RateLimit Header Fields

Presented by Darrel Miller

IETF 121 Dublin

What is used in the wild today

```
GET /example

HTTP/1.1 200 OK
Date: Sat, 26 Oct 2024 15:00:00 GMT
Content-Type: application/json
Content-Length: 123
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 10
X-RateLimit-Reset: 3600
```

But there are variants of these, in syntax and semantics

What was proposed at WG Adoption

```
RateLimit-Limit: containing the requests quota in the time window;
RateLimit-Remaining: containing the remaining requests quota in the current window;
RateLimit-Reset: containing the time remaining in the current window, specified in
seconds.
```

The goal was to standardize, to improve interoperability, but maintain compatibility as much as possible.

```
RateLimit-Limit: 100, 100;window=10
```

And optionally provide additional policy information

WG decided on evolution over compatibility

```
RateLimit-Policy: 100;w=10
RateLimit: limit=100, remaining=50, reset=5
```

Policy remains optional

Named Policies

```
RateLimit-Policy: "default";q=100;w=10
RateLimit: "default";r=50;t=30
```

Simplify correlation between quota policy and current service limit

Consistent use of single character parameters

Quota Units

```
RateLimit-Policy: "egress";q=65535;qu="content-bytes";w=10
```

requests:
This value indicates the quota is based on the number of requests processed by the resource server. Whether a specific request actually consumes a quota unit is implementation-specific.

content-bytes:
This value indicates the quota is based on the number of content bytes processed by the resource server.

concurrent-requests:
This value indicates the quota is based on the number of concurrent requests processed by the resource server.

Specification requests a registry for these values

Achieving Interoperability

Observation: There appear to be very few generic client-side handlers of Rate Limiting headers. There are some service specific uses of these headers but very minimal in comparison to usage of “retry-after”.

Hypothesis: Generic client code is only usable if a client application is only operating on a single quota “partition”. That is rarely the case.

Quota Policies define one dimension. Usage over time. They do not address usage of what and by who. All requests are considered equal.

API providers need to protect their service and their business by allocating capability based on a wide range of request attributes. E.g. resource, method, end-user, application.

Previous conversations used the term “scope” and interoperable support was deemed “out of scope” for this document.

Quota Partition

Quota Partition:
A quota partition is a division of a server's capacity across different clients, users and owned resources.

This draft introduces a Partition Key parameter to allow servers to advertise which “partition” the quota and service limit is associated to.

```
RateLimit-Policy: "peruser";q=100;w=60
RateLimit: "peruser";r=30;t=10;pk=:cHsdsRa894==:
```

Why is a partition key useful?

- Clients and intermediaries can use the partition key to track independent service limits.
- If clients know how to generate partition keys via out-of-band knowledge, they can proactively defer requests based on advertised policies.
- Generic clients could use heuristics to predict partition keys.
- Formalizing the notion of partition keys facilitates the creation of generic code that reacts to rate limiting fields.

Predicting a PartitionKey on the client

```python
def CreatePartitionKey(req):
    # Extract the HTTP method
    method = req.method

    # Extract the first path segment
    path_segment = req.path.split('/')[1]

    # Decode the JWT token
    token = req.headers.get('Authorization').split()[1]
    decoded_token = jwt.decode(token, options={"verify_signature": False})
    subject_claim = decoded_token.get('sub', 'unknown')

    # Construct the string value
    result = f"{method}:{path_segment}:{subject_claim}"

    # base64 encode the result
    return result.encode('utf-8').hex()
```

Illustrative example only. Code was mostly written by an LLM.

Should we make this truly generic?

Should the construction of partition keys defined in a similar way to how HTTP Message Signatures are?

```
Signature-Input: sig1=("@method" "@target-uri" "@authority" \
  "content-digest" "cache-control");\
  created=1618884475;keyid="test-key-rsa-pss"
```

Example (with significant amounts of hand waving):

```
RateLimit-Policy: "peruser";q=100;w=60;pkd=("@method" "@authority"
"@jwtclaim-subject" "@jwtclaim-client-id")
```

Specific areas needing feedback

- String vs Token for quota policy identifier
- Initial registry of quota units
- Is partition key parameter valuable enough?
- Is a way of communicating partition key format in-band essential?
- Is pk needed on RateLimit-Policy or just RateLimit?