Interconnection and Interoperability Mechanism between Handle System and Domain Name System: A Postprint on Implementation Based on Markup Language Description of Protocol Data Units

Authors: Zou Hui, Madi, Wang Wei, Liu Yang, Mao Wei, Shao Qing

Date: 2018-05-20T00:00:00+00:00

Abstract

Based on theoretical and practical considerations, the Handle System and Domain Name System will coexist for a considerable period in the future. However, the incompatibility between the resolution protocols and encoding rules of these two identifier resolution systems prevents data space sharing and information flow, thereby degrading user experience. Consequently, achieving interconnection and interoperation between the two identifier resolution systems represents an urgent issue requiring resolution. Through analysis of the advantages and disadvantages of existing solutions, we find that separating Protocol Data Units from the protocol itself can address the incompatibility of resolution protocols and encoding rules between the two systems. Leveraging this separation mechanism, we design and implement a proxy server-based interconnection and interoperation mechanism between the Handle System and Domain Name System. Experimental results demonstrate that, compared with the traditional client-server model, this mechanism exhibits a small increment ratio in resolution response time across various application scenarios, all within acceptable ranges.

Full Text

Preamble

Inter-operation Mechanism for Handle System and Domain Name System: Implementation Based on Markup Language Describing Protocol Data Unit

Zou Hui[1],[2],[3], Ma Di[3], , Wang Wei[2],[3], , Liu Yang , Mao Wei[2],[3], Shao Qing[3]
(1. Computer Network Information Center, Chinese Academy of Sciences, Bei-

jing 100190, China; 2. University of Chinese Academy of Sciences, Beijing 100049, China; 3. KNET Technologies, Beijing 100190, China; 4. Internet Domain Name System Beijing Engineering Research Center Ltd, Beijing 101400, China; 5. China Academy of Telecommunication Research, Beijing 100191, China)

Abstract: Based on theoretical and practical considerations, the Handle System and Domain Name System (DNS) will coexist for a considerable time. However, the incompatibility between their resolution protocols and encoding rules creates barriers between their data spaces, hindering information flow and degrading user experience. Therefore, achieving interoperability between these two identifier resolution systems is an urgent challenge. By analyzing the advantages and disadvantages of existing solutions, we found that separating the protocol data unit from the protocol itself can resolve this incompatibility. Leveraging this separation mechanism, we designed and implemented a proxy server-based inter-operation mechanism for the Handle System and DNS. Experimental results demonstrate that compared with traditional client-server mode, the increase in resolution response time under this mechanism is small across different application scenarios and remains within acceptable ranges.

Key Words: Handle system; domain name system; inter-operation

0 Introduction

DNS (Domain Name System) [1], invented in 1983, has become a fundamental module for various applications such as HTTP, FTP, and Email due to its simplicity and practicality. However, DNS only names physical devices on the Internet and cannot adequately meet current management requirements for data resource objects. URL (Uniform Resource Locator) [2] provides finer-grained naming services for various types of data resource objects based on DNS technology. Since URLs embed location information into identifiers, they must change when data resource objects migrate; otherwise, they cannot be located.

As Internet data resource objects increase in variety and quantity, redundant storage and nested usage become more frequent, leading to serious storage waste and growing demand for effective resource management. Addressing these needs and recognizing limitations of existing technologies, the Handle System [3] was proposed as an effective, reliable, and scalable global name service system. Its naming objects are more granular than DNS, with different management mechanisms and service models, and it provides permanent identifiers for data resource objects with embedded security features.

However, completely replacing DNS with Handle technology is currently unrealistic both theoretically and practically. First, application layer protocols require operating system support to invoke Handle resolution modules, and achieving this integration will be a long process. Second, although Handle System has

been applied in specific projects, its performance, efficiency, and stability need experimental verification across different application domains. Therefore, Handle System and DNS will coexist for a considerable time due to their technical characteristics and application scopes, making their interoperability both necessary and important.

---

1 Background

1.1 Current Status and Future

Currently, user terminal operating systems only support DNS resolution modules, placing us in a "DNS-only resolution environment." As Handle applications expand, we can envision a future where every user terminal natively supports both DNS and Handle resolution modules, creating a "compatible resolution environment" where Handle System and DNS naturally achieve interoperability. This paper focuses on the current "DNS-only resolution environment" and proposes a solution for Handle-DNS interoperability, providing a good information sharing environment for their coexistence while allowing transition time toward the "compatible resolution environment."

Existing research on Handle-DNS interoperability includes several approaches. Reference [4] builds a Handle-DNS naming service system by mapping name spaces and converting data record formats, storing DNS resource records under a Handle prefix subtree. Reference [5] proposes an IoT heterogeneous identifier system interconnection mechanism at the encoding and resolution protocol level, assigning standard identification codes to each resolution protocol and storing mapping relationships in independent servers. Since current IoT systems mainly use DNS and Handle protocols, this solution can be considered a "superset" of Handle-DNS interoperability. CNRI (Corporation for National Research Initiatives) proposed an industrial solution using a Web server that implements both HTTP and Handle protocols as a proxy. Users input URLs containing Handles and parameters, the Web server interacts with Handle System on their behalf, and displays results as JSON data on web pages.

However, these solutions have limitations. Reference [4] assumes clients and servers both support Handle and DNS protocols and requires dynamic translation of DNS data space, making maintenance time-consuming with poor scalability and difficult deployment. Reference [5] requires clients in a "compatible resolution environment," which doesn' t match current Internet terminal capabilities. CNRI' s proxy solution works in the "DNS-only environment" but is limited to specific applications (browsers) and expression methods (URLs), with limited operations and inability to work in constrained environments without HTTP support.

1.2 Domain Name System and Handle System

DNS is a distributed database mapping domain names to IP addresses, providing convenient Internet access. As a fundamental Internet protocol with long-term stability proven through practice, it remains essential for addressing and location even under attack, and has become an indispensable addressing technology.

The Handle System [6], developed by Robert Kahn' s CNRI in 1995, assigns unique identifiers to Internet data resource objects and provides permanent identification, dynamic resolution, and security management. With strong extensibility and compatibility, it has gained increasing attention across various domains. Like DNS, it is a complete identifier resolution system with naming, registration, and resolution functions, coexisting with DNS as another addressing technology. Therefore, data space interoperability between DNS and Handle System has become an urgent issue [7].

1.3 Interoperability

Within this paper, "interoperability" refers to mutual operation and communication between Handle System and DNS. Mutual operation means user hosts can query any identifier type to either resolution system. Communication means information openness and sharing across different identifier resolution data spaces, making data spaces visible to clients regardless of protocol type.

---

2 Overview Design

This solution deploys a server cluster as a proxy to enable Handle information retrieval in the "DNS-only resolution environment." As shown in Figure 1 [Figure 1: see original paper], for user hosts, the process involves three main steps: first, recognizing the identifier type in application request messages; when the type is Handle, locating the proxy server through host configuration protocols or out-of-band configured location information; second, negotiating transmission protocols (HTTP, CoAP, SIP, etc.) and data formats (JSON, XML, YAML, etc.) with the proxy server; finally, representing Handle request information in the negotiated format and encapsulating it in transmission protocol messages.

Thus, user host operations can be summarized as four functions: identifier recognition, proxy server location, data format conversion, and transmission protocol invocation. Since Handle request information may originate from various applications, this mechanism implements these functions as an independent component that users can optionally install based on their needs. This approach satisfies both application differences and scenario-specific requirements.

For the proxy server, after receiving transmission protocol messages from user hosts and extracting their contents, it must add missing fields to the limited Handle request information from users to generate complete Handle request messages. Correspondingly, when receiving Handle response messages, the proxy

server extracts user-perceivable content, organizes it in the negotiated format, and encapsulates it in transmission protocol messages for response.

Tables 1 and 2 illustrate the phased state information of Handle request and response message transmission, showing how data states change across different implementation entities and actions.

---

3 Detailed Design

This section uses HTTP protocol and JSON markup language as examples for the transmission protocol and data format, describing the implementation details of this mechanism. However, in actual operation, the transmission protocol and markup language are not limited to specific combinations but depend on the supported capabilities of user hosts and proxy servers.

3.1 JSON Markup Language and HTTP Protocol

JSON [8] is a lightweight data exchange format with good readability and rapid writability, suitable for data exchange processing. As a "description tool" for Handle data, it depicts user-input Handle raw data according to Handle protocol message structure, forming JSON-formatted Handle data.

HTTP [9], as a member of the TCP/IP protocol suite, has become a fundamental Internet protocol carrying most Internet traffic, with default support in user host operating systems. As a content-oriented application layer protocol, it can carry diverse data object types, providing flexibility for format selection and simplifying protocol compatibility design details such as identifier encoding conversion, PDU adaptation, and data record format conversion. Additionally, most firewalls, proxies, and control devices open port 80, allowing HTTP messages to pass through smoothly. As a "transmission tunnel" for Handle data, it provides a transmission path [10] for formatted encapsulated data.

3.2 Component (User Host)

Applications serve as the entry point for user Handle data input, while the component processes unstructured Handle data and organizes it into JSON format. For PDU adaptation efficiency, JSON format data is designed based on Handle message structure [11], which consists of four parts: envelope, header, body, and credential, each with specific functions. The envelope guides Handle message transmission, the header guides each Handle operation' s specifics, the body contains request/response data content, and the credential detects message tampering during transmission.

Limited by their "Handle knowledge," users cannot input certain fields before Handle message generation, such as envelope parts and certain header/credential fields, which can only be filled by Handle protocol entities (the proxy server in

this paper). Therefore, Handle message content is filled by both users and proxy servers, while Handle message generation is performed only by the proxy server.

User hosts (the component) can only perceive partial Handle request/response information including Header, Body, and Credential, corresponding to the header, body, and credential parts of Handle messages. Each Handle request/response message is defined as a JSON object. User and proxy server responsibilities for fields are shown in Figure 2 [Figure 2: see original paper], with gray areas indicating user fields and white areas indicating proxy server fields.

For Handle request message JSON objects, besides the Body structure being determined by OpCode and ResponseCode (actually determined by OpCode alone, with ResponseCode set to 0 for all requests), Header and Credential structures are relatively fixed, each represented as a JSON object. Figures 3 [Figure 3: see original paper] and 4 [Figure 4: see original paper] illustrate the JSON object data structures for Header and Credential sections.

Table 3 shows user-controlled Handle message fields including OpCode (operation type), ResponseCode (server response), flags for primary site processing, signature requirements, encryption requirements, cache validation, connection persistence, permission filtering, and digest inclusion. Signer information and signature details are also user-controlled.

A concrete example demonstrates a user initiating a Handle resolution request, specifying processing by Primary Site Handle servers and returning only Public_Read permission data records, with cache server validation required. The user signs the Handle request data to declare identity for proxy server authentication. The session key is stored in Handle 200/5555 index 2, using SHA-A signature algorithm on content "shjdkwjhgdna2ixnmdmsnbdjhd."

For Handle response message JSON objects, besides Body structure being determined by OpCode and ResponseCode, Header and Credential structures are fixed, identical to request messages (Figures 3 and 4).

3.3 Proxy Server

As a "transfer and processing station" for Handle messages, the proxy server completes a Handle resolution/management operation through two interactions: with user hosts and with Handle System. After receiving HTTP messages on port 80, the proxy server extracts JSON data and fills missing fields according to Figure 2 to build complete Handle request messages. Conversely, after receiving Handle response messages, the proxy server converts them to JSON format and encapsulates them in HTTP messages for response. During this process, opposite to the component' s function, the proxy server discards uninteresting information such as fields guiding Handle response transmission.

Due to its special position, the proxy server, as the user host' s agent in identifier resolution, comprehensively masters each Handle request operation and corre-

sponding response data. This enables it to function as a public cache server, storing each Handle request/response to rapidly respond to subsequent requests for the same prefix or Handle, improving user experience while reducing Handle System load. As a "communication bridge" between user hosts and Handle System, it splits end-to-end communication into two connections (user host to proxy server and proxy server to Handle System) that cooperate transparently to users. Therefore, the proxy server must maintain state mapping tables containing session ID, session keys, and request IDs to correctly match the two connections belonging to the same communication.

Introducing a third party into end-to-end communication creates security concerns, as the proxy server becomes a "man-in-the-middle." Section 5 addresses this in detail.

According to RFC3652 Handle System Protocol (ver2.1) Specification, there are 14 request operation types and 31 response result types. Handle message content is determined by the <OpCode, ResponseCode> combination. Analysis shows Handle request Body has 6 data structures and response Body has 8 data structures.

For user-initiated Handle resolution operations, responses contain one or more data records associated with the Handle. Handle data records are either custom or predefined. Predefined records are for management operations, while custom records can be registered under the 0.NA/0.TYPE prefix. Both share a common data structure that can be JSON-formatted. Figure 5 [Figure 5: see original paper] shows a URL data record example containing seven fields: index, type, data, TTL, permissions, timestamp, and reference, with details in RFC3651 [12].

---

4 Experiments

4.1 Experimental Design

This experiment involves three entity types: user hosts, proxy servers, and Handle servers, with parameters shown in Table 4 . Considering different proxy application scenarios, we designed two experiment groups: the first group simulates public proxy service environment using multi-process; the second group simulates private proxy service environment using multi-threading. To evaluate optimization effects, each group has a corresponding comparison experiment with direct client-server Handle resolution. To avoid environmental inconsistencies, user host and Handle server parameters remain consistent across both groups.

Public proxy maximum concurrent queries are set to 50,000, while private proxy is set to 10,000, reflecting their different workload characteristics.

4.2 Experimental Results

Considering Handle System's primary function is providing Handle identifier resolution services, this experiment focuses on resolution operations and compares response time between proxy-based and traditional direct connection mechanisms.

4.2.1 Public Proxy Public proxy serves unrestricted clients, so parsed Handles are random with weak correlation. Processes, as minimum resource allocation units with independent data spaces and CPUs, simulate public proxy environments where resolution requests are independent, completing full GHR (Global Handle Registry) → LHS (Local Handle Service) → Handle query flows.

This experiment uses 20 processes to simulate user resolution requests, comparing concurrent resolution time efficiency between direct and proxy-based Handle resolution. Total time from first request to last response is measured to calculate average resolution response time. Nine experiment runs are conducted with incrementally increasing request volumes, as shown in Figure 6 [Figure 6: see original paper].

Figure 6 shows average resolution response times for both methods fluctuate within a range, with proxy method generally higher than direct method. However, the incremental proportion is small and negligible.

4.2.2 Private Proxy Private proxy serves specific organizations (e.g., factories, logistics companies), so parsed Handles share prefixes or strong access correlations. Threads within a process share resources (CPU, data space), simulating private proxy environments where requests have dependencies. Subsequent requests can reuse LHS information from the first GHR query, omitting repeated GHR lookups.

This experiment uses 20 threads to simulate user requests, measuring time from second request to last response to calculate average resolution response time (excluding the first request due to dependency differences). Seven experiment runs are conducted with incrementally increasing request volumes, as shown in Figure 7 [Figure 7: see original paper].

Figure 7 shows proxy method average response time is generally higher than direct method, but gradually converges as query volume increases. The incremental time is only a few milliseconds. Figures 6 and 7 also show that in public proxy environments, due to weak Handle correlations, more time is spent querying GHR for LHS information compared to private proxy environments.

5 Security and Performance

Handle System has embedded security features providing end-to-end encryption and signature authentication. Signatures provide non-repudiation by authenticating identities, while encryption ensures data privacy and integrity. The proxy server' s functional characteristics make it a "man-in-the-middle" that must interpret and reconstruct Handle queries and responses. Since end-to-end encryption and signing only protect protocol data units, the proxy becomes a trusted intermediary that can sniff all communication content, making end-to-end security unenforceable and introducing potential security risks.

According to RFC 3651 Section 4.2.2 [10], proxy servers are not part of Handle System' s management or verification scope—users do not validate Handle response data from proxy servers. Whether users trust proxy-returned data depends on established trust relationships, which users control based on their policies. Users must balance security and availability: trusting public proxies introduces security risks, while not trusting them sacrifices availability.

Proxy servers can be public or private. Public proxies, like public DNS recursive resolvers (e.g., Google' s 8.8.8.8), serve unrestricted clients without service constraints, creating trust establishment challenges. Users must evaluate factors like reputation, location, and maintainer information to decide whether to trust a public proxy.

Private proxies, like organizational internal DNS resolvers, serve only internal members within a trust domain. In industrial applications, organizations typically form closed application loops and trust domains where internal entities mutually trust each other. Therefore, private proxy trust relationships are established by default without user configuration.

As a critical connection bond in the interoperability system, proxy servers handle numerous tasks (message extraction, PDU adaptation, mapping maintenance) and can become performance bottlenecks. They are also attack targets (e.g., DDoS), which can increase latency or cause service failure. For security and performance, proxy server clusters with load balancing algorithms or BGP+Anycast can prevent overload and single-point failures.

---

6 Conclusion

To address data space sharing issues between Handle System and DNS caused by incompatible resolution protocols, this paper analyzed three existing solutions' pros and cons, designed a proxy server-based protocol-data separation mechanism, and provided implementation details. We analyzed proxy application scenarios, security and performance issues, and conducted experiments for different proxy types. Results show that in public proxy environments, the average response time difference from direct connection is negligible. In private

proxy environments, the average resolution response time increment is acceptable for small batches, and converges toward direct connection time (only a few milliseconds difference) as query volume increases.

With further Handle adoption in industrial domains and understanding of organizational structures and resolution operation characteristics—where organizations form closed application loops and item queries are batch-oriented—this scheme achieves Handle-DNS interoperability without significantly reducing resolution efficiency, demonstrating good applicability.

References

[1] Mokapetris P. Domain names: concepts and facilities, RFC1034 [R]. 1987.
[2] Berners-Lee T, Masinter L, McCahill M. Uniform resource Locators (URL), RFC 1738 [R]. 1994.
[3] Sun S, Lannom L, Boesch B. Handle system overview, RFC 3650 [R]. 2003.
[4] Wang Feng, Sun Xun, Mao Wei, et al. Handle-DNS naming service system design and implementation [J]. Microelectronics & Computer, 2004, 21(1): 39-44.
[5] Kong Ning, Shen Shuo, Liu Bing, et al. An IoT heterogeneous identifier resolution method and system: China, 2013104076132 [P]. 2013-09-09.
[6] Guo Xiaofeng, Sun Xun. Development and application of Handle system [J]. Digital Library Forum, 2013, (8): 18-24.
[7] Kahn R, Maffei A. Terminology and use cases for interoperability of identifier resolution systems [EB/OL]. draft-kahn-dsii-id-res-sys-00, 2012.
[8] Bray T. The JavaScript object notation (JSON) data interchange format, RFC7159 [R]. 2014.
[9] Fielding R, Gettys J, Mogul J, et al. Hypertext transfer protocol–HTTP/1.1, RFC2616 [R]. 1996.
[10] Popa L, Ghodsi A, Stoica I. HTTP as the narrow waist of the future Internet [C]// Proc of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. 2010.
[11] Sun S, Reilly S, Lannom L, et al. Handle system protocol (ver 2.1) specification, RFC 3652 [R]. 2003.
[12] Sun S, Reilly S, Lannom L. Handle system namespace and service definition, RFC 3651 [R]. 2003.

Note: Figure translations are in progress. See original paper for figures.