IETF QUIC Deployment

Wei Sun
sunwei@kuaishou.com

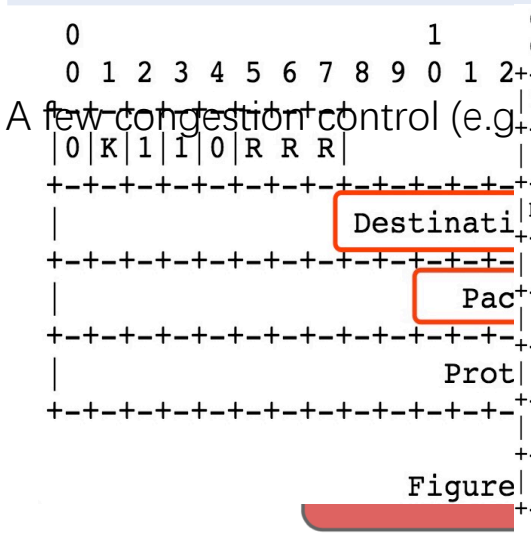
Outline

- Background
- Deployment
- Test Results
- Future Work

Background – Why QUIC?

TCP Issues	QUIC Solution
1. Conn latency. HTTP: 1-RTT, HTTPS: 3-RTT, 2RTT(TLS 1.2)	1. First conn: **1-RTT**， subsequent conn: **0-RTT** (TLS 1.3)
2. HOL blocking	2. Multi-streams avoids HOL blocking
3. 3G/4G/WIFI switching requires new connection	3. Encode hostId in DCID， LB routes packet by DCID
4. Retransmission ambiguity (inaccurate RTT)	4. Unique packet number for each packet

A few congestion control (e.g.



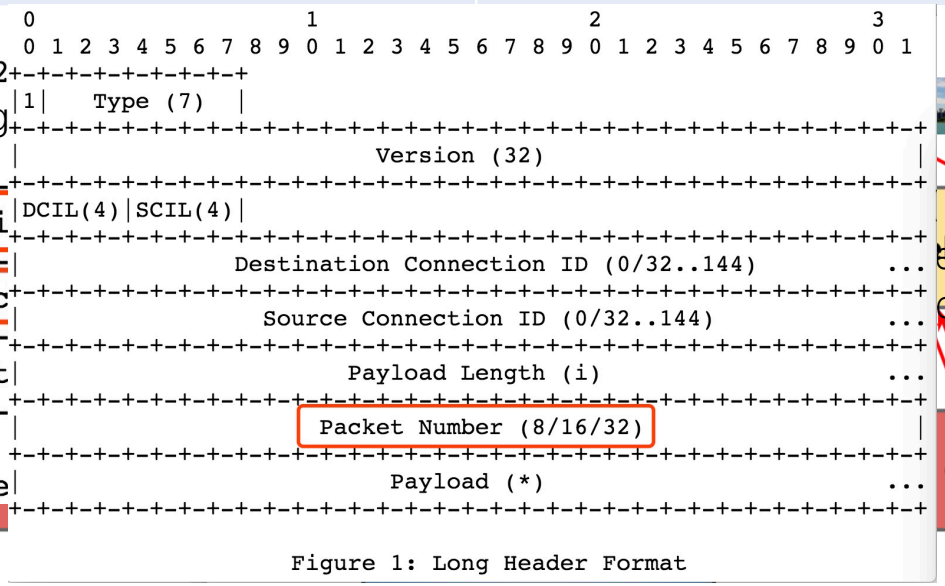
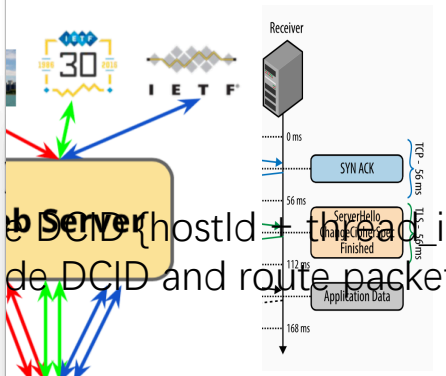


Figure 1: Long Header Format



e DCID {hostId + thread_id}
de DCID and route packet by
use， **2-RTT**)

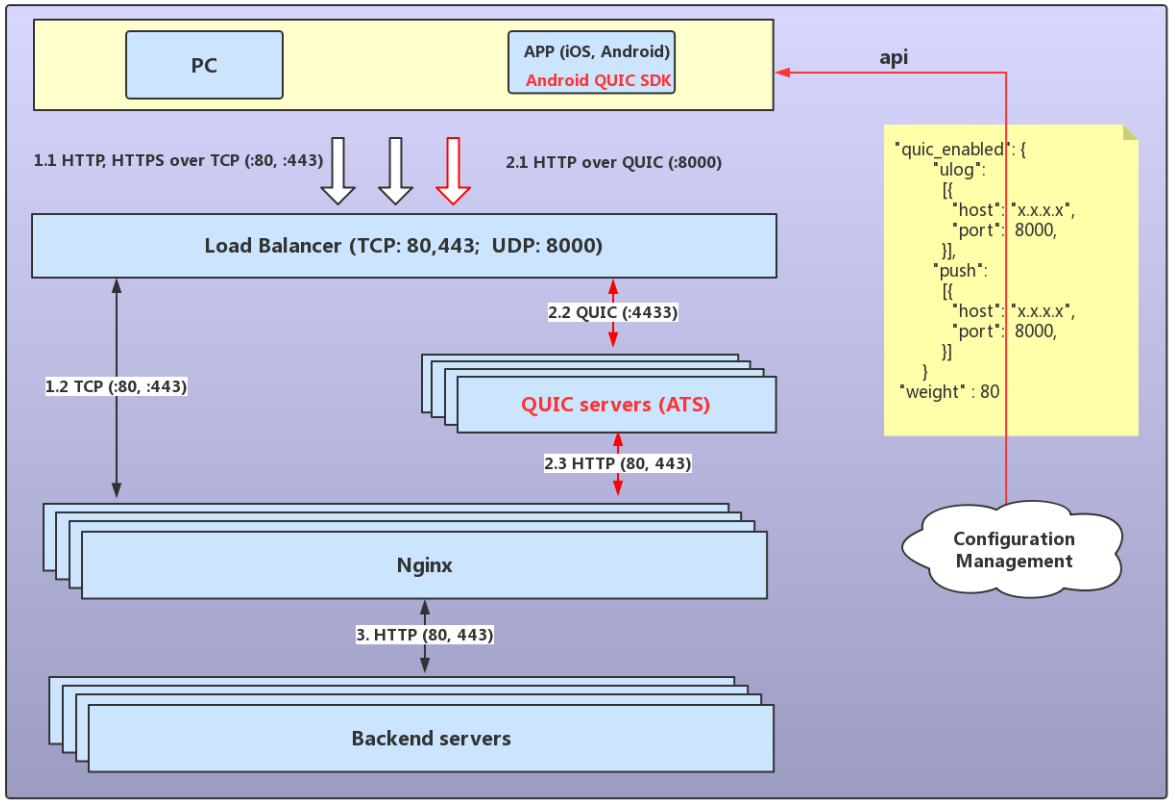
(Images are from internet)

Outline

- Background
- Deployment
  - System Architecture
  - Connection Migration
  - Congestion Control
  - 0-RTT
  - Others
- Test Results
- Future Work

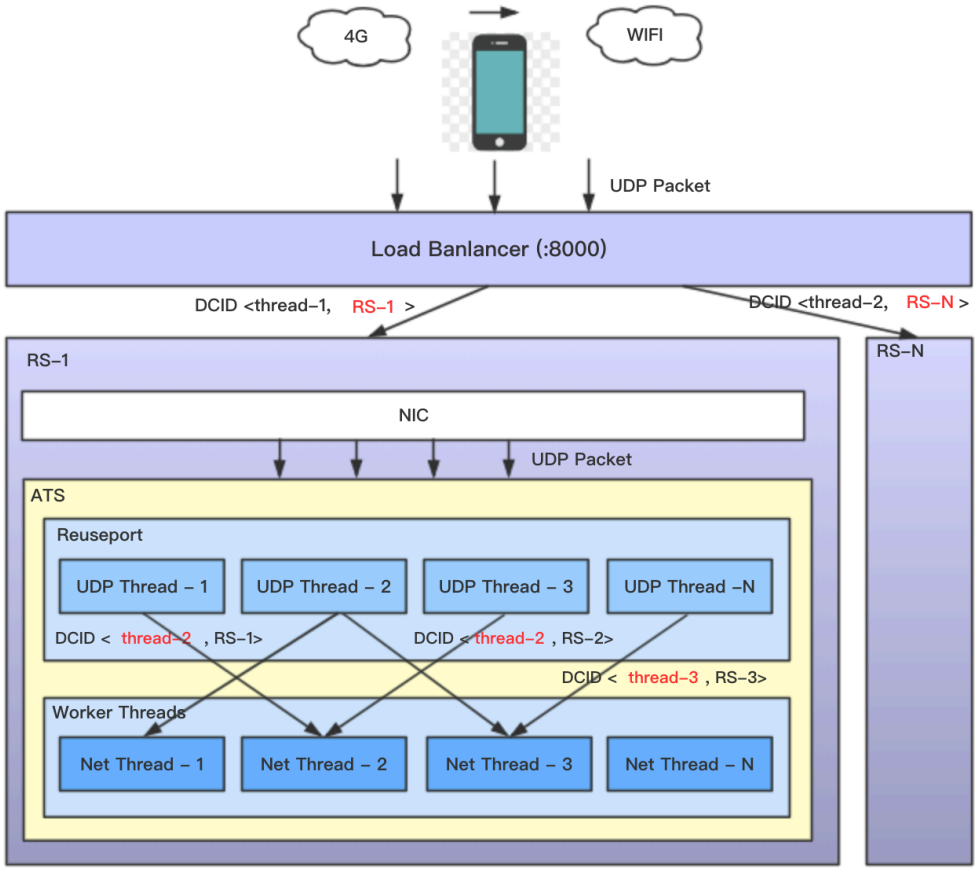
Deployment - System Architecture

- **Client**: quic_sdk
- **Server**: ATS (quic draft-12)
- **Config**: quic related configuration



Deployment - Connection Migration

- **Client SDK**: re-send req when IP address changes
- **LB**: route the UDP packet by address in DCID
- **Server**:
  - DCID generation: encode hostId+ thread id in DCID
  - Decode DCID and assign UDPPacket to the thread by thread id
  - Support SO_REUSEPORT



Deployment - Congestion Control

- **New Reno**: Default CC (loss-based) in IETF QUIC
- **BBR**: Developed by Google in 2016, available in Linux 4.9+ and gQUIC
  - *Congestion window = max BW * min RTT * cwnd_gain*
  - *Pacing rate = max BW * pacing_gain*
- **Our work**:
  - Integrated with BBR v1 (by Beixing Zuo)
  - Configurable CC modes, BBR is the default CC on production.
  - Test: BBR is **10x+ faster** than New Reno for transferring 1MB data in 5% packet-loss environment

Deployment - 0-RTT Support

- Stateless TLS session reuse: SSL_OP_NO_ANTI_REPLAY
- Store the PSK in APP storage for 1 week
- 0-RTT reused ratio: **79% ~ 98%**

Deployment - Others

- Memory issue fix
- Performance optimization
- QUIC related metrics

Outline

- Overview
- Deployment
- Test Results
- Future Work

Test Results – A/B Test

- Serving partial user API traffic (HTTP & HTTPS) in production
- **Success Ratio**: increase **+1.1pp**
- **Latency Result**: Average HTTP (**-28%**), HTTPS (**-48%**)

Type	Avg Latency	P25 Latency	P50 Latency	P75 Latency	P90 Latency	P95 Latency	P99 Latency
HTTP	-28%	-10%	-10%	-8%	-13%	-31%	-53%
HTTPS	-48%	-28%	-58%	-63%	-61%	-48%	-52%

(updated at 2019.4.27)

Test Results – Server Performance

- 1. Non-reused TLS conn: each request initiates a new TLS conn
- 2. Mixed-reused TLS conn: new conn / 0-RTT / reused conn = 1/10/50

	RPS (K)	Avg Latency (ms)	P99 Latency(ms)
1. Non-reused TLS Conn	37.5	14.8	50.7
2. Mixed-reused TLS Conn	100.4	10.4	49.6

Outline

- Overview
- Deployment
- Test Results
- Future Work

Future Work

- Performance optimization
  - UDP GSO, Zero copy
  - TLS hardware acceleration
- Update to BBR v2
- Remove crypto for an internal use case
- Co-work with ats-quic and contribute some work