Network Tokens
draft-yiakoumis-network-tokens-01

Abstract

   Network tokens is a method for endpoints to explicitly and securely
   coordinate with networks about how their traffic is treated. They
   are inserted by endpoints in existing protocols, interpreted by
   trusted networks, and may be signed or encrypted to meet security and
   privacy requirements. Network tokens provide a means for network
   operators to expose datapath services (such as a zero-rating service,
   a user-driven QoS service, or a firewall whitelist), and for end
   users and application providers to access such services. Network
   tokens are inspired and derived by existing security tokens (like JWT
   and CWT), and borrow several of their core ideas along with security
   and privacy properties.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on December 17, 2020.

Copyright Notice

Table of Contents

1. Introduction

   This specification motivates and describes network tokens, a method
   for endpoints to explicitly coordinate with networks about how their
   traffic is treated. They provide a means for networks to expose
   datapath services, and for end users to access such services by
   appropriately tagging their traffic. Network tokens are intended for
   scenarios where there is explicit coordination and trust between
   endpoints and the network, like a zero-rating service, a user-driven
   QoS service, or a firewall whitelist.

1.1. Network Token Overview

   A network token is a small piece of data that end users attach to
   their packets. As packets flow through the network, intermediate
   nodes MAY detect tokens, interpret them, and apply the desired
   service to the packets that carry them (and possibly to all other
   packets from the same flow).

   Tokens MAY be digitally signed, integrity protected and/or encrypted
   to account for privacy and security, and can be provisioned to
   prevent replay and spoofing attacks.

   Tokens carry simple claims that can drive network policy.

   For example, a token might just state the name of the application
   that a packet originates from, which can then be used by firewalls
   and/or zero-rating whitelists. Such a token (an "application
   token"), would be signed with the private key of the application
   provider, and could be interpreted and verified by any one with the
   application provider's public key. It may also be bound to the IP

address of the application provider's server that generates the
traffic so it cannot be used in a different context.

Similarly, a token might state the request of an end-user to access a
network service, such as a low-latency and reliable QoS SLA, in a
user-centric, application agnostic, and privacy preserving means.
Such a token (a "user-centric token"), will hold a unique user
identifier (like an MSISDN), the keyword "lowlatency", an expiration
date, a nonce for revocability, and will be encrypted with a network
operator's secret key. The contents of the token will be opaque and
uninterpretable by everyone other than the operator (including the
user).

Tokens are policy-agnostic, i.e., they just provide a unified
mechanism to communicate and interpret certain claims in the
datapath. Network services built using tokens can dictate the
desired policy through token (or token metadata) distribution, and
the cryptographic functions applied to them.

Network tokens do not dictate a dedicated header or protocol to be
inserted. Instead, they are incorporated as options and extensions
into a variety of existing protocols. For example, they can be
carried as TLS Client and Server Hello extensions during a TLS
handshake, as IPv6 Hop by Hop Options, or as attributes during a
STUN-enabled flow setup. Network tokens are largely opaque to the
protocols that carry them.

Network tokens are inspired and derived by existing security tokens,
like JSON Web Token (JWT) [RFC7519] and CBOR Web Token (CWT)
[RFC8392], and borrow a lot of their properties in terms of security
and privacy. In fact, network tokens MAY be represented as JWT and
CWT objects, and respectively use JOSE and COSE technologies for
signing and encryption.

2. Motivation

Network traffic differentiation is widely deployed in enterprise,
residential, and cellular networks. Typical use cases are firewall
whitelists, zero-rating programs, and custom QoS SLAs where certain
traffic is granted special treatment.

A common concern for all such services is how to identify traffic of
interest in order to map it to (and enforce) the desired network
policy. The mechanism that does this essentially becomes the
interface between network operators, application providers and end-
users, and has direct implications to network management and
security, user privacy, business practices, and compliance with net
neutrality regulation.

Identifying traffic of interest is not straight forward, as it often
depends on context not present in the related packets themselves.
For example, the decision to allow a packet through a firewall or
zero-rating whitelist is based on the application that generated the
packet, while routing a flow through a low-latency path depends on
the desire of a user to prioritize this flow and potentially pay for
it. Requirements around accountability and verification, fraud
prevention, privacy preservation, and compliance with net neutrality
regulation just make the task harder.

This section discusses usecases, existing mechanisms that map traffic
to network differentiation services, and then details some of the
challenges through the perspective of different stakeholders, and how
network tokens can help to address them.

2.1. Use cases Overview

2.1.1. Zero Rating

Zero rating (and similarly sponsored data and other application-
specific data plans) is the practice of differentiating charging of
internet access based on the application that generated data. For
example, a mobile operator might allow its users to stream music
without paying for data as part of a promotional offer, or purchase a
discounted data plan that can be used only for certain applications.
It is deployed in several cellular networks along with data usage
caps.

Zero-rating services require close collaboration between application
providers and mobile operators. The typical workflow involves the
application provider sharing an application signature with the
operator (i.e., a list of domains and IP addresses used to serve
traffic), operators configuring their networks to detect traffic with
these characteristics, and then map it to predefined charging groups.
The process repeats whenever there are updates in application
signatures.

A critical metric for zero-rating integrations is accuracy, i.e.,
what percentage of the traffic from a specific application is
detected (and properly charged) by the network. Undetected traffic
that should otherwise be zero-rated, leads to unexpected charges or
packet drops for users. Zero-rating traffic that shouldn't be zero-
rated, leads to loss of revenue for operators. As such, network
operators evaluate detection accuracy for each application against a
threshold, and appropriately decide whether to add an application or
not. There are typically four sources of inaccuracy:

o Third-party traffic: Most modern applications include third-party
   traffic which cannot be counted as part of the application
   signature (e.g., analytics, ads, social network plugins, etc).

o Out-of-sync application signature: Application signatures often
   change (when servers and/or domains are added or deleted). When
   these changes are not incorporated in the network configuration,
   inaccuracies occur.

o Fraudulent behavior: Malicious users may attempt to masquerade
   their traffic to appear as eligible zero-rating traffic when it is
   not. For example, one can setup a rogue proxy server, spoof a
   zero-rated domain through the SNI field, and route all traffic
   through this proxy. This has lead some operators to only perform
   zero-rating based on IP addresses.

o Unsupported detection methods: this is common when application
   providers use peer-to-peer connectivity, or when their traffic
   comes from CDN servers with shared IP addresses and the operator
   does not support domain-based signatures.

Another important metric for zero-rating is to keep onboarding and
operational overhead low, as operators typically zero-rate multiple
applications. For example, a common practice, driven by regulatory
and/or commercial requirements is to apply the same treatment to a
group of applications from specific categories (e.g., music, video,
social networks, gaming). From a regulatory perspective, the goal is
to provide a level playing field for competing application providers
according to net neutrality principles. From a commercial
perspective, grouping multiple applications together can improve
perceived value for users.

Zero-rating is typically either application-specific, or specific to
a category of applications, and implies a trust relationship between
an application provider and the network.

2.1.2. Firewall Whitelist

A firewall whitelist shares many characteristics with zero-rating as
far as it concerns this document. It is typically application-
specific, and implies a trust relationship between an application
provider and the network operator.

A critical metric for firewall whitelists is performance, as
firewalls may become bottlenecks in otherwise well-provisioned
networks. Another metric is accuracy, as letting insecure traffic
through a firewall can become a security risk.

Firewall whitelists are typically used in enterprise networks to
allow traffic from certain applications through the network, or
bypass an expensive classification path.

[2.1.3](#). **QoS**

QoS services have been historically deployed in a variety of
networks, and emerging use cases like SD-WAN and 5G slicing renewed
interest to how they are implemented.

This document is related only to QoS policies where a subset of the
traffic to/from an endpoint receives special treatment. For example,
a scenario where a user's traffic is throttled at 10Mbps during peak
evening hours is out-of-scope for this document. A scenario where a
user accesses a 10Mbps connection for a flat fee, an hourly 100Mbps
connection for an hourly fee, and she can dynamically decide which
packet to send to which QoS SLA is in scope for this document.

QoS policies might be either application specific, or user-centric.
Application-specific applications are similar to zero-rating and
firewall whitelists services, and imply trust between an application
provider and the network operator.

In contrast, user-driven QoS policies imply trust between the user
and the network operator, while the role of the application
developer, if any, is to facilitate their interaction. In other
words, the network operator is not concerned with whether the traffic
from an application is eligible for certain treatment. It just needs
to verify that the end-user requires a flow to receive a special
treatment. A user might be an individual, an enterprise, or an
organization that wants to use a third-party application over a QoS
SLA provided by a network operator.

For example, a user-driven approach for mobile networks is
recommended by regulators in multiple countries as a way to offer QoS
differentiation in a way compliant with net neutrality regulation.
Additionally, services might have to be application-agnostic (i.e.,
the user should be able to use it for any application they want) and
privacy preserving (i.e., the network operator doesn't need to know
the application associated with this traffic to offer the service).

Similar requirements arise in enterprise networks, where companies
purchase multiple SLAs from their connectivity provider and want to
decide how they differentiate traffic from different applications
through respective paths.

QoS services are often charged based on usage, and therefore
accountability and verification are important aspects of it.

2.2. Existing mechanisms

The two primary methods that map traffic to network differentiation services today are DiffServ and Deep Packet Inspection.

2.2.1. DiffServ

DiffServ [RFC2475] uses DSCP bits in the IP header to map traffic into specific classes and drive per-hop-behavior in the network for traffic differentiation. DiffServ operates within a single administrative domain, and assumes that traffic is properly marked and classified in the network boundaries. In many practical deployments, the endpoints under consideration are not under the network's administrative domain and this causes issues.

Intermediate nodes in the path may alter or reset the DSCP bits (e.g., to use them for their own purposes), therefore nullyfying any endpoint marking. Second, and maybe most important, DiffServ has no authentication and revocation primitives: any application can set the DSCP bits and request service without the user's consent. Any developer can ask for special network treatment even if it conflicts with a user's desire, or--even worse--if it results in network charges for users, and users or operators do not have the means to easily revoke such access.

Network tokens can solve such problems as they operate across network boundaries, and support revocation and authentication. In that sense, tokens are complementary to DiffServ, not a replacement. They can be used to communicate a claim in a secure way, across network boundaries. Once a token is interpreted, DiffServ can be used within an administrative domain to drive enforcement on a per-hop basis.

2.2.2. Deep Packet Inspection

Deep Packet Inspection uses predefined application signatures to detect traffic of interest, and then enforces the desired policy. Application signatures are a combination of IP addresses, domain names, SSL certificates, and other fields, that infer the application that generated traffic by implicitly observing traffic between endpoints. They are widely deployed by network operators today as an enabler for traffic differentiation services.

The advantage of DPI is that it requires no changes from endpoints. But as new usecases come up, applications grow in complexity, and privacy or net neutrality requirements strengthen, its shortcomings become more dominant.

o Modern applications use a variety of protocols and architectures,
   integrate third-party services, establish connectivity through
   thousands of nodes, and constantly change. Maintaining
   application signatures is manual, expensive, and often inaccurate.
   They require frequent updates, involve manual interactions between
   parties, and cannot cover third-party traffic scenarios.

o Application signatures are vulnerable to fraud, as bad actors can
   spoof certain fields (like Server Name Identification) and pretend
   they are eligible for preferential treatment.

o Deep Packet Inspection policies are strictly linked with an
   application. This raises privacy concerns, as networks need to
   know the application that traffic originates from in order to
   enforce a policy.

o New privacy-enabling protocols (like DNS over HTTPS and Encrypted
   SNI) encrypt the last bits of cleartext information sent over the
   internet, further limiting DPI-based application detection.

2.3. Requirements and Challenges

2.3.1. Integration overhead

Network services often require coordination between a network
operator and an application provider. For example, firewall
whitelists and zero-rating programs require the coordination between
networks and the applications to be whitelisted (or zero-rated). It
is typical for such programs to include a large and growing number of
applications, and the integration process and mapping interface
matters a lot, as it dictates the required effort for both network
operators and application providers.

Network operators want to streamline the onboarding process for new
applications in their programs, and also minimize the overhead to
keep these integrations functional and accurate. They also want to
make it easy for third parties to integrate and use their services.
Network tokens enable operators to onboard new applications just by
granting them a new token, without additional per-application
overhead to create, evaluate, and maintain app signatures. A token-
based approach also decouples integration from the architecture of an
application provider, meaning that there is no need for updates and
maintenance every time an application partner changes its
infrastructure.

Decoupling integration from an application's infrastructure is
important for the application provider's side as well. They don't
have to pace deployment of new servers to give partnering networks

advance warning to update their DPI, and they can still leverage a
network's services even when they don't own a server or when the
traffic is originating from a third-party service. Another benefit
for application providers is control. Tokens give application
providers the capability to decide when and how to use related
services, so they can offer it only to a subset of their users, for a
limited period, or run A/B experiments without any infrastructure
overhead and without further coordination with the network.

Regulators MAY require that zero-rating programs are available to a
large number of application providers. For example many zero-rating
programs are required to onboard all applications in a category.
Network tokens offer a straight-forward and low overhead onboarding
process, and make it easier to keep implementations compliant.
Regulators may also have to monitor commercial practices for
compliance, and therefore auditability becomes important. With
network tokens, audits need to only check token distribution, which
can be as simple as a database with when an application asked for a
token, and when this was granted.

2.3.2. Detection Accuracy

Detection accuracy captures what percentage of traffic that falls
under a policy gets eventually detected and treated accordingly by
the network. It is one of the main criteria used during a zero-
rating integration, and where most problems appear. One reason is
that many applications involve traffic from third party servers that
cannot be properly accounted for (like ads, social media add-ons, or
traffic from public CDNs). Another source of detection inaccuracy
comes from application backend changes that are not properly
communicated to or acted upon by the network operator.

Many integrations do not happen just because of inaccuracy reasons,
causing issues for both application providers and operators. In
other cases, application providers might have to change the
functionality for such integrations to happen (e.g., disable ads)
which can have a significant impact on their business and/or user
experience.

Once an integration is established, failing to detect traffic may
lead to unintended charges for users and dissatisfaction. Operators
have to retroactively perform troubleshooting, deal with customer
support, and issue refunds.

Inaccuracies also pose a trade-off for regulators: if detection
inaccuracies are accepted, plan transparency issues arise (e.g.,
users might get charged for use of an application that is promoted as
free). In contrast, when inaccuracies are not allowed, some eligible

applications cannot participate without modifications to their
implementation, thus raising issues about the openness of such
programs or entry barriers to market.

Network tokens are decoupled from an application's backend and can
also be applied to third-party traffic, giving application developers
the capability to improve accuracy without affecting the
functionality of their apps.

2.3.3. Fraud Prevention

Malicious users may exploit limitations in traffic detection and get
special treatment from the network. This often happens in zero-
rating services, where malicious users setup a proxy server, connect
to it using the properties (e.g., SNI) of an otherwise zero-rated
application, and essentially zero-rate all their traffic.

When fraud happens it leads to lost revenue for network operators.
To prevent fraud, some operators require that application signatures
use only IP addresses, which are harder for bad actors to spoof.
This in turn makes integration harder. Many application providers
use CDNs that share IP addresses across multiple applications, and
they become ineligible to participate. Others that do have their own
IP addresses can still integrate, but at the cost of more frequent
updates whenever as new servers (and IP addresses) are added.

Network tokens deal with fraud through signed and/or encrypted tokens
that can be integrity protected and resistant to replay and spoofing
attacks.

2.3.4. Implementing user-centric control

One particular instance of traffic differentiation services (and
particular QoS services) is user-centric control. User-driven
control can better serve the needs of end-users, and might
additionally be driven by business and regulatory reasons.

Network tokens provide the means to put users in charge of QoS
treatment. User-specific tokens can be applied to traffic directly,
or given to applications after user consent, and be revoked at any
time, similar with OATH2 authentication workflows. They can be used
in an application agnostic manner, and don't require network
operators to limit their offerings to specific applications.

Moreover, tokens require explicit action, are verifiable, and prevent
abuse from third parties, which are necessary properties to build
network services that involve charging and accounting.

2.3.5. **Privacy**

   Privacy is often at odds with traffic differentiation services,
   especially when networks have to inspect a user's traffic to enforce
   a service, and/or this happens without a user's consent. Regulation
   around privacy, increased user awareness, as well as emerging
   protocols like Encrypted SNI and DNS over HTTPS require new ways to
   combine privacy awareness with traffic differentiation services.
   Tokens can address such concerns in a number of ways.

   Coupled with user-centric control, network operators can expose
   datapath services in an application agnostic manner and respect user
   privacy. They do not have to detect use of specific applications at
   all, all necessary information is included within a token, and users
   explicitly share relevant information with the network upon consent.

   Network tokens can also promote privacy for application-based network
   services. Today, if one network can detect an application (e.g.,
   through the use of SNI), every other network can as well. Using
   network tokens, the application provider (or user) can share this
   context only with trusted networks, keeping traffic largely opaque
   for other networks.

   As protocols like ESNI and DoH emerge, network tokens enable
   application providers to adopt them, and at the same time integrate
   with trusted networks. Respectively, they can enable network
   operators to expose traffic differentiation services, even when
   traffic is largely opaque to them.

3. **Representation**

   Network tokens can be represented in different formats. For example,
   a network operator might structure a token as a pre-defined byte
   sequence or a list of TLV-encoded fields. Alternatively, tokens can
   use existing JOSE and COSE technologies for representation, as they
   already provide a framework to securely communicate information
   between different entities. The actual representation and contents
   of a token should take into consideration the capabilities of the
   network to process them (i.e, what cryptographic functionality can
   the network support), the token's length in terms of header space,
   and requirements for integrity protection, privacy preservation, and
   attack scenarios.

   Examples in this document will use JWT to represent tokens, as they
   are well understood by the community and easily read by humans.
   Translation to a different representation format should be straight
   forward.

[4](#). Contents

The contents of a network token communicate the desired information between an endpoint and the network (e.g., the name of an application, or a user's request to access a low-latency services. Along with these claims, tokens carry necessary metadata to i) digitally sign and/or encrypt a token to meet privacy and security requirements, and ii) prevent unauthorized parties from replaying or using these tokens to inadvertently access network services (e.g., through the use of timestamps, expiration time, nonces).

[4.1](#). Network Token Common fields

Network tokens can have arbitrary fields (or claims). The fields defined below, while not mandatory, provide a starting point for a set of useful, interoperable fields. Network services using network tokens should define which specific fields they use and whether they are required or optional. Several of the fields listed below are already registered as part of JWT and CWT specifications, while others are specific for network tokens.

[4.1.1](#). 'iss' (Issuer) field

The "iss" (issuer) field identifies the principal that issued the token. For example, the issuer might be the name of the network operator that offers the service of interest.

[4.1.2](#). "sub" (Subject) field

The "sub" (subject) field identifies the principal that is the subject of the token. This could be a subscriber id, or the name of an application.

[4.1.3](#). "exp" (Expiration Time) field

The "exp" (expiration time) field identifies the expiration time on or after which the token MUST NOT be accepted for processing. The processing of the "exp" field requires that the current date/time MUST be before the expiration date/time listed in the "exp" claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. The "exp" field can be used to reduce the probability of replay attacks, restrict service access to a certain period, or to force users to refresh authentication credentials.

[4.1.4](). "iat" (Issued At) field

   The "iat" (issued at) field identifies the time at which the token
   was issued or generated. This field can be used to determine the age
   of the token, and can be used along or instead of the "exp" field.

[4.1.5](). "nti" field (Network Token ID) field

   The "nti" field provides a nonce-like value for the token. The
   identifier value MUST be assigned in a manner that ensures that there
   is a negligible probability that the same value will be accidentally
   assigned to a different data object; if the application uses multiple
   issuers, collisions MUST be prevented among values produced by
   different issuers as well. The "nti" field can be used to revoke a
   token, or prevent it from being replayed.

[4.1.6](). "bip" field (Bound IP) field

   The "bip" (Bound IP) field bounds the use of the token to a specific
   IP address. This can prevent third parties from reusing the token in
   a different context.

[5](). Network Token Format

   A token consists of the following fields (Figure X):

   o Reflect Type (4-bits): Indicates reflection properties for the
      token.

      * 0x0: Token is inserted by the origin of this flow. No
         reflection needed.

      * 0x1: Token is inserted by the origin of this flow. Reflect at
         receiver.

      * 0x2: Reflected token.

      * 0x3-0xf: Reserved

   o Token Descriptor ID (28-bits): An ID that helps the network decide
      whether and how to interpret tokens. Descriptor IDs are
      registered in the "Token Descriptor ID" registry (MSB = 0) or
      private (MSB equals 1). For private descriptor IDs, the definer
      of the value needs to take reasonable precautions to make sure
      they are in control of the part of the namespace they use (e.g.,
      by using a OUI prefix). A token descriptor might just indicate
      that the token payload is a JWT, or point to a structure that
      holds keys and other information to interpret a token.

o Token Payload: Depending on the application, the token payload
   might be a self-contained JWT or CWT (as plaintext, signed, or
   encrypted), a set of TLV-encoded values, or has its own custom
   format.

The length of the token is arbitrary, but must follow the limitations
imposed by the protocol it is encapsulated. For example, if the
token is carried as an IPv6 hop-by-hop option, the total length of
the token cannot exceed 2048 bytes.

```
         0         1         2         3
         01234567890123456789012456789 01
        +-------------------------------+
        | rfl |   token descriptor id   |
        +-------------------------------+
        |                               |
        |                               |
        |          token payload        |
        |                               |
        |                               |
        +-------------------------------+
```

6. Example Network Tokens

This section discusses example network tokens and how they can serve
specific use cases.

6.1. Application Token

An application token can be used to whitelist traffic from trusted
applications for a zero-rating or firewall whitelist scenario (as
discussed in Section {#usecases}.

The following example verifies that a network flow is coming from
"The Godfather App".

The token payload is encoded as a JWT, and encapsulated as a TLS
Extension attached in a Server Hello Message.

The Reflect Type is 0x00 (i.e., peers should not reflect it), with
the expectation that network flows can setup appropriate state for
the reverse flow as well. The Token Descriptor ID is 0x03, which
might represent a registered value for application tokens.

The JWT encodes the following object.

The header of the JWT has the following fields:

{"alg":"ES256", "kid":"N6fr1MDrEuu1eXRkFbcpX4WY62SKN7TKrhYf9PfJEd8"}

   The token is signed using the Elliptic Curve Digital Signature
   Algorithm, and the public key can be looked-up in a pre-defined
   database using the "kid" thumbprint.

   The JWT payload has the following fields:

{"sub":"The Godfather App", "iat":1588116732, "exp":
1588117732,"bip":"140.54.35.194"}

   The token is created by the application provider. It states that
   this flow originates from "The Godfather App", along with the time
   that it was created and when it expires. The token is signed with
   the app provider's public key, and any network can verify this
   through the attached signature. The token is also bound to a
   specific IP address, and therefore cannot be reused in a different
   context. For example, the application provider could configure all
   exit gateways to attach a token for all outgoing flows.

6.2. User-centric Token

   A user-centric token may be used to access a custom QoS SLA (e.g.,
   low latency) from a mobile operator. This is an application-agnostic
   and privacy-preserving token, i.e., users can use it for any traffic
   they want and the network operator doesn't need to know what
   application is associated with ths token.

   The token payload is encoded as a JWT, and can be inserted to STUN
   (as STUN attributes) or IPv6 packets (as IPv6 Hop-by-Hop extension
   header).

   The Reflect Type is 0x1, i.e., peers should reflect the token to
   setup state for the reverse flow. The Token Descriptor ID is 0x01,
   stating that the token payload is encoded as a JWT object.

   The header of the JWT has the following fields:

   {'alg':'dir','enc':"A256CBC-HS512", 'app_id':14098715987234}

   The token is generated by the operator, and signed with the AES-256
   algorith, using an operator's symmetric key. The app_id points to an
   operator-specific identifier associated with its own services.

   The payload of the token has the following fields, requesting for
   low-latency treatment, and bounding the start and end time of the
   token. It also has a unique identifier to allow revocation.

{'srv':'lowlatency', 'msisdn':'+4151111111111', 'nti': 5871234, 'iat':
1588116732, 'exp':1588203132}

   Each token is valid for 24 hours. As the encryption key is bound to
   a specific user, it cannot be used by another context. The token is
   opaque to everyone other than the operator (including the user). The
   Operating System (or an agent) in the user's device can request a
   token, and grant it to specific applications based on a user's
   request. Users can revoke access by telling an operator to blacklist
   the nti associated with this token.

   Besides accessing a low-latency service, this token serves two
   requirements: * it is application agnostic and can be used for any
   application a user wants * it preserves privacy. There is no
   indication about specific applications, and no identifier that can be
   linked to a user.

7. Network Tokens and Encapsulating protocols

   Network tokens are inserted in existing protocols by leveraging
   extension capabilities and do not require a dedicated header. The
   contents of the token are largely opaque to the protocol that carries
   them (i.e., they cannot read or verify a token).

   To support tokens, a protocol needs to allow its users to specify the
   token to be used (e.g., while opening a socket or configuring a
   connection), and appropriately reflect a token according to the value
   of a token's Reflect Type.

   While tokens are designed to be self-contained, the protocols that
   carry them inevitably affect its use. In particular:

   o the maximum size of tokens is dictated by the provision of the
      protocol extension that carries them.

   o Protocols that use a checksum over transmitted data (like TLS or
      optionally STUN) ensure that a token cannot be tampered or removed
      by intermediary nodes without the endpoints noticing it

   o Implementations should also consider whether the protocol
      guarantees that a token is contained in a single packet or might
      be carried over multiple packets.

   This section discusses the use of tokens in three widely used
   protocols, and section {#iana} describes recommended IANA changes for
   each protocol.

For the examples below we will use the following 227-byte long
network token, which encodes (in hex notation) the low-latency token
described in Section {#lowlatencytoken}.

10000001 # **Network token is represented as JWT, reflect at node**
65794a68624763694f694a6b615849694c434a6c626d4d694f694a424d6a553251304a444c556
8544e544579496e302e2e565f4278546d4e692d6b6337735a376b504e514851412e5a42764269
6d6d46705579555165e544579634b36244566a31378546d4e692d6b6337735a376b50476676c4
7597579666f734b43565234744576306a4d31735f4352484c50706a6335536d7703336487576
677541467c7979396f3a526e7976532484cd377033336487553559797170e514851412e5a42764
2696d6d4670557955165765676c47597579666f734b43565234744576306a4d31716535597971
702d5946796838566a396f72337a7351624e6d4e64757969747427658436854766c66633055704
4545597d429869d34e5486a63963305f6a5267424f39745a6e4535438566a396fe64757969747
27658766c6655704417e252636f267349565f36d585a6e4547568f7672637835f4352484c5417
72e32556e32636a6f6a5267424f396349565f7869536d586357745a6e4535495475684973456a
4f76726378472e3f9 # Network token payload (as JWE)

7.1. **Network Tokens as a TLS Extension**

Network tokens can be encoded as TLS extensions during the handshake
phase in ClientHello, ServerHello, and HelloRetryRequest messages, as
these are defined in [RFC8446]. As the handshake happens before
encryption is established between the two communicating endpoints,
the token will be sent unchanged and can be interpreted by any
trusted networks.

Network tokens used as TLS extension are flow-specific, i.e., the
network should apply the policy linked to this token to all packets
that belong to this flow.

The token comprises the extension data. Section {#iana} requests the
value 57 as a Network Token extension type. For TLS backward
compatibility, the first 16-bit after the extension type should
encode the length of the extension data. The following bytestream
defines the extension, including type and lengh data:

```
3900e3  # Network Token Extension with 227 bytes length
```
1000001 **# Network token is represented as JWT, reflect at node**
```
65794a68624763694f694a6b615849694c434a6c626d4d694f694a424d6a553251304a444c556
8544e544579496e302e2e565f4278546d4e692d6b6337735a376b504e514851412e5a42764269
6d6d46705579555165e544579634b36244566a31378546d4e692d6b6337735a376b50476676c4
7597579666f734b43565234744576306a4d31735f4352484c50706a6335536d37703336487576
677541467c7979396f3a526e7976532484cd37703336487553559797170e514851412e5a42764
2696d6d4670557955516576676c47597579666f734b43565234744576306a4d31716535597971
702d5946796838566a396f72337a7351624e6d4e6475796974727658436854766c66633055704
4545597d429869d34e5486a63963305f6a5267424f39745a6e4535438566a396fe64757969747
27658766c6655704417e252636f267349565f36d585a6e4547568f7672637835f4352484c5417
72e32556e32636a6f6a5267424f396349565f7869536d586357745a6e4535495475684973456a
4f76726378472e3f9 # Network token payload (as JWE)
```

   If the reflect type is set to 0, the peer takes no additional step.
   If reflect type is 0x1, the peer should attach a Network Token
   request at the ServerHello message, set the type to 0x2, and copy the
   rest of the token in it. The extension data for the ServerHello
   message that reflects the token mentioned above is listed below.

```
3900e3  # Network Token Extension with 227 bytes length
```
2000001 **# Network token is represented as JWT, reflected token**
```
65794a68624763694f694a6b615849694c434a6c626d4d694f694a424d6a553251304a444c556
8544e544579496e302e2e565f4278546d4e692d6b6337735a376b504e514851412e5a42764269
6d6d46705579555165e544579634b36244566a31378546d4e692d6b6337735a376b50476676c4
7597579666f734b43565234744576306a4d31735f4352484c50706a6335536d37703336487576
677541467c7979396o3a526e7976532484cd37703336487553559797170e514851412e5a42764
2696d6d4670557955516576676c47597579666f734b43565234744576306a4d31716535597971
702d5946796838566a396f72337a7351624e6d4e6475796974727658436854766c66633055704
4545597d429869d34e5486a63963305f6a5267424f39745a6e4535438566a396fe64757969747
27658766c6655704417e252636f267349565f36d585a6e4547568f7672637835f4352484c5417
72e32556e32636a6f6a5267424f396349565f7869536d586357745a6e4535495475684973456a
4f76726378472e3f9 # Network token payload (as JWE)
```

   TLS does not allow for extensions to be originated by the server if
   they are not defined in the ClientHello message. To account for
   cases where tokens need to be inserted by the server, the client
   might send an empty Network Token extension which allows the server
   to respond to with the appropriate token. Alternatively, the server
   can use a HelloRequestRetry message to ask the peer to re-send a
   ClientHello message with the NetworkToken extension included. The
   HelloRequestRetry should include the token. Depending on the reflect
   type the client peer might include an empty or populated Network
   Token extension on the subsequent ClientHello message.

   The use of Network Token between ClientHello, ServerHello, and
   HelloRequestRetry messages resembles the mechanics of the cookie TLS
   extension.

The TLS handshake is protected for message integrity, and as such
guarantees that network tokens cannot be dropped by intermediary
nodes.

7.2. Network Tokens as a STUN Attribute

Network tokens can be inserted as attributes in STUN Binding Request
and Binding Response messages, during the handshake that preceeds
WebRTC realtime communication flows [RFC5389].

Network tokens used as STUN attributes are flow-specific, i.e., the
network should apply the policy linked to this token to all packets
that belong to this flow.

The token comprises the attribute data. Section {#iana} requests the
value 0x8030 as a Network Token STUN attribute. The following
bytestream shows the attribute for the low-latency token described
earlier, including attribute type and length.

(TODO: We need to deal with padding here as STUN requires 32-bit
boundaries.)

```
803000e3  # Network Token Attribute with 227 bytes length
1000001 # Network token is represented as JWT, reflect at node
65794a68624763694f694a6b615849694c434a6c626d4d694f694a424d6a553251304a444c556
8544e544579496e302e2e565f4278546d4e692d6b6337735a376b504e514851412e5a42764269
6d6d46705579555165544579634b36244566a31378546d4e692d6b6337735a376b50476676c4
7597579666f734b43565234744576306a4d31735f4352484c50706a6335536d37703336487576
677541467c7979396f3a526e7976532484cd3770333648755355979717 0e514851412e5a42764
2696d6d4670557957555165576670c47597579666f734b43565234744576306a4d31716535597971
702d5946796838566a396f72337a7351624e6d4e6475797697472765843685476c66633055704
4545597d429869d34e5486a63963305f6a5267424f39745a6e4535438566a396fe64757969747
27658766c6655704417e252636f267349565f36d585a6e4547568f7672637835f4352484c5417
72e32556e32636a6f6a5267424f396349565f7869536d586357745a6e4535495475684973456a
4f76726378472e3f9 # Network token payload (as JWE)
```

If the reflect type is set to 0, the peer takes no additional step.
If reflect type is 0x1, the peer should attach a Network Token
request at the Binding Response message, set the type to 0x2, and
copy the rest of the token in it. The attribute data for the Binding
Response message that reflects the token mentioned above is listed
below.

```
803000e3  # Network Token Extension with 227 bytes length
2000001 # Network token is represented as JWT, reflected token
65794a68624763694f694a6b615849694c434a6c626d4d694f694a424d6a553251304a444c556
8544e544579496e302e2e565f4278546d4e692d6b6337735a376b504e514851412e5a42764269
6d6d46705579555165544579634b36244566a31378546d4e692d6b6337735a376b50476676c4
7597579666f734b43565234744576306a4d31735f4352484c50706a6335536d37703336487576
677541467c7979396f3a526e7976532484cd37703336487553559797170e514851412e5a42764
2696d6d4670557957955516576676c47597579666f734b43565234744576306a4d31716535597971
702d5946796838566a396f72337a7351624e6d4e6475796974727658436854766c66633055704
4545597d429869d34e5486a63963305f6a5267424f39745a6e4535438566a396fe64757969747
27658766c6655704417e252636f267349565f36d585a6e4547568f7672637835f4352484c5417
72e32556e32636a6f6a5267424f396349565f7869536d586357745a6e4535495475684973456a
4f76726378472e3f9 # Network token payload (as JWE)
```

   STUN messages can be protected for message integrity, and as such
   they can guarantee that network tokens cannot be dropped by
   intermediary nodes.

7.3. Network Tokens as an IPv6 Hop-by-Hop Extension Header

   Network tokens can be inserted as an IPv6 Hop-by-Hop Extension
   header, as defined in Section 4 of [RFC8200].

   Network tokens used as IPv6 extension headers can be either flow or
   packet specific. The expectation must be defined by the network
   service itself, and the endpoints can decide to which packets to
   insert a token. For example, they can insert a token at every packet
   of a specific flow, every few seconds, or only at the first packet of
   a flow. The network should accordingly implement the policy.

   When tokens are attached to all packets of a flow, it is important to
   keep the length of the token small, to avoid overhead. It is
   therefore recommended that in such cases implementations consider
   representation formats that can minimize the overall length of a
   token. Size-efficient representation formats are out-of-scope for
   this document.

   The token comprises the IPv6 extension header data. Section {#iana}
   requests the value 0x1F as a Network Token Extension Header. The
   following bytestream shows the header for the low-latency token
   described earlier, including option type and length (in 8-octet
   units).

```
1f1d  # Network Token Hop-by-Hop option with 29 8-octet length
1000001 # Network token is represented as JWT, reflect at node
65794a68624763694f694a6b615849694c434a6c626d4d694f694a424d6a553251304a444c556
8544e544579496e302e2e565f4278546d4e692d6b6337735a376b504e514851412e5a42764269
6d6d46705579555165e544579634b36244566a31378546d4e692d6b6337735a376b50476676c4
7597579666f734b43565234744576306a4d31735f4352484c50706a6335536d37703336487576
677541467c7979396f3a526e7976532484cd37703336487553559797170e514851412e5a42764
2696d6d4670557955516576676c47597579666f734b43565234744576306a4d31716535597971
702d5946796838566a396f72337a7351624e6d4e6475796974272765843684766c66633055704
4545597d429869d34e5486a63963305f6a5267424f39745a6e4535438566a396fe64757969747
27658766c6655704417e252636f267349565f36d585a6e4547568f7672637835f4352484c5417
72e32556e32636a6f6a5267424f396349565f7869536d586357745a6e4535495475684973456a
4f76726378472e3f9 # Network token payload (as JWE)
```

   If the reflect type is set to 0, the peer takes no additional step.
   If reflect type is 0x1, the peer should attach the Network Token Hop-
   by-hop option for messages in the reverse direction for this flow,
   set the type to 0x2, and copy the rest of the token in it. The data
   for the Hop-By-Hop option that reflects the token mentioned above is
   listed below.

```
1f1d  # Network Token Hop-by-Hop option with 29-octet bytes length
2000001 # Network token is represented as JWT, reflected token
65794a68624763694f694a6b615849694c434a6c626d4d694f694a424d6a553251304a444c556
8544e544579496e302e2e565f4278546d4e692d6b6337735a376b504e514851412e5a42764269
6d6d46705579555165e544579634b36244566a31378546d4e692d6b6337735a376b50476676c4
7597579666f734b43565234744576306a4d31735f4352484c50706a6335536d37703336487576
677541467c7979396f3a526e7976532484cd37703336487553559797170e514851412e5a42764
2696d6d4670557955516576676c47597579666f734b43565234744576306a4d31716535597971
702d5946796838566a396f72337a7351624e6d4e6475796974272765843684766c66633055704
4545597d429869d34e5486a63963305f6a5267424f39745a6e4535438566a396fe64757969747
27658766c6655704417e252636f267349565f36d585a6e4547568f7672637835f4352484c5417
72e32556e32636a6f6a5267424f396349565f7869536d586357745a6e4535495475684973456a
4f76726378472e3f9 # Network token payload (as JWE)
```

   TODO (YY): There is no clear way currently for peers to understand
   how to reflect tokens (per-packet, per-flow, and when). If this is
   understood by the context of the token the peer will need to be aware
   of the token, which is undesired. The token should have all
   information regarding reflection. We have to see whether the current
   two-bits can clarify this, or whether we need to make reflection part
   of the protocol-specific part and not the token itself.

8. Implementation Considerations

   Network token applications (and their implementations) must decide
   the contents of the token, what protocol to insert them to, whether
   tokens are per-packet or per-flow, and whether they need to be

reflected by peers on the reverse flow. This section discusses
common consideration.

8.1. Contents

When deciding the contents of a token, applications should take
effort to include only the necessary information and keep the size of
the token small. They should also take into consideration the trust
relationships between different stakeholders (i.e., the network
operator, the application provider, the operating system, the end
user) and pick the right encryption and signing properties. Another
element to consider is potential abuse scenarios and how to prevent
token replays and protect from malicious users, given assumptions
around the network architecture. For example, to prevent use of a
user-specific token by a another user, a token might be bound to a
user identifier that the network can separately verify while
processing packets and tokens (e.g., a well-defined IP address or an
MSISDN in case of a cellular network).

8.2. Encapsulating protocol

This specification describes three protocols where tokens might be
inserted, and it is expected that this list will grow.

IPv6 tokens sit at the narrow waist, can be applied to any traffic,
and support both per-packet and per-flow granularity. IPv6 tokens
require OS support for any practical usecase. The downside of IPv6
tokens is that IPv6 adoption is still limited, and many networks drop
packets with IPv6 extension headers with rates described in
[RFC7872]. However, this is expected to improve over time.

TLS tokens are limited to TLS/DTLS sessions, and can only be used at
a flow granularity. Implementation does not depend on the OS, but
rather at the cryptographic library used (e.g., BoringSSL or
OpenSSL). As TLS handshake is integrity-protected, intermediate
nodes cannot drop or alter a token.

STUN tokens are targeted to real-time communications, and can only be
used at a flow granularity. Implementation does not depend on the
OS, but rather at the library that initiates the related flows (e.g.,
WebRTC). As STUN messages are integrity-protected, intermediate
nodes cannot drop or alter a token.

The primary reason for making network tokens protocol agnostic is to
ease adoption and enable interested parties (Operating System,
application providers, users, network vendors and operators) to use
them in a way that better fits the intended usecase. The protocol to
use will largely depend on the parties involved in a usecase, the

availability of tokens in different protocols, and the capability of endpoints and the network to insert and interpret tokens.

When possible, networks should make efforts to accept tokens in different protocols to allow further adoption from endpoints. As processing of tokens remains the same independent of the protocol that carries them, the main overhead from detecting tokens in multiple protocols will come from parsing and detecting tokens in different parts of the header.

8.3. Network Token granularity

The granularity of a token is either per-packet or per-flow and is closely related with the protocol where tokens are inserted. The trade-offs for each option are discussed below.

8.3.1. Per-packet granularity

Per-packet granularity allows stateless processing from network nodes and the capability to pinpoint the exact packets for traffic differentiation (e.g., for flows that combine multple types of traffic). Moreover, it preserves policy if the flow of interest gets rerouted.

On the other hand, per-packet granularity limits potential encapsulating protocols. From the protocols described in this specification only IPv6 allows per-packet granularity.

Per-packet granularity implies that all packet processing will involve cryptographic functions which might be expensive (or unavailable). Finally, the size of the token should be small as it will add overhead to all packets.

8.3.2. Per-flow granularity

Tokens with per-flow granularity can be inserted in multiple protocols. As the token is sent only once per flow, its size is less important compared to per-packet tokens. Limiting crypto to one packet per flow reduces the packet processing overhead, and allows implementations that combine a fast path (with no crypto support) with a slow, crypto-enabled path.

The main disadvantage of per-flow granularity is that it requires per-flow state. As policy enforcement depends on this state, per-flow granularity also requires that all packets of this flow will get routed through the node that interprets tokens.

8.4. Token to DiffServ mapping and reflection

Tokens might be interpreted one or more times within a network.

In many cases, a single interpretation will be enough. One example
is network policies that are enforced at a single point and involve
per-flow state (like zero-rating). In such scenarios, token
interpretation and policy enforcement can take place all in once.
Alternatively, when policy enforcement involves multiple nodes (e.g.,
a low-latency service that spans the wired and radio network),
network owners can use existing mechanisms (like DiffServ, QCI tags,
or reflective QoS) to enforce the policy across multiple nodes within
the same network domain.

In cases where a reverse flow might get routed through a different
path, token reflection should be used.

9. Security Considerations

As any cryptographic application, it is important for users of
network token applications to protect asymmetric private and
symmetric secret keys, and employ countermeasures to various attacks.

The security of network tokens relies upon on the protections offered
by the underlying signing and encryption technologies. It is
therefore recommended that implementations of network tokens use
existing and well-understood cryptographic frameworks (like JOSE and
COSE) to protect tokens, or careful consider security implications if
they provide their own format.

While tokens are integrity protected, an intermediary node can in
theory replace or remove a token. Protection against this can be
provided by additional integrity protection from the encapsulating
protocol itself, as is the case with TLS handshakes and STUN.

Network tokens may require processing in software, as current
hardware platforms do not support cryptographic capabilities. This
might impose a security risk and exposure to an attack, as traffic
could be diverted towards the slow path, and in return degrade the
overall performance of a node. It is recommended that
implementations adequately account for such scenarios, either by
setting a rate-limit to packets that go through a slow path or
ensuring that the overall functionality is not affected.

[10](). IANA Considerations { #iana }

[10.1](). Token Descriptor ID Registry

   This section establishes the IANA "Network Token Descriptor ID"
   registry for token descriptors. The registry records the descriptor
   ID and a reference to the specification that defines it.

   Values are registered on a Specification Required [[RFC5226]()] basis
   after a three-week review period, on the advice of one or more
   Designated Experts. However, to allow for the allocation of values
   prior to publication, the Designated Experts may approve registration
   once they are satisfied that such a specification will be published.

   Within the review period, the Designated Experts will either approve
   or deny the registration request, communicating this decision to the
   review list and IANA. Denials should include an explanation and, if
   applicable, suggestions as to how to make the request successful.

   Criteria that should be applied by the Designated Experts includes
   determining whether the proposed registration duplicates existing
   functionality, whether it is likely to be of general applicability or
   whether it is useful only for a single application, and whether the
   registration description is clear.

[10.1.1](). Initial Registry Contents

   o Token Descriptor ID: 0x1

   o Description: Token is represented as a JSON Web Token

   o Specificaton Document(s): This document.

   o Token Descriptor ID: 0x2

   o Description: Token is represented as a Concise Binary
      Representation Object

   o Specification Document(s): This document

[10.2](). IPv6 Hop-By-Hop options registration

   This section registers the value 0x0F as a IPv6 Hop-By-Hop and
   Destination Option for network tokens.

   o Hex Value: 0x1F

   o Binary Value: 0x00011111

o Description: Network Token

o Reference: This document

[10.3]. TLS ExtensionType Registry

This section registers the value 57 as a TLS Extension Type for
network tokens.

o Value: 57

o Description: Network Token

o Reference: This document

[10.4]. STUN Attributes Registry

This section registers the value 0x8030 as a STUN attribute for
network tokens.

o Value: 0x8030

o Description: Network Token

o Reference: This document

[11]. References

[11.1]. Normative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an
           IANA Considerations Section in RFCs", RFC 5226,
           DOI 10.17487/RFC5226, May 2008,
           <https://www.rfc-editor.org/info/rfc5226>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing,
           "Session Traversal Utilities for NAT (STUN)", RFC 5389,
           DOI 10.17487/RFC5389, October 2008,
           <https://www.rfc-editor.org/info/rfc5389>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web
           Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May
           2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)",
           RFC 7516, DOI 10.17487/RFC7516, May 2015,
           <https://www.rfc-editor.org/info/rfc7516>.

   [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token
              (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015,
              <https://www.rfc-editor.org/info/rfc7519>.

   [RFC7872] Gont, F., Linkova, J., Chown, T., and W. Liu,
              "Observations on the Dropping of Packets with IPv6
              Extension Headers in the Real World", RFC 7872,
              DOI 10.17487/RFC7872, June 2016,
              <https://www.rfc-editor.org/info/rfc7872>.

   [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6
              (IPv6) Specification", STD 86, RFC 8200,
              DOI 10.17487/RFC8200, July 2017,
              <https://www.rfc-editor.org/info/rfc8200>.

   [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig,
              "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392,
              May 2018, <https://www.rfc-editor.org/info/rfc8392>.

   [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol
              Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018,
              <https://www.rfc-editor.org/info/rfc8446>.

11.2. Informative References

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119,
              DOI 10.17487/RFC2119, March 1997,
              <https://www.rfc-editor.org/info/rfc2119>.

   [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z.,
              and W. Weiss, "An Architecture for Differentiated
              Services", RFC 2475, DOI 10.17487/RFC2475, December 1998,
              <https://www.rfc-editor.org/info/rfc2475>.

Authors' Addresses

   Yiannis Yiakoumis
   Selfie Networks, Inc

   Email: yiannis@selfienetworks.com

   Nick McKeown
   Stanford University

   Email: nickm@stanford.edu

Frode Sorensen
Norwegian Communications Authority

Email: frode.sorensen@nkom.no