Resolving Smarter, Not Harder: Performance Gains via Query-Aware Filtering

Dipsy Desai Liz Izhikevich

Jelena Mirkovic

University of Southern University of California USC

Information Sciences

California Los Angeles

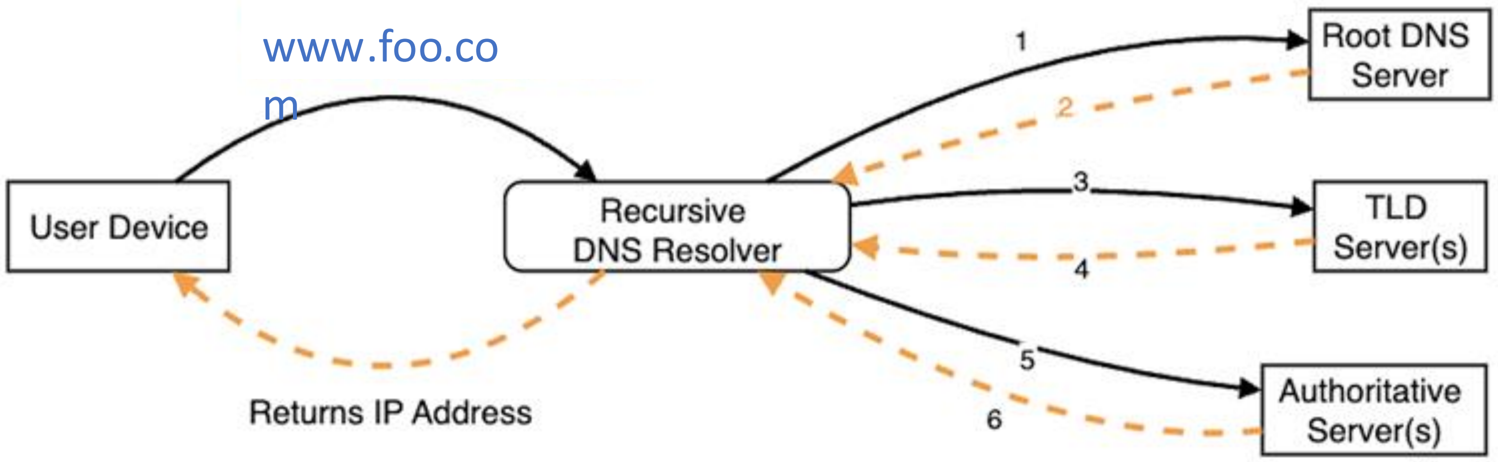
Institute

Pulse Research Week

9 December 2025

Domain Name System (DNS)

- Phonebook of the Internet
- Translates human-readable domain names to machine-readable IP addresses
- Hierarchical in nature
  - No single entity is responsible for resolving the whole name



What is the Need for this research?

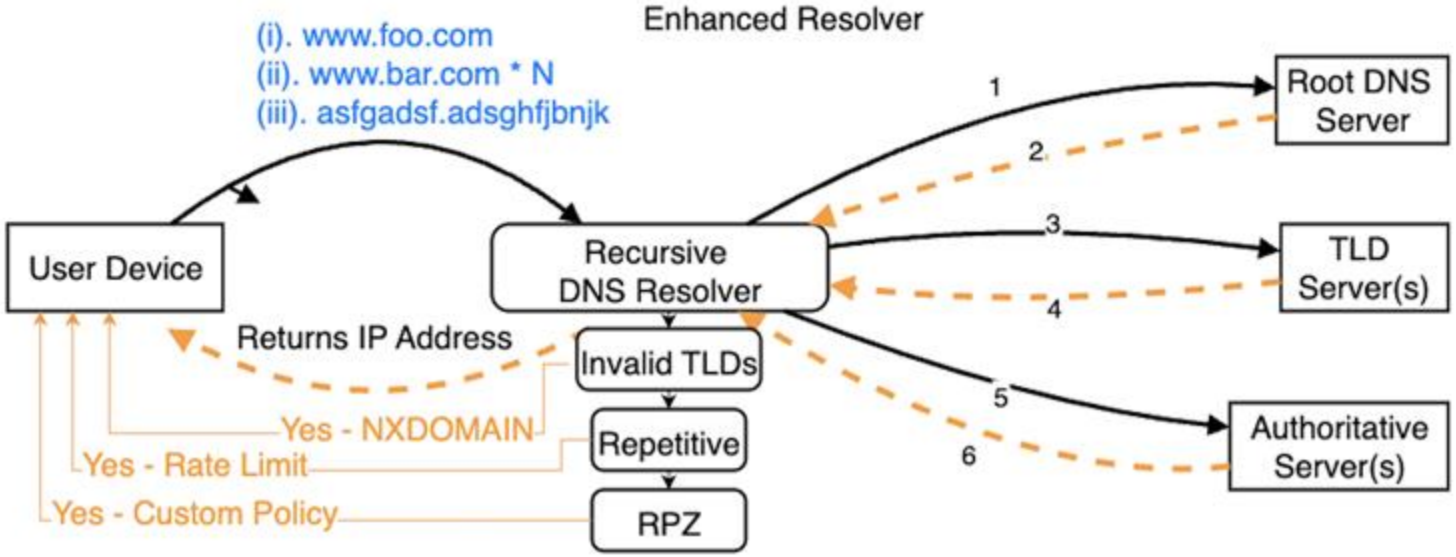
- > 70% of queries to root servers are junk (SIGCOMM’92, SIGCOMM CCR’08, PAM’03)
    - Queries to Non-existent TLDs
    - Repeated Queries to the same TLD (before the TTL expires)

- DDoS Attacks on critical infrastructure
    - f.root-server in 2002, 6 root servers in 2006, 13 root servers and Dyn in 2016

- Used by botnets and C&C systems, impact performance, resulting in poor user experience (USENIX Security’12, SIGCOMM-NeT’04, PAM’04, IEEE Euro CND’11)

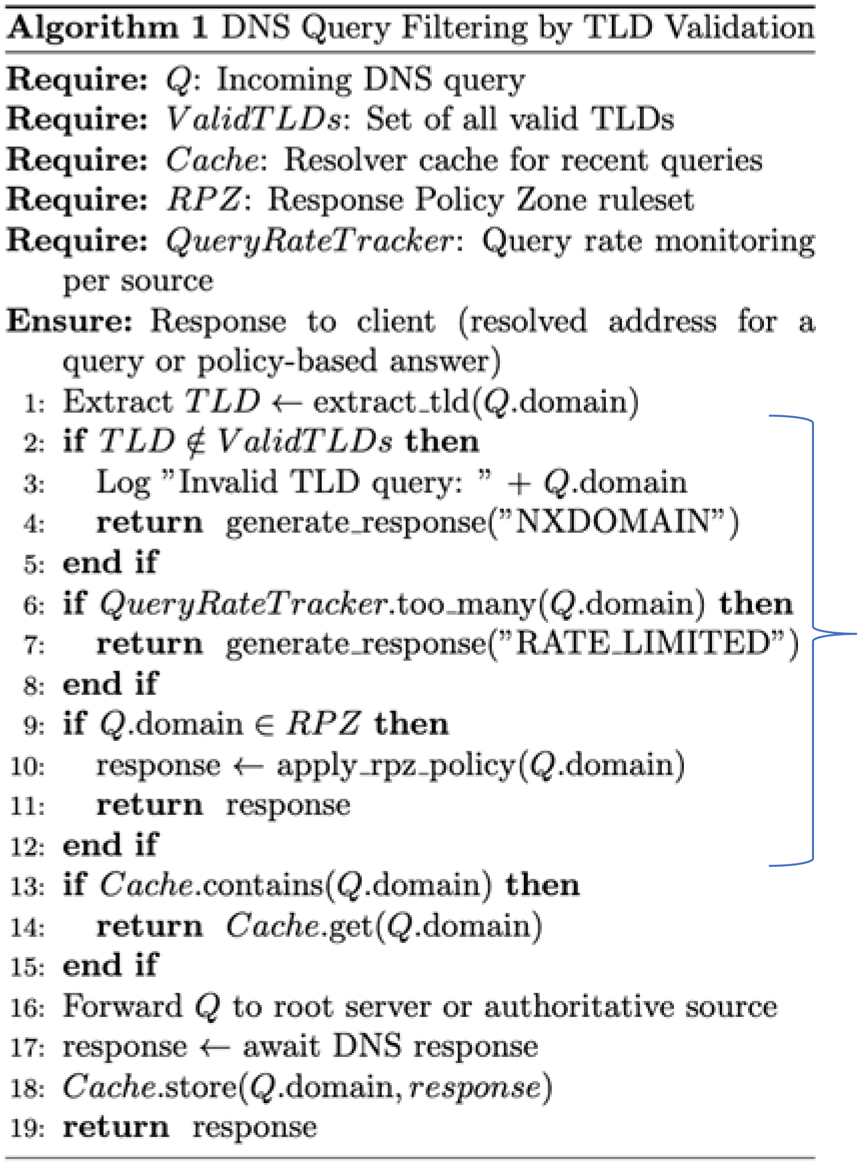
Our Proposition

- Enhanced Resolver - makes decisions locally
- TLD validation, Rate limiting and RPZ support
- Most effective in filtering queries to root servers



Making Decisions Locally

- List of TLDs from IANA
  - Fetch once every 24 hours
- Generate NXDOMAIN response locally for non-existent TLDs
  - Prevents such queries from going outside the network
- Configure RPZ rules as needed
  - For example to handle Chromium queries, split horizon DNS specifications, etc.



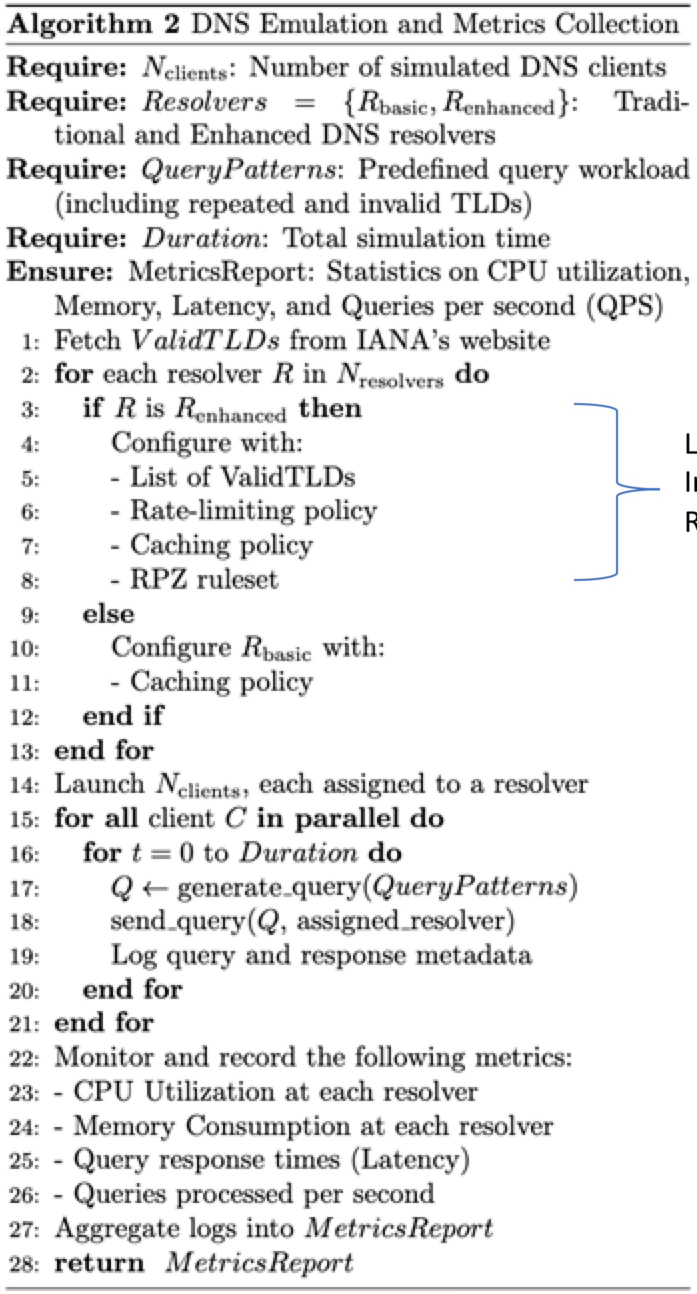
Algorithm 1 DNS Query Filtering by TLD Validation

```
Require: $Q$: Incoming DNS query
Require: $ValidTLDs$: Set of all valid TLDs
Require: $Cache$: Resolver cache for recent queries
Require: $RPZ$: Response Policy Zone ruleset
Require: $QueryRateTracker$: Query rate monitoring
    per source
Ensure: Response to client (resolved address for a
    query or policy-based answer)
 1: Extract $TLD \leftarrow$ extract_tld($Q$.domain)
 2: if $TLD \notin ValidTLDs$ then
 3:     Log "Invalid TLD query: " + $Q$.domain
 4:     return generate_response("NXDOMAIN")
 5: end if
 6: if $QueryRateTracker$.too_many($Q$.domain) then
 7:     return generate_response("RATE_LIMITED")
 8: end if
 9: if $Q$.domain $\in RPZ$ then
10:     response $\leftarrow$ apply_rpz_policy($Q$.domain)
11:     return response
12: end if
13: if $Cache$.contains($Q$.domain) then
14:     return $Cache$.get($Q$.domain)
15: end if
16: Forward $Q$ to root server or authoritative source
17: response $\leftarrow$ await DNS response
18: $Cache$.store($Q$.domain, $response$)
19: return response
```

Lines 2—12
In Enhanced
Resolver Only

Implementing Resolvers

- Unbound 1.23 and SPHERE Testbed
- Down-sampled Query Pattern from b.root-server’s DITL collection in 2025
  - To emulate real world DNS Traffic
- Evaluate both Basic and Enhanced Resolver
  - Collect and compare performance Metrics



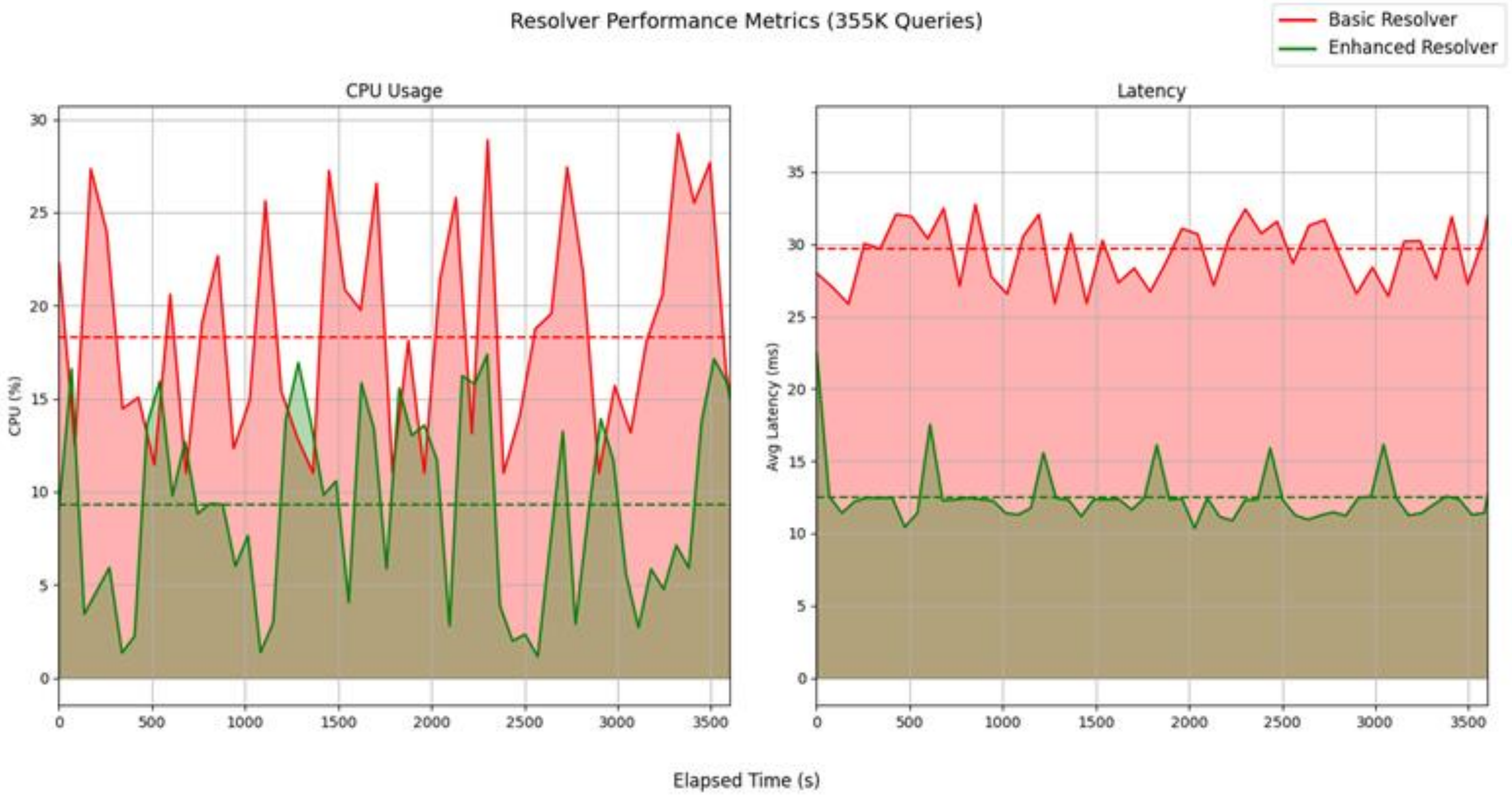
Algorithm 2 DNS Emulation and Metrics Collection

```
Require: $N_{\text{clients}}$: Number of simulated DNS clients
Require: $Resolvers = \{R_{\text{basic}}, R_{\text{enhanced}}\}$: Traditional and Enhanced DNS resolvers
Require: $QueryPatterns$: Predefined query workload (including repeated and invalid TLDs)
Require: $Duration$: Total simulation time
Ensure: MetricsReport: Statistics on CPU utilization, Memory, Latency, and Queries per second (QPS)
 1: Fetch $ValidTLDs$ from IANA's website
 2: for each resolver $R$ in $N_{\text{resolvers}}$ do
 3:   if $R$ is $R_{\text{enhanced}}$ then
 4:     Configure with:
 5:     - List of ValidTLDs
 6:     - Rate-limiting policy
 7:     - Caching policy
 8:     - RPZ ruleset
 9:   else
10:     Configure $R_{\text{basic}}$ with:
11:     - Caching policy
12:   end if
13: end for
14: Launch $N_{\text{clients}}$, each assigned to a resolver
15: for all client $C$ in parallel do
16:   for $t = 0$ to $Duration$ do
17:     $Q \leftarrow$ generate_query($QueryPatterns$)
18:     send_query($Q$, assigned_resolver)
19:     Log query and response metadata
20:   end for
21: end for
22: Monitor and record the following metrics:
23: - CPU Utilization at each resolver
24: - Memory Consumption at each resolver
25: - Query response times (Latency)
26: - Queries processed per second
27: Aggregate logs into $MetricsReport$
28: return $MetricsReport$
```

Lines 3—8
In Enhanced
Resolver Only

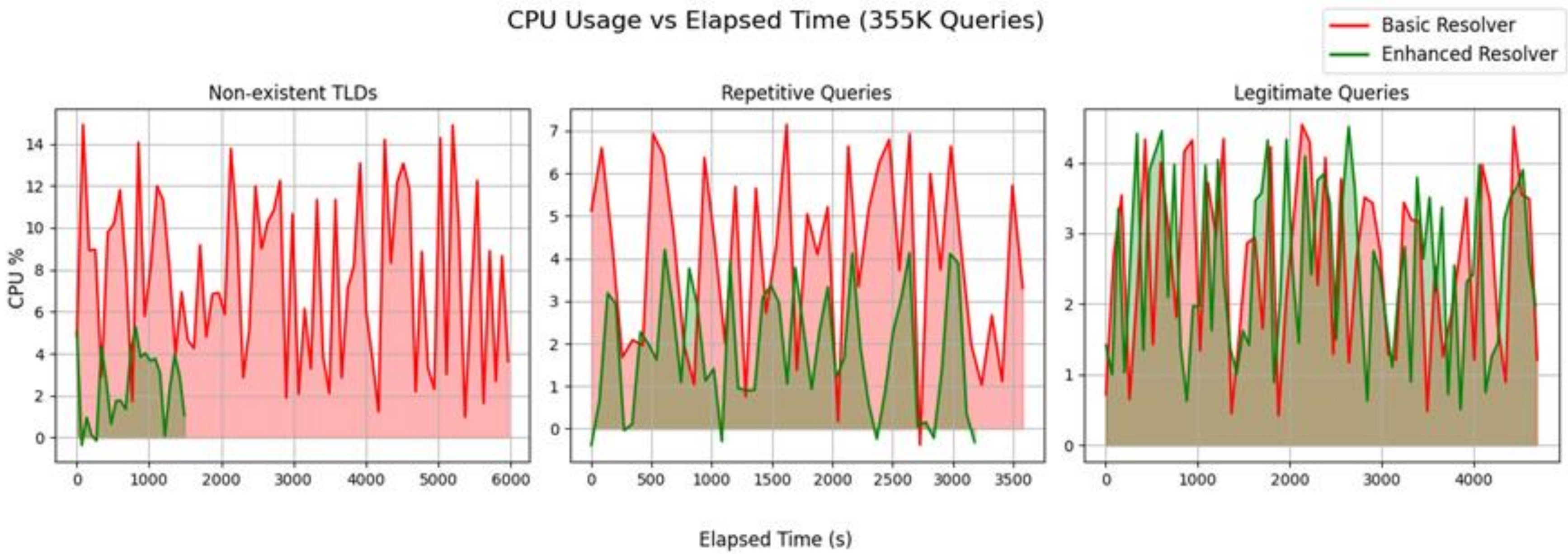
CPU Usage and Latency

- 355K queries over 1h time period
- > 44% decrease in CPU Usage (Enhanced at 10%, Basic at 18%)
- > 55% reduction in latency (Enhanced at 13ms, Basic at 30 ms)



CPU Usage for Different Types of Queries

- > 68% reduction for Non-existent TLDs
- > 42% reduction for Repetitive queries
- Identical for Legitimate queries



Enhanced at 2.56%
Basic at 8%

Enhanced at 2%
Basic at 3.45%

Enhanced at 2.37%
Basic at 2.34%

Memory Consumption

- Resolver in a /16 network
    - 50% increase in memory overhead
    - Enhanced at 36MB and Basic at 24MB (doesn't include memory used for DNS caching)
    - Not a significant increase for modern resolvers

- Memory Overhead: $M(C, T) = M_0 + m \times C \times T$
    - M_0 – Baseline memory (24MB in our case)
    - C – Number of clients (C_0 is 65536)
    - T – Number of TLDs (T_0 is 1500)

- m = 200 bytes to store metadata for each client in our case
    - Proportional to the # of clients a resolver is serving and the # of TLDs

Benefits

- Doesn’t require changes to the DNS architecture
  - As opposed to RFC 8806, Handley et al. (HotNet’04), Allman et al. (HotNets’19)
  - Implementable using open-source DNS software

- Deployed at a Resolver but benefits propagate across DNS ecosystem
  - Root Servers – Receive only 24.34% of current query load (1.8 billion queries)
  - Recursive and Forwarding Resolvers - Reduced CPU load and Query Latency
  - Clients and end users – Enhanced QoE (low latency and timeouts)

Limitations and Challenges

- Experiments in controlled environment using synthetic workloads
    - Doesn’t fully capture the diversity and unpredictability of the real-world DNS
- Results based on implementation in Unbound 1.23
    - Although generalizable, alternate deployments may influence performance
- Quantified Memory overhead using per-source per-TLD pair, but did not Optimize
    - Memory consumption would grow larger for per-source per-domain pairs

Conclusion

- DNS has a lot of junk queries, even today!
- Lightweight, Source-aware Query Filtering Mechanism
- Significantly reduces CPU usage (> 44%) and Latency (> 55%), with minimal Memory overhead (12 MB for 66k clients)

Thank you! Internet Society Pulse

Questions?