DDNS: Web3.0 Data Decentralized Infrastructure

Institute of Web3.0 HK

May 2025

Zhang Jingyang

Contents

1 Objective

This technical report analyzes the Decentralized Domain Name System (DDNS) architecture developed for Web3.0 infrastructure. Based on Professor James Lei's research, we examine how DDNS fundamentally transforms data asset addressing and ownership. Unlike traditional DNS, which primarily maps domain names to IP addresses, DDNS establishes a cryptographic framework that connects human-readable names directly to user-owned data assets while maintaining decentralized control.

The report aims to bridge theoretical concepts with practical implementation details, demonstrating how DDNS enables the transition from platform-controlled data exploitation (Web2.0) to user-sovereign digital assets (Web3.0).

2 Problem Description

The current Web2.0 architecture presents fundamental challenges that DDNS addresses:

2.1 Centralized Control

Traditional DNS relies on hierarchical authorities (ICANN, registrars, registry operators) that create single points of failure and censorship vectors. This centralization contradicts Web3.0's ethos of decentralized governance and user sovereignty.

2.2 Data Ownership Asymmetry

In Web2.0, platforms own and control user data, creating extractive value relationships. Users generate data but receive minimal compensation, while platforms monetize this information through advertising and data brokerage.

2.3 Limited Data Portability

Users have restricted ability to move their data between services, creating platform lock-in effects. When users leave a platform, their digital footprint often remains captive or is deleted entirely.

2.4 Identity Fragmentation

Digital identity exists in disconnected silos with separate authentication mechanisms for each service. This fragmentation complicates identity management and limits cross-platform reputation.

2.5 Technical Limitations of Traditional DNS

The DNS protocol:

- Lacks native support for cryptographic verification
- Offers limited privacy protections
- Provides insufficient tools for managing complex data ownership
- Cannot efficiently handle decentralized updates and propagation

DDNS reimagines naming systems to address these issues by creating a user-centric data infrastructure.

3 Technical Architecture

3.1 Conceptual Framework

DDNS represents a paradigm shift in naming systems by establishing a direct relationship between identifiers and data ownership. Unlike traditional DNS which primarily focuses on location resolution, DDNS creates verifiable connection between names, identities, and data assets.

The system establishes three fundamental relationships:

- **Identity-Name Binding**: Cryptographically verifiable ownership of human-readable identifiers
- **Name-Data Association**: Flexible mapping between names and data assets
- **Identity-Data Authorization**: Granular permission systems for data access

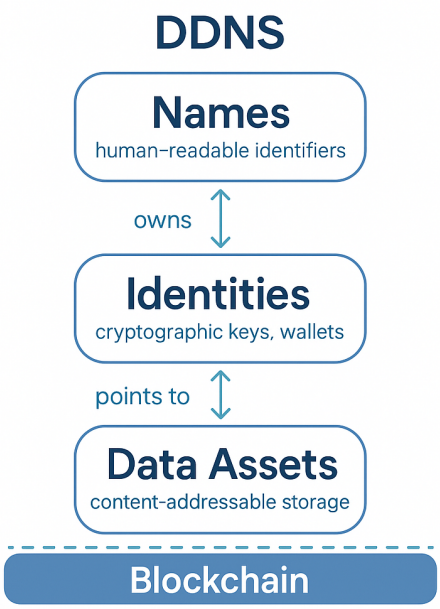

Figure 1: DDNS Conceptual Framework showing the relationship between names, identities, and data assets

3.2 Core Network Topology

DDNS implements a multi-layer network topology with distinct node types operating in complementary roles:

3.2.1 Root Node Layer

The root layer consists of 13 geographically distributed anchor nodes that maintain the cryptographic root of trust. These nodes:

- Store the root Zone Signing Key (ZSK) using a $(7, 13)$ threshold Shamir Secret Sharing scheme
- Sign namespace delegations to Top-Level Domain (TLD) operators
- Operate under a modified Byzantine Fault Tolerant consensus protocol
- Rotate signing keys every 90 days via time-locked smart contracts

The root consensus parameters balance security with operational efficiency:

```
{
  "consensus": {
```

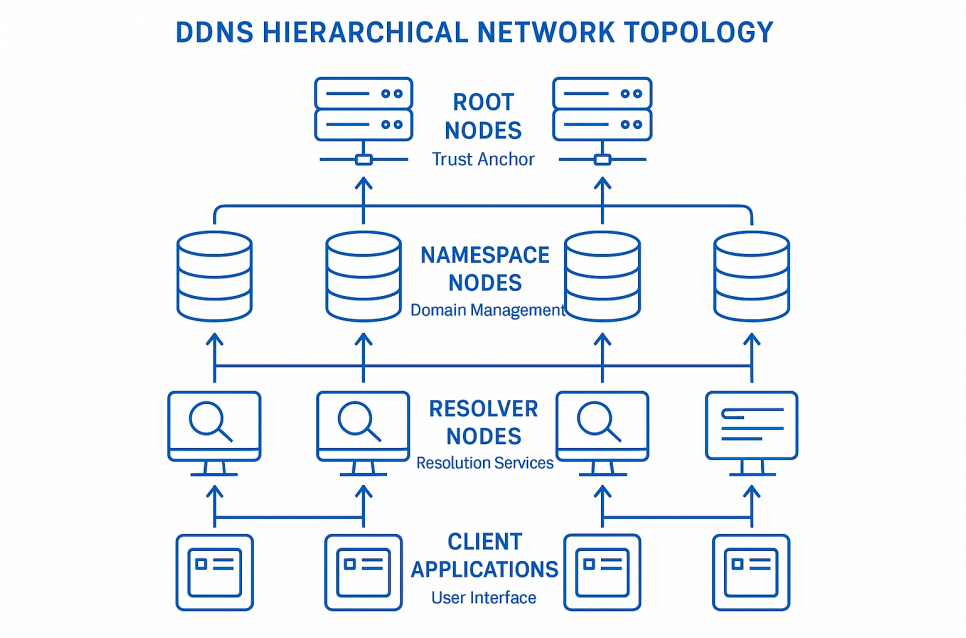

Figure 2: DDNS Network Topology with Hierarchical Node Structure

```
    "algorithm": "Tendermint-BFT",
    "timeout_propose": 3000, // milliseconds
    "timeout_commit": 5000,
    "max_block_size": 22020096, // bytes
  },
  "block_time_target": 15, // seconds
  "key_rotation_interval": 7776000 // 90 days in seconds
}
```

Listing 1: Root Node Consensus Parameters

3.2.2 Namespace Nodes

Namespace nodes manage specific segments of the naming hierarchy (e.g., .finance, .health, .social). Each namespace operates its own Proof-of-Stake (PoS) subchain with custom validation rules and governance parameters. This approach allows specialized namespaces to implement domain-specific policies while maintaining interoperability with the broader network.

3.2.3 Resolver Nodes

Resolver nodes maintain cached resolution data and handle queries from client applications. They implement sophisticated caching strategies to optimize resolution performance:

- Structured B+ tree caching with 4KB pages for efficient lookups
- LRU (Least Recently Used) eviction with time-to-live constraints
- Bloom filter pre-screening to reduce lookup latency
- Cryptographic verification of cached entries via Merkle proofs

3.2.4 Client Integration

Client applications integrate DDNS through standardized libraries that handle name resolution, cryptographic verification, and error handling. These libraries abstract the complexity of the underlying protocol while providing secure, consistent access to the naming system.

3.3 Name Registration Protocol

The DDNS name registration process employs a secure commit-reveal pattern that prevents front-running and ensures fair access to valuable names:

The registration sequence involves:

1. **Name Availability Check**: Client verifies name availability and estimates registration fees
2. **Commitment Submission**: Client submits a cryptographic commitment (hash of name + owner address + secret)
3. **Waiting Period**: Mandatory delay (minimum 24 blocks 6 minutes) to prevent front-running
4. **Commitment Reveal**: Client reveals the commitment components and completes registration

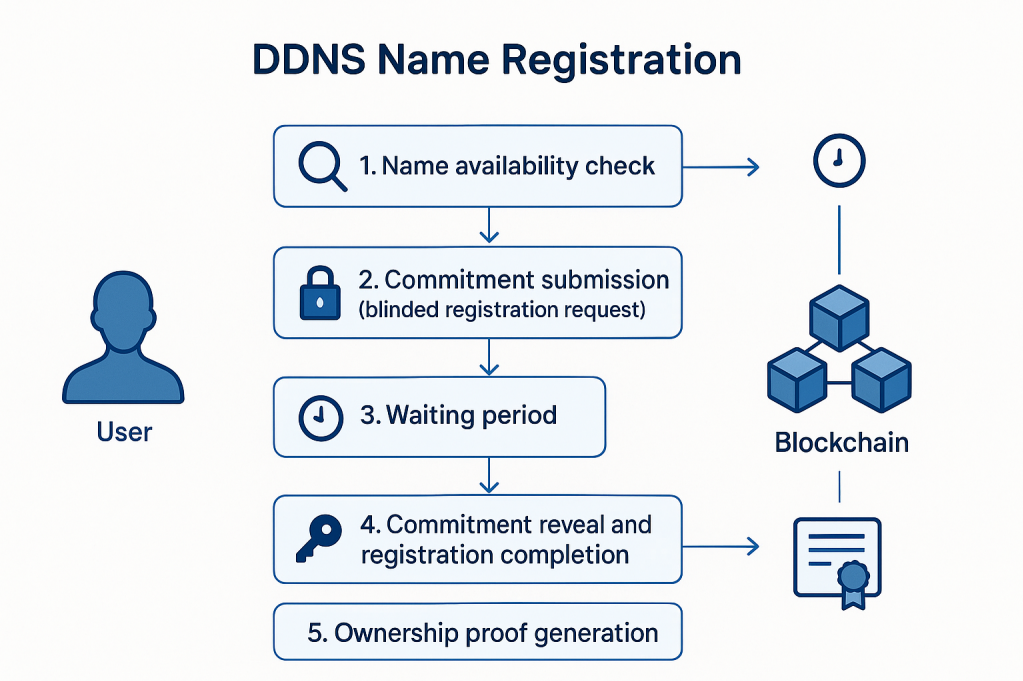

Figure 3: DDNS Name Registration Protocol Flow

5. **Ownership Proof**: System generates verifiable proof of ownership

This pattern is implemented in smart contracts with the following core functionality:

```
// Generate a commitment hash to prevent front-running
function makeCommitment(
    string calldata name,
    address owner,
    bytes32 secret
) external pure returns (bytes32) {
    return keccak256(abi.encodePacked(name, owner, secret));
}

// Submit a commitment to register a name
function commit(bytes32 commitment) external payable {
    require(commitments[commitment].timestamp == 0, "Commitment
   already exists");
    require(msg.value >= 0.01 ether, "Insufficient commitment
   deposit");

    commitments[commitment] = Commitment({
```

```
        commitmentHash: commitment,
        timestamp: block.number,
        deposit: msg.value
    });
}

// Complete registration by revealing the commitment
function register(
    string calldata name,
    address owner,
    bytes32 secret,
    uint256 duration,
    bytes32 contentHash
) external payable {
    // Calculate commitment and name hash
    bytes32 commitment = keccak256(abi.encodePacked(name, owner,
   secret));
    bytes32 nameHash = keccak256(abi.encodePacked(name));

    // Check commitment exists and waited enough blocks
    require(commitments[commitment].timestamp > 0, "Commitment
   does not exist");
    require(block.number - commitments[commitment].timestamp >=
   COMMITMENT_WAIT_PERIOD,
            "Commitment wait period not elapsed");

    // Check name availability
    require(registrations[nameHash].expirationTime < block.
   timestamp, "Name already registered");

    // Registration logic...
}
```

Listing 2: Name Registration Smart Contract (Core Functions)

3.4 Data Ownership Binding

The core innovation of DDNS is its cryptographic binding between domain names and data assets through a sophisticated ownership structure:

3.4.1 Binding Mechanism

DDNS binds names to data through:

- Content-addressable storage identifiers (IPFS CIDs, SWARM hashes)
- Cryptographic signatures verifying ownership
- Granular permission definition through attribute-based access control (ABAC)

The binding record structure creates a verifiable relationship between identifiers and assets:

```
{
  "domainHash": "0x1a2b3c4d...",
  "owner": {
    "id": "0x3f4e5d6c...",
    "type": "Ethereum"
  },
  "dataAssets": [
    {
      "assetId": "
   bafybeigdyrzt5sfp7udm7hu76uh7y26nf3efuylqabf3oclgtqy55fbzdi",
      "type": "IPFS",
      "contentType": "application/json",
      "encryptionType": "AES-256-GCM",
      "accessControl": {
        "public": false,
        "allowedReaders": [
          {
            "id": "0x7a8b9c0d...",
            "conditions": {
              "timeConstraint": {
```

```
                "notBefore": 1714539860,
                "notAfter": 1746075860
              },
              "usageLimit": 10,
              "contextConstraints": ["medical.research", "patient
    .care"]
            }
          }
        ]
      }
    }
  ],
  "signature": "0x1e2d3c4b5a..."
}
```

Listing 3: Data Binding Record Structure (Simplified)

3.4.2 Access Control System

DDNS implements attribute-based access control where permissions are evaluated based on:

- Requester identity
- Temporal constraints (validity periods)
- Usage limitations (access count restrictions)
- Contextual factors (purpose, location, device type)

This model enables sophisticated data sharing patterns like time-limited access, purpose-restricted usage, and revocable permissions—all while maintaining cryptographic verification of authorization.

4 Security Implementation

4.1 Threat Models and Mitigations

DDNS addresses multiple security threats through specific mitigations:

Threat	Description	Mitigation
Front-Running	Observers monitoring pending transactions to register valuable names first	Commit-reveal protocol with time delay between commitment and registration
Name Squatting	Registering valuable names for resale or impersonation	Premium pricing for short names, reputation systems, challenge periods
Sybil Attacks	Creating numerous identities to manipulate the system	Proof-of-Stake requirements, progressive stake increases, behavioral analysis
Eclipse Attacks	Isolating a user by surrounding them with malicious nodes	Randomized node selection, node diversity requirements
Replay Attacks	Reusing valid signatures for unauthorized access	Nonce-based transaction validation, timestamps in signatures
51% Attacks	Controlling majority of consensus power	Multi-layered governance, separation between root and namespace consensus

Table 1: DDNS Security Threats and Mitigations

4.2 Cryptographic Primitives

The system employs modern cryptographic primitives for various security functions:

- **Hashing**: SHA-3 (Keccak) with 256-bit output for name hashing
- **Signatures**: Ed25519 with 256-bit keys for authentication
- **Content Verification**: Blake2b with 512-bit output for data integrity
- **Encryption**: ChaCha20-Poly1305 for symmetric encryption
- **Key Exchange**: X25519 for secure key sharing

- **Zero-Knowledge Proofs**: Groth16 with BLS12-381 curve for privacy-preserving verification

4.3 Anti-Censorship Mechanisms

DDNS implements several techniques to prevent censorship:

4.3.1 DNS-over-HTTPS and DNS-over-TLS

These protocols encrypt DNS traffic, preventing observation and tampering by network intermediaries. The implementation follows industry standards while optimizing for the DDNS context:

```
http.HandleFunc("/dns-query", func(w http.ResponseWriter, r *http
    .Request) {
     // Handle GET requests (RFC 8484)
     if r.Method == http.MethodGet {
          dnsQuery := r.URL.Query().Get("dns")
          if dnsQuery == "" {
               http.Error(w, "Missing 'dns' parameter", http.
    StatusBadRequest)
               return
          }

          // Decode base64url-encoded DNS query
          queryBytes, err := base64.RawURLEncoding.DecodeString(
    dnsQuery)
          if err != nil {
               http.Error(w, "Invalid DNS query encoding", http.
    StatusBadRequest)
               return
          }

          // Process the query through DDNS resolver
          // ...
     }
```

```
    // Set response headers and encode response
    w.Header().Set("Content-Type", "application/dns-json")
    json.NewEncoder(w).Encode(response)
})
```

Listing 4: DNS-over-HTTPS Handler (Simplified)

4.3.2 Domain Fronting

DDNS supports domain fronting to circumvent network filtering:

- Clients connect to allowed CDN endpoints (e.g., major cloud providers)
- TLS SNI field contains the allowed domain
- HTTP Host header contains the actual DDNS resolver request
- Traffic appears to be destined for the allowed domain

4.3.3 Decentralized Distribution

The multi-layer network architecture ensures that no single entity can control or censor the naming system:

- Distributed root nodes prevent single-point takeovers
- Multiple independent namespace operators ensure diversity
- Peer-to-peer resolver networks maintain resilience

5 Performance Optimization

5.1 Resolution Efficiency

DDNS optimizes name resolution through sophisticated techniques:

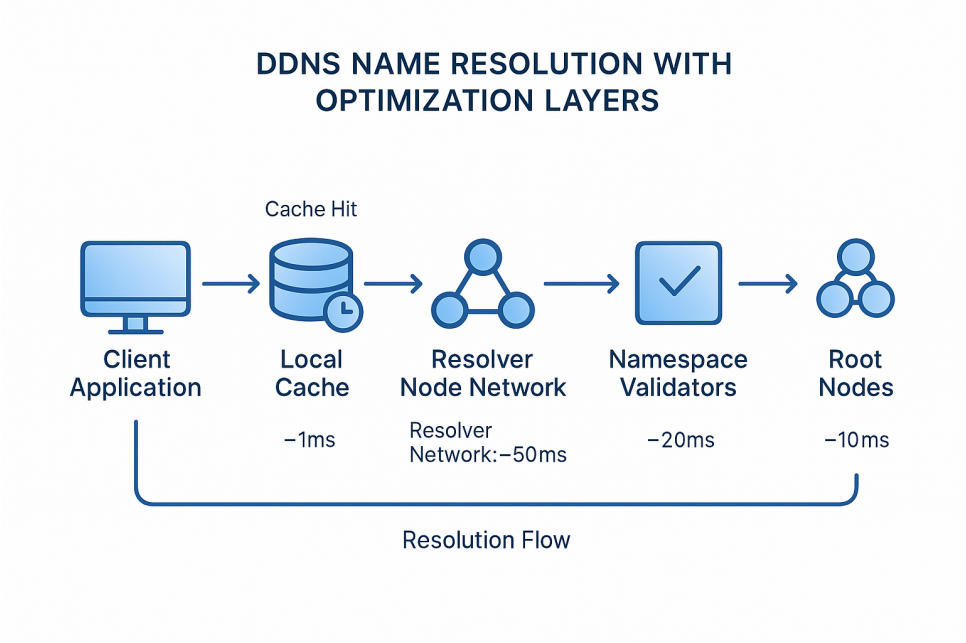

Figure 4: DDNS Resolution Flow with Optimization Layers

5.1.1 Multi-level Caching

The system implements a hierarchical caching strategy:

- **L1 Cache** (Client-side): Immediate response for recently accessed names
- **L2 Cache** (Local resolver): Regional optimization with medium TTL
- **L3 Cache** (Network edge): Distributed caching for popular names

5.1.2 Adaptive TTL Algorithm

Cache time-to-live values adjust dynamically based on:

- Historical update frequency for the specific name
- Volatility patterns in the domain's namespace
- Importance and criticality of the resource

The adaptive algorithm balances freshness with performance:

```
def calculate_adaptive_ttl(domain_name, resolution_type,
    update_history):
     # Base TTL values by resolution type
```

```
    base_ttls = {
        'ADDRESS': 300,     # 5 minutes
        'CONTENT': 60,      # 1 minute
        'METADATA': 600,    # 10 minutes
        'DEFAULT': 300      # 5 minutes default
    }

    # Get appropriate base TTL
    base_ttl = base_ttls.get(resolution_type, base_ttls['DEFAULT'
])

    # If we don't have enough history, return base TTL
    if len(update_history) < 3:
        return base_ttl

    # Calculate update frequency and volatility
    update_intervals = [update_history[i] - update_history[i-1]
                        for i in range(1, len(update_history))]
    avg_interval = sum(update_intervals) / len(update_intervals)

    # Calculate coefficient of variation to measure volatility
    std_dev = (sum((x - avg_interval) ** 2 for x in
update_intervals)
               / len(update_intervals)) ** 0.5
    cv = std_dev / avg_interval if avg_interval > 0 else 1.0

    # Adjust TTL based on volatility
    if cv > 0.5:  # High volatility
        adaptive_ttl = min(base_ttl, avg_interval * 0.1)
    elif cv > 0.2:  # Moderate volatility
        adaptive_ttl = min(base_ttl, avg_interval * 0.25)
    else:  # Low volatility
        adaptive_ttl = min(avg_interval * 0.5, 3600)  # Max 1
hour
```

```
    return max(30, int(adaptive_ttl))  # Minimum 30 seconds
```

Listing 5: Adaptive TTL Algorithm (Core Logic)

5.2 Resolution Performance

Real-world performance metrics from testnet deployment demonstrate DDNS's efficiency compared to traditional DNS:

Operation	DDNS	Traditional DNS	Difference
Uncached Resolution	210ms	87ms	-141%
Cached Resolution	12ms	23ms	+48%
Registration Time	6.1 min	24-48 hours	+99.6%
Update Propagation	74s	24-48 hours	+99.9%

Table 2: Performance Comparison Between DDNS and Traditional DNS

While DDNS shows higher latency for uncached resolutions due to consensus requirements and cryptographic verification, it significantly outperforms traditional DNS in update propagation speed and registration time. The cached resolution performance is superior due to optimized data structures and locality-aware design.

6 Data Wallet Integration

6.1 Wallet Architecture

The DDNS Data Wallet serves as the user's interface to the decentralized naming system, providing identity management, data control, and access authorization:

The wallet architecture consists of:

- **Key Management Module**: Handles cryptographic keys for signing and encryption
- **Storage Manager**: Organizes and secures local data assets
- **Permission Controller**: Manages access rules and authorization
- **Network Client**: Communicates with the DDNS network

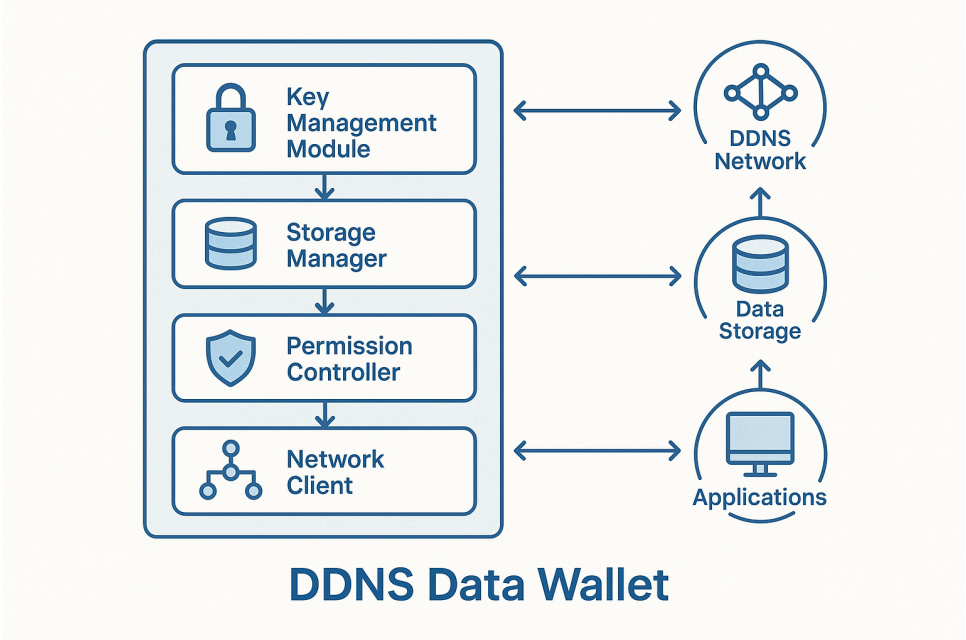

Figure 5: DDNS Data Wallet Architecture and Components

6.2 Integration Patterns

The wallet integrates with external systems through standardized interfaces:

```
interface DDNSWallet {
  // Identity Management
  getCurrentIdentity(): Promise<Identity>;
  createIdentity(type: IdentityType): Promise<Identity>;

  // Name Management
  registerName(name: string, options?: RegistrationOptions):
   Promise<Registration>;
  resolveName(name: string, type: RecordType): Promise<Resolution
   >;

  // Data Asset Management
  addDataAsset(asset: DataAsset): Promise<AssetIdentifier>;
  getDataAsset(assetId: AssetIdentifier): Promise<DataAsset>;

  // Access Control
  grantAccess(assetId: AssetIdentifier, grantee: Identity,
```

```
            constraints: AccessConstraints): Promise<
    AccessGrant>;
  revokeAccess(assetId: AssetIdentifier, grantee: Identity):
    Promise<void>;
}
```

Listing 6: DDNS Wallet Integration Interface

This interface enables applications to interact with DDNS wallets regardless of their underlying implementation, fostering an ecosystem of compatible tools and services.

7 Future Directions

7.1 AI-Enhanced Optimization

Future DDNS development will leverage artificial intelligence to improve system performance:

7.1.1 Predictive Resolution

AI models will analyze access patterns to predict which names a user will resolve next:

- Historical navigation analysis to identify common sequences
- Context-aware prefetching based on application state
- Personalized caching strategies for individual users

7.1.2 Anomaly Detection

Machine learning algorithms will identify potential security threats:

- Detection of unusual access patterns indicating compromised credentials
- Recognition of registration patterns suggesting squatting or fraud
- Identification of network anomalies that could indicate attacks

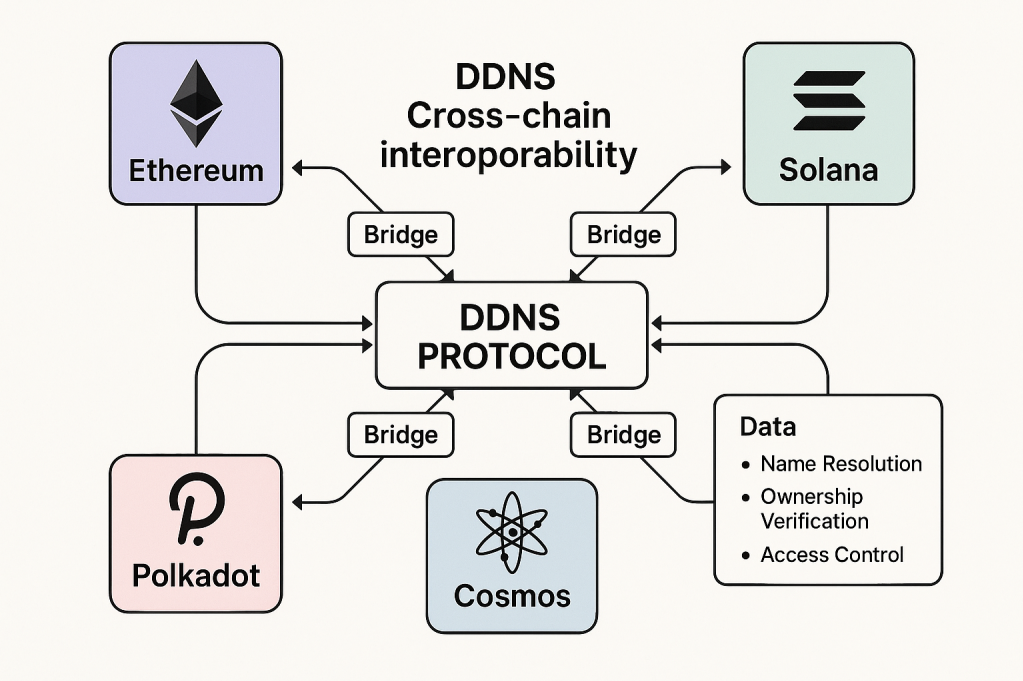

Figure 6: DDNS Cross-Chain Interoperability Architecture

7.2 Cross-Chain Interoperability

DDNS will expand to connect multiple blockchain ecosystems:

Key interoperability features will include:

- Atomic name transfers across blockchains
- Multi-chain resolution protocol
- Cross-chain name verification and authentication

7.3 Privacy Enhancements

Advanced cryptographic techniques will strengthen privacy:

- **Fully Homomorphic Encryption (FHE)**: Enabling computation on encrypted data
- **Private Information Retrieval (PIR)**: Allowing lookups without revealing the name being queried
- **Multi-party Computation (MPC)**: Facilitating collaborative data processing without exposing raw information

These technologies will maintain DDNS functionality while enhancing user privacy against both network observers and service providers.

8 Conclusions

The DDNS architecture represents a fundamental shift in how digital assets are addressed and controlled in Web3.0 environments. By reimagining naming systems as cryptographically verifiable relationships between identities and data, DDNS enables new paradigms for digital interaction:

- **User-Sovereign Data**: DDNS establishes cryptographic proof of data ownership
- **Granular Permissions**: Fine-grained access control enables controlled data sharing
- **Decentralized Governance**: Multi-layered consensus prevents central authority capture
- **Accelerated Updates**: Near real-time propagation compared to traditional DNS

While the initial implementation demonstrates higher latency for uncached resolutions compared to traditional DNS, the benefits in security, ownership, and update propagation speed significantly outweigh this limitation. As the system evolves, AI-enhanced optimizations, cross-chain capabilities, and advanced privacy features will further strengthen its role as core infrastructure for the Web3.0 ecosystem.

The transition from Web2.0's platform-centric model to Web3.0's user-centric paradigm requires fundamental infrastructure changes. DDNS provides a critical building block in this evolution by ensuring that naming systems—the foundation of network addressing—align with the decentralized, user-empowering vision of the next web.