FASTUDX · UDX3.0

FastUDX 技术白皮书

面向实时场景的可靠 UDP 传输协议

版本 v1.2 | 2026-11-04 更新 | 武汉蛮牛科技有限公司

这本白皮书怎么读

同一份材料，三类读者的关注点完全不同。按你的角色挑入口，5 分钟就能拿到你要的答案。

决策者

看 第 5 章的两组实测（5.3 一对一交替、5.4 十二家同场）与 第 6 章选型表，判断「值不值得替换现有 TCP / KCP 方案」。

工程师

看 第 3、4 章的架构与机制，直接跳到 8.2 抄走一段可编译代码，再下载 Demo 跑通第一条流。

采购 / 项目负责人

看 第 7 章适用场景、第 9 章授权与交付；附录 A 的术语表可直接搬进你的方案文档。

1 摘要

FastUDX（UDX3.0）是一套完全自主研发、纯 C++ 实现、零第三方依赖的**可靠 UDP 传输协议**，面向实时音视频、文件传输、金融行情、物联网等对**低延迟**与**弱网稳定性**要求苛刻的场景。

自 2005 年第一版至今持续迭代 20 余年，FastUDX 已在 Windows、Linux、macOS、Android、iOS 与嵌入式 ARM 六大平台稳定运行，服务 30+ 家企业客户，覆盖智能家居、安防监控、无人机与机器人控制、物联网、楼宇对讲、分布式办公、即时通讯等领域。

核心能力一览：

- **超低延迟**：局域网端到端延迟 < 1 ms，相比 TCP 减少 50%+ 的等待时间，无队头阻塞；
- **强抗丢包**：自适应 FEC 前向纠错 + ARQ 选择性重传，30% 丢包环境下保持流畅传输；
- **多路径聚合**：WiFi / 4G / 5G / 有线组成一个统一预算的通道，主控统一持有段编号与重传账本；单链路中断时在途数据在判死瞬间转由其余链路承载，业务不中断，路径恢复后自动绑回（含故障注入实测，见 4.5）；
- **P2P NAT 穿透**：内置 Cone / Symmetric NAT 穿透，中继转发兜底，确保 100% 连通率；

- **无锁内核**：自研无锁链表、无锁 Map、无锁内存池，基于 EBR（Epoch-Based Reclamation）内存回收，彻底消除锁竞争；
- **全平台零依赖**：纯标准 C++ 接口，凡支持 C++ 的环境均可运行，100% 自主可控。

我们不只讲架构，也给可复现的实测。2026-10-31，在一条 RTT 约 260 ms、丢包 1.7%~6.7% 波动的跨境链路上，以 30 秒固定窗口、单连接下载 1 GiB、严格 A/B 交替的方式，与开源 QUIC 实现 ngtcp2（BoringSSL，BBR 拥塞控制）做了 9 组对照：**FastUDX 9 组全胜，中位吞吐 5.40 MiB/s 对 2.18 MiB/s，中位领先 2.48 倍。**脚本、判定口径与原始日志全部公开，欢迎任何同类方案在同等条件下复核与挑战（详见 5.3）。

同月我们还做了一次 **12 家开源 QUIC 实现同场横评**（同一条链路、交错轮转 3 轮）。在**高时延（RTT ≈ 240 ms）+ 高丢包（ICMP 8.9%~24.4%）的跨境公网、单连接大文件下载**这一限定场景下，29 组有效组合中 FastUDX 以 8.35 MiB/s 列第 1，xquic-Copa 8.14 紧随其后（差 2.5%，同档），第三名 picoquic-bbr1 仅 3.68 MiB/s；cubic / reno 系在同等链路下最好也只有 0.037 MiB/s。ngtcp2 三组因该轮服务端 GSO 失效、链路丢包统计失真，数据不采信并已整体排除。参测名单、排除依据、逐轮原始数据与适用边界见 5.4。

---

2 背景：实时传输为什么难

2.1 TCP 的先天局限

TCP 为"可靠"付出了实时性的代价：

- **队头阻塞（Head-of-Line Blocking）**：单个丢包阻塞整条字节流，后续已到达数据无法交付；
- **RTO 放大**：丢包后重传定时器可达数百毫秒（尤其在高 RTT 链路），实时业务无法承受；
- **拥塞窗口震荡**：丢包即减半的拥塞控制策略导致发送速率锯齿波动，音视频出现卡顿与花屏；
- **内核协议栈难以定制**：面向通用场景设计，无法按业务定制重传策略、FEC、多路径。

2.2 裸 UDP 的不可靠

裸 UDP 无连接、无重传、无排序、无流控，直接把可靠性留给应用开发者，等于让每个项目重复造一轮传输层——这正是 FastUDX 要消灭的重复劳动。

2.3 现有 UDP 可靠方案的选择困境

方案	定位	局限

KCP	纯 ARQ 快速重传，轻量	功能单一：无内置 FEC、无多路径、无 P2P/NAT、无拥塞控制框架
QUIC	面向 Web 的传输协议（HTTP/3）	强制 TLS 1.3、实现重、资源占用高，嵌入式与纯 UDP 业务集成成本大
SRT	面向广电视频贡献	场景聚焦音视频单向流，通用性与嵌入式覆盖有限
UDT	早期可靠 UDP 实现	维护停滞，生态薄弱

结论：需要一个**通用、轻量、可嵌入、可深度定制**的可靠 UDP 传输层。FastUDX 应运而生。

---

3 总体架构

FastUDX 采用分层架构，各层可独立裁剪，适配从服务器到嵌入式 ARM 的全谱系场景：

应用接入层	C++ API (FastUdx.h) · C API (udxapi.h) · P2P 平台 SDK
能力层	多路径聚合 · P2P/NAT 穿透 · 文件传输 · 媒体推送 · 加密
可靠传输层	自适应 FEC + ARQ · 拥塞控制 · 接收窗口流控 · 分片重组
高性能数据面	无锁队列 · 无锁 Map · 内存池 (EBR) · 零拷贝路径
Socket 抽象层	UDP · 多网卡绑定 · 跨平台网络抽象

---

4 核心机制

4.1 混合纠错：自适应 FEC + ARQ

纯重传方案必须等待一个 RTT 才能修复丢包；纯 FEC 方案在零丢包时浪费带宽。FastUDX 采用混合策略：

- **自适应 FEC 前向纠错**：冗余率随实测丢包率动态调整，多数丢包在一个 RTT 内被直接恢复，无需重传；

- **ARQ 选择性重传**：对 FEC 未覆盖的丢失进行精准补发，快速重传基于 ACK 空洞检测触发；
- **去重与乱序重组**：接收侧自动消除重复包、重组乱序分片，对上交付有序数据流。

4.2 接收窗口流控与毫秒级背压恢复

FastUDX 的接收侧建立**全链路挂账可见**的流控体系：

- 接收缓冲占用（协议内部 + 应用层队列）实时上报，窗口随消费进度动态通告；
- 接收窗口塌 0 时立即通知对端停发，从根本上杜绝"快发慢收"导致的缓冲膨胀与丢包；
- 窗口恢复采用**事件驱动 + 定时探针双通道**：消费恢复后发送侧毫秒级感知并恢复发送，不依赖单一超时兜底。

该机制已在可复现的"闸门测试"中验证：接收方暂停读取 → 发送方在约一个窗口内被精确制止 → 接收方恢复读取 → 发送在毫秒级恢复。任何人可下载 Demo 复现该场景。

4.3 拥塞控制与平稳发送

- 内置带宽探测与拥塞控制框架，发送速率随链路容量自适应；
- 流量整形保证**发送间隔均匀**，无 TCP 的锯齿与突发抖动——这正是音视频场景流畅性的关键；
- 不依赖丢包即减半的激进退让策略，在高丢包链路保持吞吐。

4.4 无锁内核与 EBR 内存回收

多线程是高性能服务端的标配，锁竞争是性能的头号杀手。FastUDX 数据面全链路无锁化：

- **自研无锁链表 / 无锁 Map / 无锁内存池**，热点路径无任何 lock 前缀重型操作；
- 基于 **EBR（Epoch-Based Reclamation）** 的安全内存回收，读写并发不阻塞；
- 单核即可跑满千兆带宽，多核横向扩展到万兆无锁瓶颈。

4.5 多路径聚合与链路故障接管

多路径要解决的首要问题不是"一条链路不够快"，而是**一条链路会断**。手机在 WiFi 与 4G 之间切换、单运营商出口故障、机器人穿越不同基站、跨境链路整体抖动——这些都是链路级失效，不是带宽不足。FastUDX 的多路径按这个前提设计：多条链路组成一个统一管控的通道，任一链路退出都不影响数据流的连续性。

- **一个预算、一个账本**：主控连接统一持有段编号空间、重传账本与窗口预算，各子流保有独立 Socket 与独立 RTT 记录。全通道只有一条编号空间，不存在"两条链路各发一半、对端拼不起来"；同一段在任意子流上补发即去重，不重复占用窗口。

- **三道写入闸门**：总预算闸门（未确认字节超过总窗口的 4 倍即停写）、段级在建量闸门（已建未写段数超过 `max(250, 总窗口/256)` 即停写）、子流可用性闸门（只有全部可用子流都写不进时才判“满”）。“路径越多、在途越多”的膨胀被挡在写入侧，内存占用与排队时延不随路径数线性增长。
- **判死与接管在同一毫秒**：一条路径约 10 秒无回包即判死（实测 9.93 ~ 10.99 秒，随心跳次数与超时参数浮动）。判死瞬间，该路径的在途数据**同毫秒**转入主控账本，由其余链路继续发送——不存在“断了才想起来搬数据”的窗口。
- **主控失效不影响会话**：只要还有任意一条子流可发，会话就继续。实测主控连接永久中断（子路全活）时，文件照样完整传完；主控与一条子路同时永久中断、只剩一条子路时亦然。
- **路径可自动绑回**：判死路径按 10 秒周期重试重加入，链路恢复后自动绑回同一主控，路径数补齐。

以下为可复现的**链路故障注入实测**：跨境链路（RTT 约 260 ms）、服务器出口限速 12 Mbit/s、单文件 128 MiB、3 条路径，故障在传输进行中注入。判据不是“连接没断”，而是**落盘文件的 MD5 与源文件一致**：

注入的故障	行为	完成耗时	落地完整性
无故障（基线）	—	98 s	MD5 一致
一条子路永久中断	判死瞬间在途数据转入主控，剩余链路独扛	99 s	MD5 一致
主控连接永久中断（子路全活）	会话不中断，由剩余子路发完	100 s	MD5 一致
一条子路 + 主控永久中断（剩 1 条子路）	同上	102 s	MD5 一致
全部路径同时中断 45.5 s	子路判死后由主控独扛，等待链路恢复	147 s	MD5 一致

> **能力边界与工程建议**：① **聚合收益取决于链路组合**——吞吐由各链路可用带宽、统一持有的总预算与调度开销共同决定；收益最明显的场合是“单链路接近饱和而另一条空闲”或“单链路时延抖动剧烈需要错峰发送”。② 只要还有一条路径活着，会话就能传完（表中五行全部通过），单条链路失效对总耗时的影响在个位数秒级：基线 98 s，一条子路永久中断 99 s，主控永久中断 100 s；生产部署建议至少两条物理独立的链路。③ 路径重加入有代价：同样 45 秒全断，开启重加入的单次传输可能比关闭时更慢（实测慢 41%），它的收益在长连接与持续业务，不在单次传输的墙钟时间。自测方法见 5.2。

4.6 P2P NAT 穿透与中继兜底

- 内置 P2P NAT 穿透模块，覆盖 Cone NAT、Symmetric NAT 等主流类型；
- 穿透失败自动降级**中继转发**，确保 100% 连通率；
- 提供 P2P 接入平台 SDK（DevAppSDK.h）：设备注册、激活、在线管理、智能路由、音视频推拉流、云台控制、云存录像——支持私有化部署与独立运营。

---

5 性能特征与实测验证

5.1 已公开性能指标

指标	数值	条件
端到端延迟	< 1 ms	局域网
延迟对比 TCP	降低 50%+ 等待时间	无队头阻塞
抗丢包	30% 丢包下流畅传输	自适应 FEC + ARQ
吞吐	单核跑满千兆	高吞吐数据面
发送形态	均匀平稳	无突发抖动
平台覆盖	6 平台	Win / Linux / macOS / Android / iOS / 嵌入式 ARM
跨境弱网对照（vs ngtcp2-BBR）	中位领先 2.48 倍（9/9 全胜）	RTT 约 260 ms、丢包 1.7%~6.7%，口径见 5.3

5.2 建议的自验证方法（可复现）

不轻信任何白皮书数字——包括本文。FastUDX 提供全平台 Demo，建议按以下路径自测：

1. 从官网下载中心获取 Windows / Linux / Android 测试程序；
2. 局域网直连测试基线延迟与吞吐；
3. 使用网络损伤工具（如 Linux `tc netem`、Windows Clumsy）注入 5% / 30% 丢包，对比 FastUDX 与 TCP 的表现；

4. 执行"闸门测试"验证背压恢复：接收端暂停读取再恢复，观察发送侧毫秒级停顿与恢复。

5.3 对照实测：跨境弱网 vs ngtcp2（QUIC，BBR）

5.40 vs 2.18	**9** / 9 组	**2.48** 倍
FastUDX vs ngtcp2-BBR · MiB/s 中位	严格「先 A 后 B」交替 · 全胜	中位领先 · 单组区间 1.73x ~ 3.20x

性能结论只有两种可信形式：可复现，或可证伪。以下是一次完整的对照实测，并保留对 FastUDX 不利的波动（见下表最低值那几轮），不做有利挑选。

5.3.1 规则：我们凭什么比较，也接受同样的挑战

- **同一时刻**：两种协议在同一时间窗内交替运行，抵消链路时变；不允许一方只跑在链路最好的时段；
- **同一口径**：只统计"应用层拿到的字节 ÷ 实际耗时"，不采信内核计数、瞬时峰值与跑分截图；
- **同一参数**：固定 30 秒窗口、单连接、1 GiB 文件、下载方向、默认收发缓冲；
- **可复现**：脚本、参数、原始日志全部公开，任何人可重跑；
- **可证伪**：原始数据按轮次列出，不筛选、不删点。

5.3.2 环境与参数

项	配置
链路	本机（WSL2，NAT 出网）↔ 海外 VPS，跨境 RTT 约 260 ms，实测丢包 1.7% ~ 6.7% 波动
被测 A	FastUDX（UDX3.0，构建 7e1d3137，非 FEC 变体）
被测 B	ngtcp2（开源 QUIC 实现，BoringSSL examples），拥塞控制 BBR
数据	1 GiB 随机文件，单连接，仅下载
窗口	固定 30 s，超时强制中断，结算已交付字节
样本	3 批 × 3 组 = 9 组，每组严格"先 A 后 B"交替

5.3.3 结果（单位 MiB/s，均为应用层交付）

组	批次	FastUDX	ngtcp2-BBR	倍数	该轮实测丢包
1	A	4.65	2.31	2.01x	12.82%
2	A	6.70	2.33	2.88x	6.33%
3	A	5.44	3.00	1.81x	6.91%
4	B	5.40	3.12	1.73x	11.12%
5	B	5.87	2.04	2.88x	6.45%
6	B	5.78	2.13	2.72x	6.72%
7	C	3.30	1.03	3.20x	6.87%
8	C	4.17	1.38	3.03x	10.53%
9	C	4.53	2.18	2.08x	17.00%

汇总：FastUDX 中位 5.40 MiB/s、均值 5.09；ngtcp2-BBR 中位 2.18、均值 2.17；**中位领先 2.48 倍、均值领先 2.36 倍，9 组全胜**（单组倍数区间 1.73x ~ 3.20x）。

5.3.4 数据说明了什么

批次	链路丢包	FastUDX 均值	ngtcp2-BBR 均值	倍数
A	1.67%	5.60	2.55	2.20x
B	约 1.7%	5.68	2.43	2.34x
C	3.33% → 6.67%	4.00	1.53	2.62x

- A、B 两批链路条件相近，结果也相近（5.60 对 5.68，倍数 2.20x 对 2.34x），说明测量可重复；
- C 批丢包抬到 3.3%~6.7%，FastUDX 下滑 29%，ngtcp2-BBR 下滑 40%——**丢包越重，差距越大**，与自适应 FEC + ARQ 混合纠错的设计目标一致；
- 全程 RTT 稳定在 259~291 ms，倍数差异不来自 RTT，而来自对丢包的处理方式；

- 需要说清楚：本次绝对值低于我们 10 月 29 日在另一链路条件（18.3% 丢包）下测得的 9.64 MiB/s。不同链路之间不能横向比较，我们只比较**同一时刻的对手**。

5.3.5 欢迎复现，欢迎挑战

1. 下载 Demo，准备两台机器（或直接跨境），用 `tc netem`、Clumsy 注入 1%~10% 丢包；
2. 固定 30 s、单连接、下载 1 GiB，只统计应用层交付字节 ÷ 耗时；
3. 两种协议交替各跑 3 轮以上，取中位；
4. 把脚本、原始日志、机器与链路信息发到 wwwllg@126.com，我们复核后**无论输赢都公开结果**，并在本白皮书与官网更新中如实标注。

KCP、QUIC（quiche / ngtcp2 / quinn / msquic）、SRT、UDT 以及任何自研可靠 UDP 实现，我们都愿意同条件比较。我们相信：**能被打败的方案，才配被信任。**

> 边界声明：以上数据仅代表"该链路 + 该实现 + 该参数"下的对照结果，不构成对 QUIC 协议本身的否定。ngtcp2 的 cubic 实现在本链路 30 s 内仅交付数百 KB，无法形成有效对照，故只采用 BBR；如你认为参数选择对方不利，欢迎按你的建议重测。

5.4 十二家 QUIC 实现同场横评：高时延高丢包跨境链路下的相对位置（2026-10-27）

8.35 MiB/s	**11** / 12 家	**29** 组
FastUDX · 11 家有效数据第 1	数据有效 · ngtcp2 已排除（见 5.4.2）	实现 × 拥塞控制 × 3 轮交错

一句话结论：在「跨境公网 + RTT ≈ 240 ms + ICMP 丢包 8.9%~24.4% + 单连接大文件下载」这一格场景里，FastUDX 与最强开源 QUIC 实现同档、且略高一线；离开这一格，请用 5.2 的方法自己测——我们不做外推。

5.3 是「一对一严格交替」的对照，回答的是"同一时刻、同一条链路上谁更快"。本节回答另一个问题：**在一种明确限定的链路状态下，FastUDX 在开源 QUIC 生态里处于什么位置。**

先划边界，再谈结论。本节的测量场景是：跨境公网；RTT ≈ 240 ms；每轮开跑前以 ICMP 15 包 @0.2 s 抽样，丢包 **8.9% ~ 24.4%**；单连接、持续下载 1 GiB 大文件；固定 30 秒窗口；两端默认收发缓冲、未调内核参数。**本节结论不适用于**局域网与低丢包内网、短连接与小报文交互、多流并发、高带宽低时延专线——那些场景需要各自单独测试，我们不做外推。

测量方法：同一台服务器灌流、同一条链路，各实现按固定顺序**交错轮转 3 轮**，12 家共 29 组实现与拥塞控制组合进入统计。为什么不是 30 组：ngtcp2 的三组数据经核验测量前提不成立，已整体排除，理由见 5.4.2。

5.4.1 参测实现

实现	主导方 / 语言	参测拥塞控制
FastUDX	蛮牛科技 / C++	自有
ngtcp2 （数据不采信，见 5.4.2）	开源社区 / C++（BoringSSL + nghttp3）	bbr / cubic / reno
xquic 1.9.8	阿里巴巴 / C	copa / bbr
picoquic 1.1.54	picoquic 社区 / C	bbr / bbr1 / prague
tquic	腾讯 / Rust	Copa / BBR / BBR3 / Cubic
msquic 2.4.8	Microsoft / C	bbr
lsquic	LiteSpeed / C	Adaptive / BBRv1 / Cubic
quinn	quinn-rs / Rust	bbr / cubic
quiche	Cloudflare / Rust	bbr / bbr2 / cubic
s2n-quic	Amazon / Rust	bbr / cubic
quic-go	quic-go / Go	cubic
quicly	Fastly / C	reno / cubic / pico / cuback

5.4.2 结果（每实现取其最优拥塞控制的 3 轮均值，11 家数据有效）

#	实现	最优 CC	MiB/s	相对榜首（越长越快）	占榜首
1	**FastUDX** 自研	native	**8.35**	7.78 ~ 8.68	100%

2	**xquic**	copa	8.14	7.96 ~ 8.35	97.5%
第二档 · 2 ~ 4 MiB/s —— 可用，但与第一档差一个身位					
3	**picoquic**	bbr1	3.68	2.36 ~ 5.49	44.1%
4	**tquic**	Copa	2.70	2.38 ~ 2.87	32.3%
5	**msquic**	bbr	2.42	1.92 ~ 2.72	29.0%
6	**lsquic**	Adaptive	2.27	0.17 ~ 4.01	27.2%
第三档 · 1 MiB/s 及以下 —— 30 秒窗口内交付量不足，实质不可用					
7	**quinn**	bbr	1.04	0.94 ~ 1.12	12.5%
8	**quiche**	bbr	0.55	0.48 ~ 0.69	6.6%
9	**s2n-quic**	bbr	0.09	0.08 ~ 0.10	1.1%
10	**quic-go**	cubic	0.021	0.017 ~ 0.024	0.25%
11	**quicly**	pico	0.019	0.017 ~ 0.021	0.23%

在**本条链路、这套参数**下，11 家数据有效的实现里 **FastUDX 以 8.35 MiB/s 居首**；xquic-Copa 8.14 与之相差 2.5%，属于同一档；第三名 picoquic-bbr1 为 3.68 MiB/s，只有第二名的 45%；第四名之后全部低于 2.8 MiB/s。**这是一个限定场景下的相对位次，不能读成"FastUDX 在所有场景都最快"。**在低丢包内网或短连接场景，各家差距会显著收窄，甚至换位。

关于被排除的 ngtcp2：ngtcp2 的三个组合都完成了测量、数值也不低，但该轮它的服务端 GSO 失效，链路丢包统计恒为 0.00%，无法证明它与其他实现跑在同一段链路状态上——**测量前提不成立，数据不可比。**按同一标准，我们对任何一家的失真数据都不采信，因此将 ngtcp2 整家排除、不参与排名（原始日志保留在结果目录，随时可复核）。ngtcp2 与 FastUDX 的对照以 5.3 为准：那一批链路统计正常、9 组严格交替，FastUDX 中位领先 2.48 倍。

批后复测：本批结束后我们对 FastUDX 做了参数收敛并复测，同一链路、同一口径再测三次，分别为 8.39 / 8.47 / 8.51 MiB/s（均值 **8.47**，该轮链路丢包 27.0%），较批内均值再高约 1.4%，说明 8.35 不是偶然峰值。

5.4.3 传统拥塞控制在弱网上整体失效

拥塞控制族	参测组合	最好成绩	最差成绩
● cubic / cuback / reno 丢包驱动型：把每次丢包都当拥塞	9 组	0.037 MiB/s（lsquic Cubic）	0.012 MiB/s
● BBR 系（bbr / bbr1 / bbr2 / BBRv1 / BBR / BBR3） 带宽建模型：同族内相差近两个数量级	11 组	3.88 MiB/s（xquic bbr）	0.061 MiB/s（tquic BBR3）
● Copa 系 延迟驱动型：本链路表现最稳	2 组	8.14 MiB/s（xquic copa）	2.70 MiB/s（tquic Copa）
● 其他（native / Adaptive / pico / prague） 自研或实验性算法	4 组	8.35 MiB/s（FastUDX）	0.019 MiB/s（quicly pico）

在 ICMP 丢包 8.9%~24.4%、RTT 约 240 ms 的跨境链路上，**丢包驱动型（cubic / cuback / reno）整体失效不是实现差异，而是算法前提不成立**：它们把每一次丢包都当拥塞信号，窗口只能反复"减半—恢复—再减半"，30 秒里最好的一组也只交付 0.037 MiB/s。同样标称 BBR 的实现之间也相差近两个数量级（0.061 ~ 3.88 MiB/s），说明在这类链路上"算法选择"与"实现质量"同样重要。这正是 FastUDX 不走"纯 ARQ + 传统拥塞控制"路线的原因——默认以自适应 FEC + ARQ 混合纠错，把可恢复的随机丢包挡在拥塞控制之外。

> 口径与可比性：本节所有数字都来自**同一批次、同一链路、交错轮转**，只能横向读，**不能与 5.3 的绝对值互比。**两批的服务器、链路与丢包水平都不同：5.3 是 RTT 约 260 ms、丢包 1.7%~6.7% 的现网 A/B；本节是 RTT 约 240 ms、ICMP 丢包 8.9%~24.4% 的极端弱网。两条结论各自成立、互不推导：在 5.3 的条件下 FastUDX 中位领先 ngtcp2 2.48 倍；在本节的条件下 FastUDX 列有效数据第 1、与 xquic-Copa 同档（8.35 对 8.14）。**差距取决于链路状态，不存在"某个实现永远快几倍"。**另：ngtcp2 在本批的链路丢包统计因服务端 GSO 失效而恒为 0，整家数据已排除，详见 5.4.2。

---

6 与主流方案的选型对比

> 下表基于各方案公开资料整理，供技术选型参考；其中“跨境弱网实测吞吐”一行来自本白皮书 5.3 的可复现实测，“开源 QUIC 生态横评”一行来自 5.4 的 12 家同场横评（11 家数据有效），变量、脚本与原始日志均已公开；两行对应的链路条件不同，不可互相推算。

维度	FastUDX	KCP	QUIC	SRT
设计定位	通用可靠 UDP 传输	轻量 ARQ 库	Web 传输（HTTP/3）	视频贡献/传输
纠错机制	自适应 FEC + ARQ	ARQ	ARQ（依赖实现）	ARQ
多路径聚合	内置 含链路故障接管实测（见 4.5）	无	部分实现	无
P2P NAT 穿透	内置 + 中继兜底	无	无	无
平台覆盖	6 平台（含嵌入式 ARM）	跨平台	跨平台	跨平台
依赖	零第三方依赖，纯 C++	轻量	依赖 TLS 生态，实现重	SDK 依赖
加密	可选内置	无内置	强制 TLS 1.3	AES
嵌入资源占用	低（可裁剪）	低	高	中
商业支持	原厂保姆式集成服务	社区	社区/云厂商	社区/商业
跨境弱网实测吞吐（vs ngtcp2-BBR，2026-10-31，见 5.3）	**5.40 MiB/s** 9/9 轮全胜，中位 2.48x	未同条件实测	2.18 MiB/s（ngtcp2 实现）	未同条件实测

开源 QUIC 生态横评（12 家参测 / 11 家有效，2026-10-27，见 5.4）	**8.35 MiB/s** 有效数据第 1；xquic 8.14 同档	未参测	2 家进入 8 MiB/s 档 第三名 3.68	未参测

选型建议：实时互动 + 弱网 + 多链路 + P2P 直连 + 嵌入式全覆盖的场景，FastUDX 是少数一站式满足的选项。

---

7 应用场景

- **音视频通话**：视频会议、在线教育、远程医疗等实时互动；
- **无人机 / 机器人远程控制**：遥控指令与遥测双向通道、第一视角图传回传、集群协同与作业调度——指令毫秒级到达，弱网下不排队、不抖动，移动切网不中断；
- **安防监控**：IPC 摄像头推流、NVR 录播、视频分析数据回传；
- **即时通讯**：IM 文件传输、语音消息、屏幕共享与协作；
- **文件传输**：大文件断点续传、批量同步、异地容灾备份；
- **金融交易**：行情推送、高频交易指令、量化数据分发；
- **游戏加速**：帧同步、状态同步、MMO 实时战斗数据传输；
- **IoT 物联网**：工业传感器数据采集、设备远程控制与 OTA；
- **直播推流**：低延迟直播、云游戏串流、VR/AR 内容传输。

---

8 集成与生态

8.1 接口体系

接口	头文件	适用场景
C++ 面向对象接口	FastUdx.h	`CreateFastUdx()` 工厂函数 + `IFastUdx` 实例接口 + `IUdxTcp` 连接接口 + sink 回调，适合中大型 C++ 工程
纯 C API	udxapi.h	无 C++ 依赖，可直接嵌入 C 工程或通过 FFI 跨语言绑定
P2P 平台 SDK	DevAppSDK.h	设备接入、远程连接、P2P/Relay 服务

8.2 最小可用示例（真实接口）

下面两段代码取自随 SDK 发布的 `SimpleServer` / `SimpleClient` 示例工程，删去 UI 外壳后可直接编译运行。它体现两个设计取向：**所有事件经 sink 回调交给应用层，协议内部不建业务线程；发送由 ACK 驱动的 `OnStreamNeedMoreData` 续投，应用既不用轮询也不必额外开线程。**

服务端：实现三个回调即可对外服务

server.cpp · 三个回调，一个 Start()，对外服务就绪 C++

```
#include "FastUdx.h"

class CMyServer : public IUdxTcpSink
{
public:
    void OnStreamConnect(IUdxTcp* pTcp, int erro) override
    {
        if (erro) return;                          // 0 = 成功，3 = 超时
        pTcp->SetUserData((UINT64)NewSession()); // 会话挂到这条流上
    }

    void OnStreamRead(IUdxTcp* pTcp, BYTE* pData, int len) override
    {
        // 回调在协议工作线程上执行；pData 仅本次回调内有效，要留存须自行拷贝
        GetSession(pTcp)->OnRecv(pData, len);
    }

    void OnStreamBroken(IUdxTcp* pTcp) override { ReleaseSession(pTcp); }

    BOOL Start(const char* ip, unsigned short port)
    {
        m_pUdx = CreateFastUdx();
        m_pUdx->SetSink(this);                     // 事件出口
        m_pUdx->SetServerBlockSize(10);            // 内部工作线程档位
        return m_pUdx->Create((char*)ip, port);  // 绑定并开始监听
    }
    void Stop() { if (m_pUdx) { m_pUdx->Destroy(); m_pUdx = NULL; } }

private:
    IFastUdx* m_pUdx;
};
```

客户端：ACK 驱动的背压式发送

client.cpp · 一块首投 + 回调续投，不需要业务发送线程 C++

```cpp
IFastUdx* udx = CreateFastUdx();
udx->SetSink(&sink);
udx->Create(NULL, 0);                          // 本地任意端口

// bAutoRelase 传 FALSE：库内 AddLife()，由调用方 Destroy() 归还引用。
// 用默认 TRUE 拿到的是借用指针，握手失败即被库回收，再轮询就是 use-after-free。
IUdxTcp* conn = udx->Connect(host, port, FALSE, 0, 0);

// 首块投递：返回 TRUE 只表示"已拷入 UDX 缓冲"，不代表已送达
if (!conn->SendBuff(buf, n)) { /* 缓冲已满，等下面这个回调 */ }

// 收到 ACK 后由协议回调，按剩余窗口续投 —— 不需要业务发送线程
void OnStreamNeedMoreData(IUdxTcp* pTcp, int needdata) override
{
    while (needdata > 0 && HasMore()) {
        int n = NextChunk(buf, sizeof(buf));
        if (!pTcp->SendBuff(buf, n)) break;
        needdata -= n;
    }
}
```

按场景可再叠加：音视频保帧用 `SendBuffFrame(buf, len, TRUE)`（底层满也强制入队，避免拆帧）；接收侧限流用 `SetFlowControlWindow(bytes)`；多网卡聚合用 `AddMultPath(subTcp)`；需要更低重传时延可 `EnableFec(TRUE)`。全部是运行时开关，不必改协议栈。

8.3 平台矩阵

平台	支持版本	交付形式
Windows	Win 7 / 10 / 11 / Server	SDK + Demo (RAR)
Linux	CentOS 6.5+ / Ubuntu 12.04+（glibc 2.14+）x86_64	静态库 + 动态库 + 命令行 Demo + 头文件
Android	5.0+ (API 21) arm64-v8a	SDK 包 + 命令行 Demo + APK Demo
macOS	10.15 (Catalina)+ Intel x86_64	静态库 + 动态库 + 命令行 Demo + 头文件

iOS	12.0+ (arm64)	即将推出
嵌入式 ARM	HiSilicon / Rockchip / NXP 等	交叉编译工具链支持

8.4 下载与校验

SDK 以**完整包**与**单文件**两种形式提供，全部位于站点根目录，可直接 HTTP GET 获取、无需注册。下表即为最终下载地址，可点击，也可照抄到浏览器地址栏或命令行。

下载地址	平台 / 架构	大小	内容
http://www.goodudx.com/fastudx-sdk-linux-x64-1.2.tar.gz	Linux x86_64 glibc 2.14+	1.71 MB	fudx 命令行 + libFastUdx.a/.so + 4 个头文件
http://www.goodudx.com/fastudx-sdk-macos-x64-1.2.tar.gz	macOS 10.15+ Intel x86_64	1.30 MB	fudx 命令行 + libFastUdx.a/.dylib + 4 个头文件
http://www.goodudx.com/fastudx-sdk-android-arm64-1.2.tar.gz	Android 5.0+ arm64-v8a	3.76 MB	fudx 命令行 + libFastUdx.a/.so + 4 个头文件
http://www.goodudx.com/fastudx-sdk-headers-1.2.tar.gz	全平台	30.5 KB	FastUdx.h / udxapi.h / udxos.h / mntypedef.h
http://www.goodudx.com/fastudx-fudx-src-1.2.tar.gz	全平台源码 fudx 测试程序	53.3 KB	fudx 源码 22 个文件 + 4 个头文件 + 独立 Makefile + VC6/VS 工程（库文件需另取）

各平台包内含 `README.txt` 与 `SHA256SUMS`；fudx 源码包内含构建说明与头文件，库文件请从平台包或单文件直链获取。解包后逐文件校验（以 Linux 包为例）：

```
curl -O http://www.goodudx.com/fastudx-sdk-linux-x64-1.2.tar.gz
tar xzf fastudx-sdk-linux-x64-1.2.tar.gz
cd fastudx-sdk-linux-x64-1.2 && sha256sum -c SHA256SUMS
```

单文件直链

只需单个可执行文件或库时可直接下载，与整包同一根路径、无需解包：

下载地址	用途 / 大小
http://www.goodudx.com/fudx-linux-x64	Linux 命令行测试程序 · 949.3 KB
http://www.goodudx.com/fudx-macos-x64	macOS 命令行测试程序 · 711.8 KB
http://www.goodudx.com/fudx-android-arm64	Android 命令行测试程序（adb push）· 1.44 MB
http://www.goodudx.com/libFastUdx-linux-x64.a	Linux 静态库 · 3.00 MB
http://www.goodudx.com/libFastUdx-linux-x64.so	Linux 动态库 · 1.11 MB
http://www.goodudx.com/libFastUdx-macos-x64.a	macOS 静态库 · 1.82 MB
http://www.goodudx.com/libFastUdx-macos-x64.dylib	macOS 动态库 · 777.1 KB
http://www.goodudx.com/libFastUdx-android-arm64.a	Android 静态库 · 3.64 MB
http://www.goodudx.com/libFastUdx-android-arm64.so	Android 动态库 · 8.32 MB
http://www.goodudx.com/FastUdx.h	主 API 头文件 · 60.8 KB
http://www.goodudx.com/udxapi.h	C 风格 API 头文件（自包含）· 12.2 KB
http://www.goodudx.com/udxos.h	平台适配头文件 · 4.3 KB
http://www.goodudx.com/mntypedef.h	基础类型定义头文件 · 2.5 KB
http://www.goodudx.com/fastudx-fudx-src-1.2.tar.gz	fudx 测试程序源码包 · 53.3 KB
http://www.goodudx.com/fastudx-downloads.json	机器可读清单（含 SHA256 / MD5 / 字节数）· 10.2 KB

http://www.goodudx.com/fastudx-desktop-echo-demo.mp4	桌面 Echo 实机演示录像（下载后本地播放）· MP4 14.2 MB / 1 分 15 秒

四个头文件需放在同一目录：`FastUdx.h` 内部引用 `udxos.h` 与 `mntypedef.h`，`udxapi.h` 自包含。

演示视频：桌面 Echo 实机运行

不想先写代码的，直接看录像：`fudx -s` 与 `fudx -c` 两端各起一个实例并约定同名 Echo 服务，桌面程序把收到的报文原样回送、逐包打印往返时延，全程实机录屏（1 分 15 秒）。**视频只提供下载地址，页面不内嵌播放器**——十几 MB 的媒体资源内嵌会拖慢首屏并额外占用带宽，需要时下载到本地播放即可，下载地址即上表最后一行。

试用与授权

站点提供的 `libFastUdx` 库（`.a` / `.so` / `.dylib`）可直接用于功能验证与联调：功能完整、无需授权码、不依赖 license 文件。商用授权、长期版本与定制交付请联系官网 goodudx.com。

---

9 商业与支持

- **零门槛试用**：官网下载测试程序免费体验，功能验证满意后再洽谈合作；
- **保姆式集成服务**：从评估、集成到上线的全程技术支持，协助环境搭建与联调测试；
- **深度定制**：代码清晰规范，支持定制化开发与深度集成；
- **私有化部署**：P2P 接入平台支持私有化部署与独立运营；
- **100% 自主可控**：从内核到 API 完全自主研发，无第三方闭源依赖，支持国产化要求。

联系方式：wwwllg@126.com ｜ QQ: 24508609 ｜ QQ群: 941132084

作者技术专栏：UDX 协议 · CSDN 博客 —— 协议设计与工程实践手记，持续更新

---

10 常见问题（FAQ）

Q: 什么是 FastUDX?

FastUDX（UDX3.0）是武汉蛮牛科技自主研发的可靠 UDP 传输协议与 SDK，纯 C++ 零依赖，提供低延迟、抗丢包的通用数据传输能力，支持六大平台。

Q: FastUDX 与 KCP 有什么区别?

KCP 是轻量 ARQ 重传库；FastUDX 在其之外内置了自适应 FEC、多路径聚合、P2P NAT 穿透、接收窗口流控、无锁数据面与全平台交付，是完整的端到端传输解决方案。

Q: 如何在 30% 丢包的网络上保持流畅?

FastUDX 采用自适应 FEC + ARQ 混合纠错：大多数丢包由 FEC 在一个 RTT 内直接恢复，无需重传等待；配合平稳发送与流控，30% 丢包下仍保持流畅。

Q: FastUDX 支持哪些平台?

Windows、Linux、macOS、Android、iOS、嵌入式 ARM 六大平台，凡支持 C++ 的环境均可运行。

Q: 是否支持国产化与嵌入式?

是。100% 自主代码，零第三方依赖，已在 HiSilicon、Rockchip、NXP 等嵌入式平台运行，可裁剪适配资源受限环境。

Q: “无锁内核”是什么意思?

FastUDX 数据面使用自研无锁链表、无锁 Map、无锁内存池与 EBR 内存回收，热点路径无锁竞争，单核跑满千兆，多核近线性扩展。

Q: NAT 穿透成功率如何保证?

内置 Cone/Symmetric NAT 穿透；穿透失败自动降级 Relay 中继转发，确保 100% 连通率。

Q: 如何集成到我的项目?

获取 FastUdx.h（C++）或 udxapi.h（C）头文件，三步调用即可跑通（见第 8 章）；原厂提供全程集成支持。

Q: 授权与试用方式?

先免费下载 Demo 体验，功能验证满意后再洽谈合作，全程零风险。

Q: FastUDX 适合游戏加速吗?

适合。帧同步/状态同步场景受益于 <1ms 局域网延迟、链路故障时业务不中断（在途数据在判死瞬间转由其余链路承载）与 30% 抗丢包能力，已有游戏加速场景落地。

Q: 你们的性能数字经得起验证吗？敢和 QUIC 正面比吗?

经得起，但请连着场景一起读——两项都是「高时延 + 丢包 + 单连接大文件下载」的跨境公网测试：① 2026-10-31，RTT 约 260 ms、丢包 1.7%~6.7%，与 ngtcp2（BBR）9 组严格交替，FastUDX 全胜，中位 5.40 MiB/s 对 2.18 MiB/s（2.48 倍，见 5.3）；② 2026-10-27，RTT 约 240 ms、ICMP 丢包 8.9%~24.4%，12 家参测、11 家数据有效，FastUDX 8.35 MiB/s 列第 1，xquic-Copa 8.14 同档，第三名 3.68 MiB/s（见 5.4）。局域网、低丢包内网、短连接场景不在此列。方法、判定口径、排除依据与逐轮原始数据全部公开，欢迎按同样口径复核。

Q: 在 12 家 QUIC 实现横评里，FastUDX 排第几?

先把场景说清楚：2026-10-27 的同场横评是跨境公网、RTT 约 240 ms、ICMP 丢包 8.9%~24.4%、单连接 1 GiB 下载、30 秒固定窗口。12 家参测，其中 ngtcp2 因当轮服务端 GSO 失效、链路丢包统计失真，测量前提不成立而被排除，**11 家数据有效**：**FastUDX 8.35 MiB/s 列第 1**，xquic-Copa 8.14 相差 2.5%、属同档，第三名 picoquic-bbr1 为 3.68 MiB/s，第四名之后均低于 2.8 MiB/s，cubic / reno 系全部跌到 0.04 MiB/s 以下。该排名只对应该链路条件，不适用于局域网或低丢包场景。详见 5.4。

Q: 如何引用或复核这些性能数据?

5.3 与 5.4 给出了链路条件、被测版本、参数、判定口径与逐轮数据；机器可读数据集见 `/quic-benchmark-2026-10.json`，脚本与原始日志可向 wwwllg@126.com 索取。引用时请注明「FastUDX 技术白皮书 v1.2」与测试日期。

Q: 多路径能把带宽翻倍吗?

不能简单翻倍，聚合收益取决于链路组合。多路径的第一价值是链路冗余与故障接管：一条链路中断时数据流不中断，在途数据在判死瞬间转由其余链路继续发送；实测一条子路永久中断后文件仍完整落地、耗时仅比基线多 1 秒，全部路径同时中断 45.5 秒也能完整传完（见 4.5）。具体增益由各链路可用带宽、统一总预算与调度开销共同决定，需在目标链路上实测（方法见 5.2）。

Q: 如何向 FastUDX 发起对比挑战?

同条件、同口径、可复现即可：固定窗口、单连接、只统计应用层交付字节 ÷ 耗时、两种协议交替各跑 3 轮以上取中位。把协议实现、参数、脚本与原始日志发到 wwwllg@126.com，我们复核后公开结果，输赢都公开。

---

11 附录

> **附录 A：术语表**

术语	说明
ARQ	Automatic Repeat reQuest，自动重传请求
FEC	Forward Error Correction，前向纠错
EBR	Epoch-Based Reclamation，基于世代的内存安全回收算法
NAT 穿透	让内网设备之间建立直连通道的技术
Cone / Symmetric NAT	两类主流 NAT 行为模型
队头阻塞	前置丢包阻塞后续数据交付的现象
RTO	Retransmission Timeout，重传超时
ngtcp2	开源 QUIC 协议实现（C 语言，常配合 BoringSSL 使用），本白皮书 5.3 的对照对象
BBR	基于带宽与时延建模的拥塞控制算法，由 Google 提出
Copa	以排队时延为主要信号的低时延拥塞控制算法（MIT），xquic 与 tquic 均已实现
同场横评	多家实现在同一条链路、同一时段内按固定顺序交错轮转测试，以消除链路时变对结论的影响
MiB/s 与 MB/s	1 MiB = 1048576 字节，1 MB = 1000000 字节；本白皮书速率统一用 MiB/s，比 MB/s 读数约小 4.9%

附录 B：文档版本

版本	日期	说明
v1.0	2026-10-03	首版
v1.1	2026-10-31	新增 5.3 第三方对照实测（跨境弱网 vs ngtcp2-BBR，9 组）与挑战邀请；第 5.1 与第 6 章增列实测行；附录补充 ngtcp2 / BBR / MiB 术语

v1.2	2026-10-31	新增 5.4「十二家 QUIC 实现同场横评：高时延高丢包跨境链路下的相对位置」（12 家参测、29 组有效组合、3 轮交错；ngtcp2 因服务端 GSO 失效、链路丢包统计失真，数据不采信并排除）；8.2 集成示例改写为真实接口（sink 回调 + ACK 驱动背压发送）；摘要、第 6 章、FAQ、附录同步更新
v1.2 修订	2026-11-03	4.5 多路径由要点式重写为「机制 + 链路故障注入实测 + 能力边界」：统一预算与三段写入闸门、判死与在途数据同毫秒接管、主控失效不影响会话、路径自动绑回；补 5 组实测（基线 + 4 组故障注入，以落盘 MD5 为判据）；授权说明改为通用试用条款；多路径表述改为「聚合收益取决于链路组合，增益需在目标链路实测」；摘要、第 6 章选型表、FAQ 与结构化数据同步
v1.2 修订	2026-11-04	8.4 新增桌面 Echo 实机演示录像的下载地址与说明（MP4，1 分 15 秒；只给链接、页面不内嵌播放器）；页面目录改为吸顶，章节锚点增加定位留白，PDF 版补章节书签，正文可直接跳章

武汉蛮牛科技有限公司 | FastUDX（UDX3.0）

Email: wwwllg@126.com | QQ: 24508609 | QQ群: 941132084 | 作者博客（CSDN）

www.goodudx.com | © 2026 武汉蛮牛科技有限公司，本文档可自由转载，转载请注明出处