Application-Network Coordination and Integration

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs434/

05/05/2021
Admin

- Projects update
  - Week 5 (Apr 30 - May 6): Refinement; iterations
    - Week 6 (May 7 - May 13): Final implementation, final report (6-8 pages)

- Rescheduled class time
  - Monday regular class time?
Recap: Raft/etcd Protocol and Use

Raft Protocol Summary

**Followers**
- Respond to RPCs from candidates and leaders.
- Convert to candidate if election timeouts elapse without either:
  - Receiving valid AppendEntries RPC, or
  - Granting vote to candidate

**Candidates**
- Increment currentTerm, vote for self
- Reset election timeout
- Send RequestVote RPCs to all other servers, wait for either:
  - Votes received from majority of servers: become leader
  - AppendEntries RPC received from new leader: step down
- Election timeout elapses without election resolution:
  - Increment term, wait random time, start new election
  - Discover higher term: step down

**Leaders**
- Initialize nextIndex for each to last log index + 1
- Send initial empty AppendEntries RPCs (heartbeat) to each follower; repeat during idle periods to prevent election timeouts
- Accept commands from clients, append new entries to local log
- Whenever last log index ≥ nextIndex for a follower, send AppendEntries RPC with log entries starting at nextIndex, update nextIndex if successful
- If AppendEntries fails because of log inconsistency, decrement nextIndex and retry
- Mark log entries committed if stored on a majority of servers and at least one entry from current term is stored on a majority of servers
- Step down if currentTerm changes

**AppendEntries RPC**
- Invoked by leader to replicate log entries and discover inconsistencies; also used as heartbeat.

**Persistent State**
- Each server persists the following to stable storage synchronously before responding to RPCs:
  - currentTerm: latest term server has seen (initialized to 0 on first boot)
  - votedFor: candidateId that received vote in current term (or null if none)
  - log[]: log entries

**Log Entry**
- term: term when entry was received by leader
- index: position of entry in the log
- command: command for state machine

RequestVote RPC
- Invoked by candidates to gather votes.

**Arguments**
- candidateId: candidate requesting's vote
- term: candidate's term
- lastLogIndex: index of candidate's last log entry
- lastLogTerm: term of candidate's last log entry

**Results**
- term: currentTerm, for candidate to update itself
- voteGranted: true means candidate received vote

**Implementation**

AppendEntries RPC
- Invoked by leader to replicate log entries and discover inconsistencies.

**Arguments**
- term: leader's term
- leaderId: so follower can redirect clients
- prevLogIndex: index of log entry immediately preceding new ones
- prevLogTerm: term of prevLogIndex entry
- entries[]: log entries to store (empty for heartbeat)
- commitIndex: last entry known to be committed

**Results**
- term: currentTerm, for leader to update itself
- success: true if follower contained entry matching prevLogIndex and prevLogTerm

**Implementation**

Leaders
1. Create ReplicaSet
2. Create ReplicaSet
3. Notify create ReplicaSet event
4. Notify create ReplicaSet event
5. Create Pod
6. Create Pod
7. Notify create Pod event
8. Notify create Pod event
9. Modify Pod
10. Modify Pod
11. Notify modify Pod event
12. List-Watch 1
13. List-Watch 2
14. List-Watch 3
15. kubelet
16. kubectl
17. kubectl
18. kube-apiserver
19. kube Controller
20. Manager
21. kube Scheduler
22. etcd
Recap: Scheduler Impl.

- Two-step process
  - scheduler predicates, e.g.,
    - NoDiskConflict, PodFitsResources, PodSelectorMatches, PodFitsHost, CheckNodeLabelPresence, CheckServiceAffinity, PodFitsPorts
  - Ranking/scoring of qualified nodes, e.g.,
    - LeastRequestedPriority
      score=\text{int}( \frac{(\text{nodeCpuCapacity} - \text{totalMilliCPU}) \times 10}{\text{nodeCpuCapacity}} + \frac{(\text{nodeMemoryCapacity} - \text{totalMemory}) \times 10}{\text{nodeCpuMemory}}) / 2)
    - BalancedResourceAllocation
      score= \text{int}(10 - \text{math.abs}(\frac{\text{totalMilliCPU}}{\text{nodeCpuCapacity}} - \frac{\text{totalMemory}}{\text{nodeMemoryCapacity}}) \times 10)

- A good comparison: HTCondor
  - https://research.cs.wisc.edu/htcondor/publications.html#match
Recap: Container/Node Implementation

- Container is just a higher abstraction built on top of Linux namespace for isolation
- Use both system calls or command lines to manage namespaces
- Network namespace contains devices (ip link list), routing tables (netstat -rn), firewall tables (iptables), ...
- Basic networking with network namespace
  - the veth pairs
  - the Linux bridge
Create a linux bridge
- `ip link add name br0 type bridge // or brctl addbr br0`

Add veth0 to the bridge
- `ip link set dev veth0 master br0 // or brctl addif br0 veth0`

Display devices on br0
- `brctl show`
Exercise: Protocol Stack, Bridge, Devices Connection

- **Exercise:** protocol workflow if we type
  - `%ping 10.1.1.1` in root name space

- **Verify**
  - `tcpdump -i veth0`
  - `tcpdump -i veth1`
  - `tcpdump -i br0`
Deeper Dive: Protocol Stack and Bridge Multiplexing

Problem
- According to routing table, protocol stack uses (sends and receives) veth0
- veth0 forwards ARP reply to br0, not protocol stack
Deeper Dive: Protocol Stack and Bridge Multiplexing

- **Problem**
  - According to routing table, protocol stack uses (sends and receives) veth0
  - veth0 forwards ARP reply to br0, not protocol stack

- **Fixing routing problem**
  - `ip addr del 10.1.1.2/24 dev veth0`
  - `ip addr add 10.1.1.2/24 dev br0`

Exercise: if we add a new network namespace netns3 with IP 10.1.1.3 and want it to communicate with netns1, how do we set up?
Exercise: if we add a new network namespace netns4 with IP 10.1.1.4 but want it to communicate only with netns1, not netns3.
Docker: Demo

- Start a container using docker
  - ip link show // or ifconfig see current devices
  - docker run -d nginx // ifconfig to see devices again
  - docker inspect <container id> // find PID; see IP address, gateway
  - ls -l /proc/<pid>/ns // see the namespaces

- Docker networking
  - brctl show // Show all bridges
  - ifconfig <bridge>

- See docker container network namespace
  - ip netns attach <name> <pid>
  - ip netns exec <name> ip link show

```bash
# docker inspect 6f5 | grep Pid
"Pid": 15818,
# ls /proc/15818/ns/
total 0
lrwxrwxrwx 1 root root 0 Jan 10 16:16 ipc -> ipc:[4026535715]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 mnt -> mnt:[4026535789]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 net -> net:[4026535718]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 pid -> pid:[4026535791]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 user -> user:[4026531887]
lrwxrwxrwx 1 root root 0 Jan 10 16:16 uts -> uts:[4026535790]
```
### Additional Networking Devices: Tunnels

- **Tunnel**: Allows user space applications to process packets
- **Example**: Design a VPN server that encapsulates packets to a given destination set (e.g., 192.168.1.0/24) using an external encapsulation
#include <net/if.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <linux/if_tun.h>
#include <stdlib.h>
#include<stdio.h>

int tun_alloc(int flags)
{
    struct ifreq ifr;
    int fd, err;
    char *clonedev = "/dev/net/tun";

    if ((fd = open(clonedev, O_RDWR)) < 0) {
        return fd;
    }

    memset(&ifr, 0, sizeof(ifr));
    ifr.ifr_flags = flags;

    if ((err = ioctl(fd, TUNSETIFF, (void *) &ifr)) < 0) {
        close(fd);
        return err;
    }

    printf("Open tun/tap device: %s for reading...\n", ifr.ifr_name);
    return fd;
}

int main()
{
    int tun_fd, nread;
    char buffer[1500];

    /* Flags: IFF_TUN   - TUN device (no Ethernet headers)
     * IFF_TAP   - TAP device
     * IFF_NO_PI - Do not provide packet information
     */
    tun_fd = tun_alloc(IFF_TUN | IFF_NO_PI);

    if (tun_fd < 0) {
        perror("Allocating interface");
        exit(1);
    }

    while (1) {
        nread = read(tun_fd, buffer, sizeof(buffer));
        if (nread < 0) {
            perror("Reading from interface");
            close(tun_fd);
            exit(1);
        }
        printf("Read %d bytes from tun/tap device\n", nread);
    }
    return 0;
}
Tunnel Example: Execution and Configuration

- gcc tun.c -o my_tun
- ./my_tun // assume created tunnel 0 tun0; ip addr
- ip addr add 192.168.1.2/24 dev tun0
- ip link set tun0 up

Exercise: packet flow to ping 192.168.1.3
Kubernetes Pod Networking Requirements

- Pods are allocated IP addresses from a given range
- IP-Per-Pod: Each Pod has a single IP from the range
- Pods run in a direct-connect, flat address space
  - Containers in the same Pod share IP, same Linux network stack, and can connect using localhost
Setting

Discussion: Key Issues/Approaches?
Example: Flannel Workflow (ipam)

- Use etcd to coordinate
  - Share a large network address space and post to etcd
    ```
    # etcdctl set /coreos.com/network/config '{ "Network": "172.17.0.0/16" }'
    ```
  - Use etcd to coordinate address partition
    ```
    # etcdctl ls /coreos.com/network/subnets
    /coreos.com/network/subnets/172.17.18.0-24
    /coreos.com/network/subnets/172.17.19.0-24
    /coreos.com/network/subnets/172.17.20.0-24
    ```

- Modify node docker start parameter to allocate pod IP in the range, e.g.,
  - --bip=172.17.18.0/24
Example: Flannel Workflow (Routing)

- Flannel agent/node acquires underlay routing from etcd
- Build routing table using the etcd data

```bash
# etcdctl ls /coreos.com/network/subnets
/coreos.com/network/subnets/172.17.18.0-24
/coreos.com/network/subnets/172.17.19.0-24
/coreos.com/network/subnets/172.17.20.0-24

# etcdctl get /coreos.com/network/subnets/172.17.18.0-24
{"PublicIP":"192.168.14.97"}
# etcdctl get /coreos.com/network/subnets/172.17.19.0-24
{"PublicIP":"192.168.14.98"}
# etcdctl get /coreos.com/network/subnets/172.17.20.0-24
{"PublicIP":"192.168.14.100"}
```
Example: Weave using Gossip to Sync Data
Example: Calico w/ BGP vRouter
MicroServices (Kubernetes)
Core Ideas/Journey

- Virtualization of a network service, consisting of multiple components, providing scaling, reliability services [Part 1 and Part 2 revisited]
  - Kubernetes basic concepts: containers, pod, deployment/replica set, service
- Design reliable, modular control architecture for Kubernetes [election revisited]
  - Reliable data store as the core of control architecture; api server as a gateway for policy check
- Systematic framework for scheduler mapping pod to nodes
  - Predicates/ranking scheduler framework
- Connecting isolated container network namespace
  - Introduce veth (aka pipe)
- Scaling veth connectivity
  - Introduce linux bridge, and configure the bridge into the Node routing system
- Coordinating ip address management and routing system across nodes
  - Use etcd to coordinate ipam
  - Monitor etcd to obtain the mapping from overlay IP and underlay IP
- Manage overlay routing/underlay routing conflict, scaling
  - Kubernetes routing systems using BGP or a custom routing (synchronization) protocol [Part 4 revisited]
- Extend overlay routing from L3 to L4-L7
  - Service mesh, sidecar
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
- Application-network coordination and integration
Overview

So far we focus more on the layered architecture, separating network and applications (decoupled architecture)

Exercise:
- Do you know any tools/approaches for app to get network info?
- Do you know any tools/approaches for network to know app info?
- Do you remember/know any settings with explicit network (networking system) and application (networked application) coordination?
Network, Application Coordination Use Case: In-Network Scheduling/Processing

- Network routing, scheduling satisfying application QoE (or in general in-network processing supporting apps)

- Some interesting real-time app protocols
  - Apple’s HTTP Live Stream (HLS)
    - RFC 8216
  - Cloud gaming
    - Nvidia StreamGame Protocol; an open source imple. see https://github.com/moonlight-stream/moonlight-common
A small number of large applications contribute to the majority of traffic in many networks.
Network, Application Coordination/Integration: Scopes

Network-aware Networked System (Network Application)

Application-aware Networking System
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
- Application-network integration
  - Overview
  - Network-aware applications
    - ALTO (application-layer traffic optimization)
      - Foundation and intuition
Basic Challenge of ALTO Design

- Applications and networks can be designed with different objectives
  - Application: optimizes application’s utility
  - Network: optimizes network’s utility, enforces fairness, ...

- Apps and networks have privacy/scalability concerns to share complete info
A Simple, Illustration Example

- **Application objective:** optimize total throughput
  - Using a fluid model*, we can derive that: optimizing throughput $\Rightarrow$ maximizing up/down link capacity usage

$$\max \sum_{i} \sum_{j \neq i} t_{ij}$$

$$s.t. \forall i, \sum_{j \neq i} t_{ij} \leq u_i ,$$

$$\forall i, \sum_{j \neq i} t_{ji} \leq d_i ,$$

$$\forall i \neq j, t_{ij} \geq 0$$

- **Network objective:** load balancing minimizing maximum link utilization (MLU)
  - $b_e$: background traffic volume on link $e$
  - $c_e$: capacity of link $e$
  - $I_e(i,j) = 1$ if link $e$ is on the route from $i$ to $j$
  - $t^k$: a traffic matrix $\{t^k_{ij}\}$ for each pair of nodes $(i,j)$ for app $k$

$$\min_{e \in E} \max \sum_{k} \sum_{i \neq j} t^k_{ij} I_e(i,j) / c_e$$

*fluid model*
Global Traffic Engineering System Formulation

- Combine the objectives of network and applications

\[
\min_{e \in E} \max \left( b_e + \sum_k \sum_{i \neq j} t_{ij}^k I_e(i, j) \right) / c_e
\]

s.t., for any \( k \),

\[
\begin{align*}
\max \sum_i \sum_{j \neq i} t_{ij}^k \\
\text{s.t.} \forall i, \sum_{j \neq i} t_{ij}^k &\leq u_i^k, \\
\forall i, \sum_{j \neq i} t_{ji}^k &\leq d_i^k, \\
\forall i \neq j, t_{ij}^k &\geq 0
\end{align*}
\]
Why Hard: Constraints Couple Net/App

\[
\min_{\forall k: t^k \in T^k} \max_{e \in E} \left( b_e + \sum_{k} \sum_{i \neq j} t_{ij}^k I_e(i,j) \right) / c_e
\]

\[
\min_{\forall k: t^k \in T^k} \quad \text{s.t.} \quad \forall e \left( b_e + \sum_{k} \sum_{i \neq j} t_{ij}^k I_e(i,j) \leq c_e \right)
\]

Constraints couple network/applications together!
A Two-Slide Summary of Constrained Convex Optimization (Primal-Dual) Theory

\[
\begin{align*}
\max & \quad f(x) \\
\text{subject to} & \quad g(x) \leq 0 \\
\text{over} & \quad x \in S
\end{align*}
\]

- Map each \( x \) in \( S \), to \([g(x), f(x)]\)
- Top contour of map is concave
- Easy to read solution from contour (how?)
- For each slope \( q \geq 0 \), computes \( f(x) - qg(x) \) of all mapped \([f(x), g(x)]\)

\[
D(q) = \max_{x \in S} \left( f(x) - qg(x) \right)
\]
A Two-Slide Summary of Constrained Convex Optimization (Primal-Dual) Theory

\[
\begin{align*}
\max & \quad f(x) \\
\text{subject to} & \quad g(x) \leq 0 \\
\text{over} & \quad x \in S
\end{align*}
\]

\(f(x)\) concave
\(g(x)\) linear
\(S\) is a convex set

\[D(q) = \max_{x \in S} (f(x) - qg(x))\]

- \(D(q)\) is called the dual;
- \(q \geq 0\) are called prices in economics
- \(D(q)\) provides an upper bound on obj.
- According to optimization theory: when \(D(q)\) achieves minimum over all \(q \geq 0\), then the optimization objective is achieved.
Exercise
ALTO Solution Architecture:
Decouple Network and Application

\[ \min_{\forall k : t^k \in T^k} \max_{e \in E} \left( b_e + \sum_k \sum_{i \neq j} t_{ij}^k I_e(i, j) \right) / c_e \]

Introduce \( p_e \) to decouple the constraints

\[ \min_{\forall k : t^k \in T^k} \alpha \]

s.t. \( \forall e : b_e + \sum_k \sum_{i \neq j} t_{ij}^k I_e(i, j) \leq \alpha c_e \)

- Look at dual function, with dual variable \( p_e \) (\( \geq 0 \)) for the inequality of each link \( e \), and apply dual finite condition

\[ \sum_e p_e c_e = 1 \]

\[ D(\{ p_e \}) = \min_{\forall k : t^k \in T^k} \sum_e p_e (b_e + \sum_k t_e^k) = \sum_e p_e b_e + \sum_k \min_{i \neq j} \sum p_{ij} t_{ij}^k \]

\( p_{ij} \) is the sum of \( p_e \) along the path from node \( i \) to node \( j \)
Solution Architecture:

- The interface between applications and network is the dual variables \( p_{ij} \)

- There are more straightforward settings to understand \( p_{ij} \), e.g., financial cost of the path from \( i \) to \( j \)

- High-level goal: provide *provider-distance* between network points

\[ p_{ij} = \sum_{e \text{ on route } i \rightarrow j} p_e \]
Outline

- Admin and recap
- Microservice oriented architecture - Kubernetes
- Application-network integration
  - Overview
  - Network-aware applications
    - ALTO (application-layer traffic optimization)
      - Foundation and intuition
      - Basic protocol
ALTO Basic Protocol

- A network consists of nodes and paths
- A path has path properties (cost metrics) conveyed in endpoint costs or cost maps

POST /endpointcost/lookup HTTP/1.1
Host: alto.example.com
Content-Length: 248
Content-Type: application/alto-endpointcostparams+json
Accept: application/alto-endpointcost+json,application/alto-error+json

```
{
  "cost-type": {"cost-mode": "ordinal", "cost-metric": "routingcost"},
  "endpoints": {
    "srcs": ["ipv4:192.0.2.2"],
    "dsts": [
      "ipv4:192.0.2.89",
      "ipv4:198.51.100.34",
      "ipv4:203.0.113.45"
    ]
  }
}
```

HTTP/1.1 200 OK
Content-Length: 274
Content-Type: application/alto-endpointcost+json

```
{
  "meta": {
    "cost-type": {"cost-mode": "ordinal", "cost-metric": "routingcost"}
  },
  "endpoint-cost-map": {
    "ipv4:192.0.2.2": {
      "ipv4:192.0.2.89": 1,
      "ipv4:198.51.100.34": 2,
      "ipv4:203.0.113.45": 3
    }
  }
}
```

GET /costmap/num/routingcost HTTP/1.1
Host: alto.example.com
Accept: application/alto-costmap+json,application/alto-error+json

HTTP/1.1 200 OK
Content-Length: 435
Content-Type: application/alto-costmap+json

```
{
  "cost-map": {
    "PID1": {"PID2": 1, "PID3": 5, "PID3": 10 },
    "PID2": {"PID1": 5, "PID2": 1, "PID3": 15 },
    "PID3": {"PID1": 20, "PID2": 15 }
  }
}
```
A network consists of nodes and paths

A node can be an
- endpoint
- aggregation of endpoints (PID)
- abstract network element

Each node is called an entity
- An entity has properties that can be inherited
- An entity can have capabilities

A path has path properties:
- cost metrics, calendars
- vector of abstract network elements

A set of paths can form a co-flow, with:
- shared abstract network elements cross the co-flows
Offline Example: Mathematical Programming as Resource Constraint Abstraction of CoFlow

- Two circuits:
  - Flow 1 (f1): S1 -> D1
  - Flow 2 (f2): S2 -> D2

- Share common links: l3 and l4
  - It is not possible to reserve 100Gbps (for both circuits)
**Offline Example: Mathematical Programming as Resource Constraint Abstraction of CoFlow**

- **GOAL:** Use mathematical programming constraints to provide a compact representation of the available bandwidth of flows through a network.

\[ x_1 \leq 100 \quad \forall u \in \{l_1, l_2, l_5, l_6\}, \]

\[ x_2 \leq 100 \quad \forall u \in \{l_7, l_8, l_{11}, l_{12}\}, \]

\[ x_1 + x_2 \leq 100 \quad \forall u \in \{l_3, l_4\}, \]

- \( x_1 \) (f1.awb): flow 1’s available bandwidth
- \( x_2 \) (f2.awb): flow 2’s available bandwidth
Offline Example: Mathematical Programming as Resource Constraint Abstraction of CoFlow

- **GOAL**: Use mathematical programming constraints to provide a compact representation of the available bandwidth of flows through a network.

Geometrically, resource abstraction represents the resource feasible region of the network for providing resources to a set of flows. Redundant inequalities should be removed.

- $x_1 \leq 100 \quad \forall l_u \in \{l_1, l_2, l_5, l_6\}$,
- $x_2 \leq 100 \quad \forall l_u \in \{l_7, l_8, l_{11}, l_{12}\}$,
- $x_1 + x_2 \leq 100 \quad \forall l_u \in \{l_3, l_4\}$,

- $x_1$ (f1.awb): flow 1’s available bandwidth
- $x_2$ (f2.awb): flow 2’s available bandwidth
Offline Example: a Single Network to Multiple Domains

- Each member network will abstract the bandwidth sharing for each circuit into a set of linear inequalities.
Problem: Although each domain may already conduct redundancy optimization, there can be cross-domain redundancy.

The constraint on flow 2 and flow 3 at M3 (<=10) can eliminate that at M2 (<=40).
**Offline Example: Multi-Domain Redundancy Optimization**

- **Problem**: Although each domain may already conduct redundancy optimization, there can be cross-domain redundancy.

The constraint on flow 2 and flow 3 at $M_3$ ($\leq 10$) can eliminate that at $M_2$ ($\leq 40$), for example, using a **Secure Multiparty Computation approach** for eliminating cross-domain redundancy.
Aggregate the abstraction in multiple networks into a **unified, single, virtual** representation:

\[ x_1 \leq 10, \quad x_2 + x_3 \leq 10. \]
An Example Deployment at Deutsche Telekom

Steering Hyper-Giants’ Traffic at Scale

CoNEXT ’19 December 9–12, 2019 Orlando, FL, USA

Figure 13: Timeline: From a research idea to a fully operational CDN-ISP collaboration. Top: Project management and infrastructure roll-out events. Bottom: FD’s development milestones and main overhauls (++).
Real-Deployment Complexity

Steering Hyper-Giants’ Traffic at Scale

Figure 9: Flow Director: High-level system architecture.

Figure 10: Flow Director processing pipeline.

Outline

- Admin and recap
- Microservice oriented architecture – Kubernetes
- Application-network coordination and integration
  - Overview
  - Network-aware applications
  - Application-aware networking
App-Aware Networking

- First step is traffic differentiation, but is controversial
  - Diffserv allows application to set QoS—lacks granularity, security, and enforcement
  - DPI into application layer data—obsoleted by increased use of TLS, has myriad of issues when stateful
  - SPUD/PLUS proposals—requires UDP, flow state, DPI, and global definition of service characteristics

- One recent trend: User-centric, application-agnostic, privacy-aware, e.g.,
  - Zero-Rating: Category-based, inclusive, money-free agreements
Example: Network Tokens Based

1. Application asks user permission to access premium network quality service

2. Client agent fetches premium quality token with user’s credentials

3. Application attaches token to flows of interest

4. Network detects tokens and provides service

https://networktokens.org
Offline Read: OAuth2

- Structure of network tokens motivated by OAuth2
- OAuth2 framework:
  - A few good tutorials, e.g.,
    - https://www.oauth.com/playground/
Backup Slides
**Application-aware ID Option**

- **Carrying SLA Level, Application ID, User ID, Flow ID, Service Requirements**
- The length of the APP-aware ID is recommended to be 128bits

**Structure I:** Any combination of SLA level (e.g. Gold, Silver, Bronze), APP ID, and/or user ID, and/or Flow ID

<table>
<thead>
<tr>
<th>SLA Level</th>
<th>APP ID</th>
<th>User ID</th>
<th>Flow ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Structure II:** Any combination of SLA level (e.g. Gold, Silver, Bronze), APP ID, and/or user ID, and/or Flow ID, plus the arguments which indicating the service requirements of the identified application

<table>
<thead>
<tr>
<th>SLA Level</th>
<th>APP ID</th>
<th>User ID</th>
<th>Flow ID</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Structure III:** An SRv6 SID, with its arguments as the information specified in Structure II

<table>
<thead>
<tr>
<th>Locator Address</th>
<th>Function ID</th>
<th>Arguments</th>
</tr>
</thead>
</table>
Service-Para Option

- Carrying the service requirement parameters
Network Tokens

- Policy dictated by token distribution and crypto functions
- App-specific token
  ```json
  {"alg":"ES256", "kid":"N6fr1MDrEuu1eXRkFbcpX4WY62SKN7TKrhYa9PfJEd8"}.{"sub":"Skype", "iat":1588116732, "exp":1588117732, "bip":"140.54.35.194"}
  ```
- User-driven, application agnostic, privacy aware token
  ```json
  {'alg':'dir', 'app_id':'low-latency'}.{'sub':'+14151234567', 'nti':5871234,'exp':1588203132}
  ```