible to calculate which IP address the object is located at.

# Use case 3: Client-cluster communication – External unknown client, forced load sharing: **overview**

Reference, Google Maglev:

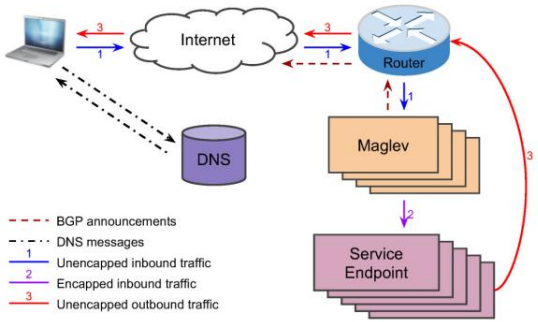
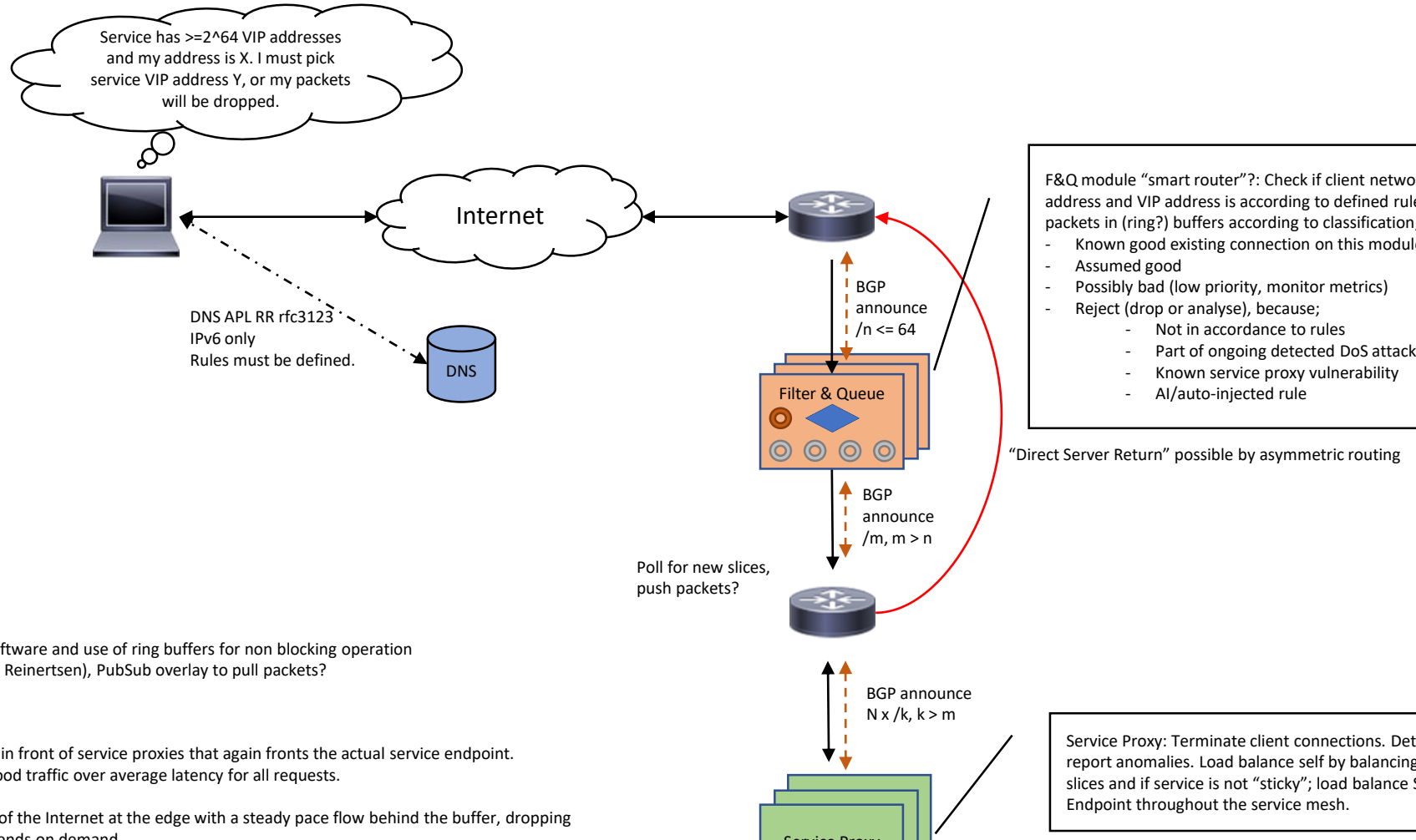



Figure 2: Maglev packet flow.




**Inspiration:**
- Google Maglev load balancer
- Lyft Envoy service proxy, istio.io
- Project Calico L3 SDN
- Pelikan Cache (Yao Yue, Twitter), data/control plane separation in software and use of ring buffers for non blocking operation
- Scheduling; Drum Buffer Rope (Goldratt). 2GLPD, Cost of Delay, (Don Reinertsen), PubSub overlay to pull packets?

**Notes**:
- Reduce attack surface by implementing a transparent adaptive filter in front of service proxies that again fronts the actual service endpoint.
- Optimize for high percentile latency for known good and assumed good traffic over average latency for all requests.
- Reduce state and state synchronization in load balancer.
- Protect from DoS attacks by decoupling the unpredictable workload of the Internet at the edge with a steady pace flow behind the buffer, dropping the least important packets if necessary, but aiming to scale up backends on demand.
- Force client side load sharing by using a new scheme (SSA) where parts of the source address determines the one or few VIP address(es) that will be allowed by the filter & queue load balancer for each client.
- Allow for Direct Server Return (DSR) without encapsulation by announcing BGP routes for many (k) small slices (l) of the VIP space creating a 1:1 relationship between any client and service proxy. Perhaps RDMA/iWARP based on whether service proxy and F&Q module is in same L2 domain. And perhaps poll or rate limited forwarding from F&Q module.
- Load balancing service proxies is achieved by handing over or splitting slices to other/new proxies. The two proxies that coordinate the handover also forward traffic from old to new instance in the short timeframe when routing is converging. Load balancing endpoints is done by the service-mesh.
- Help democratize cloud services by letting everyone build a scale out front-end "loadbalancer".

# Use case 3: Client-cluster communication – External unknown client, forced load sharing: **routing example**