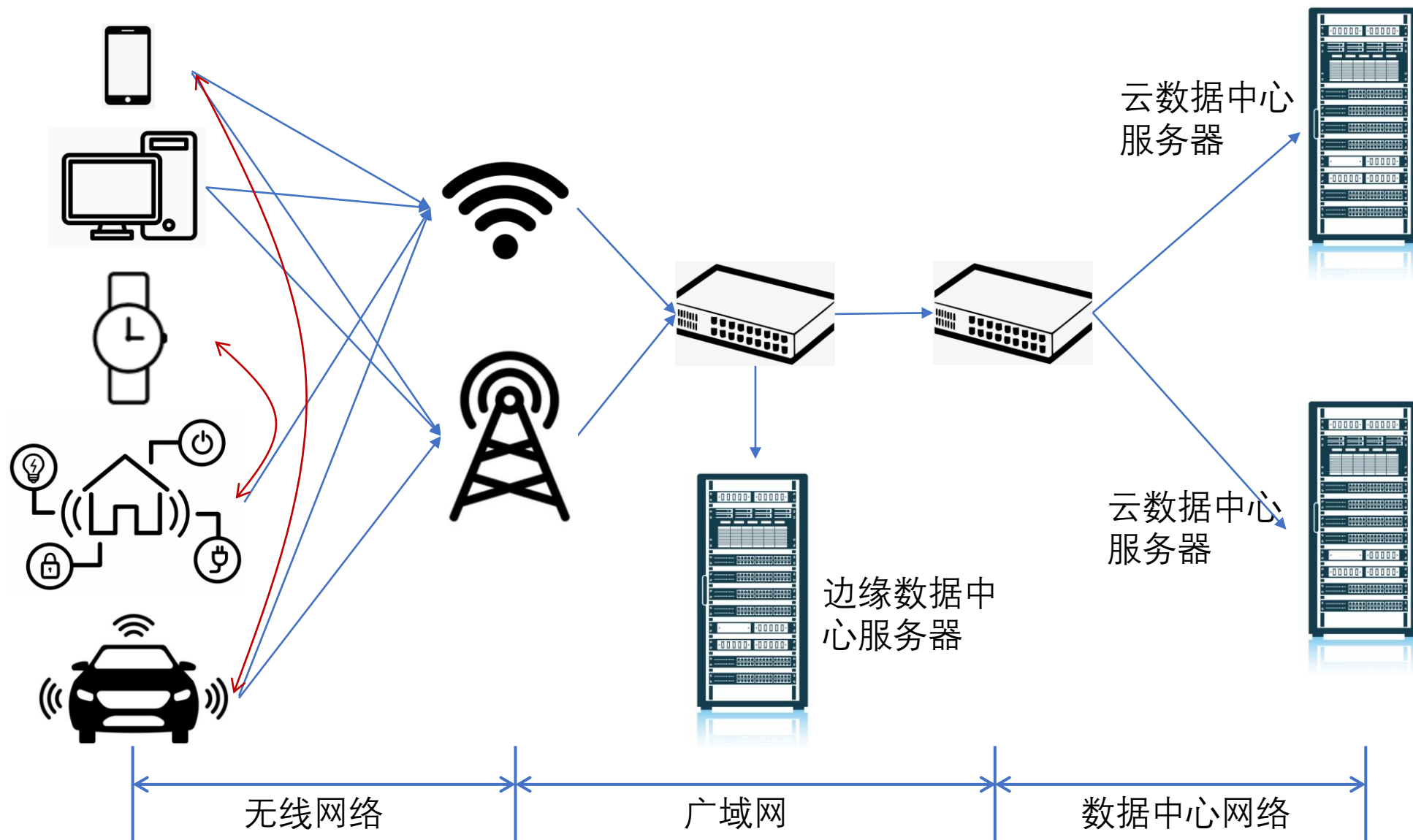


# 计算机网络的新黄金时代

华为计算机网络与协议实验室 李博杰

2022年12月12日

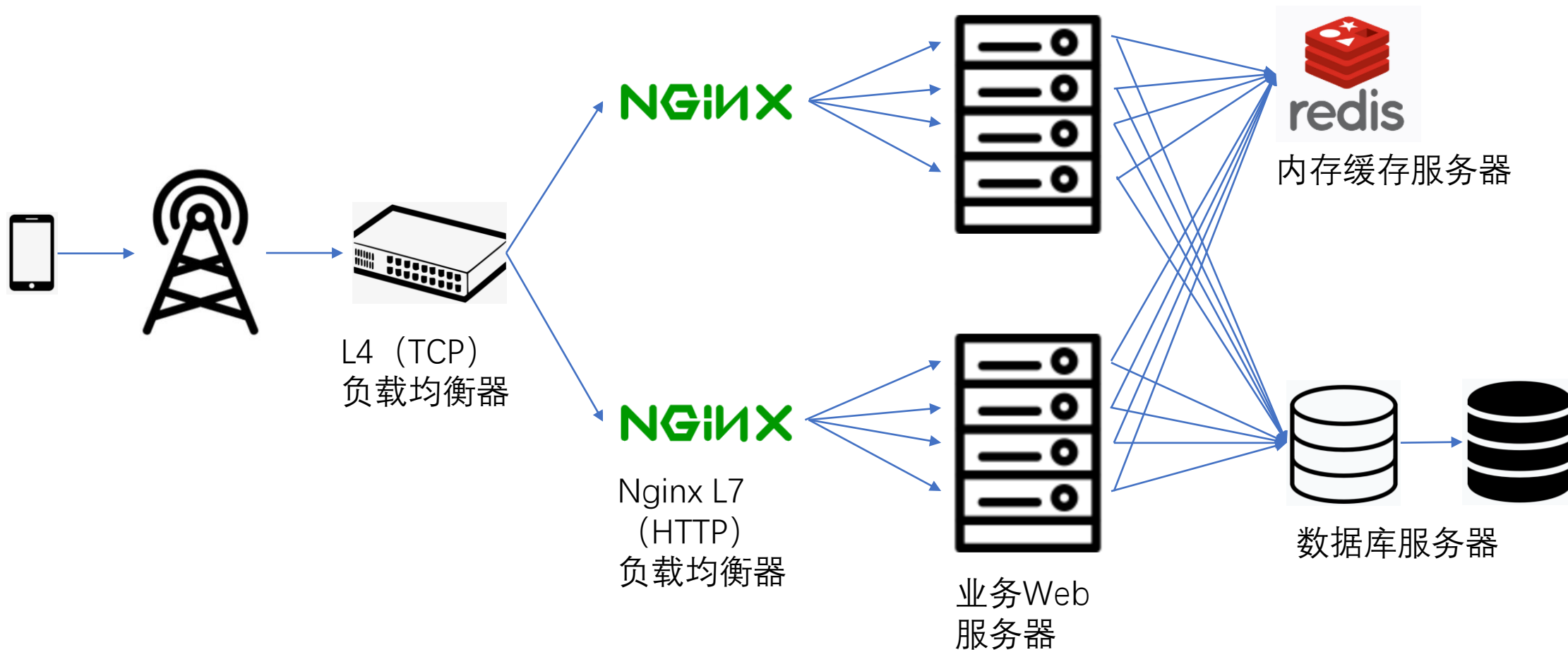
# 计算机网络的架构



# 目录

- **数据中心网络**
  - 从Web服务到大规模异构并行计算
  - 编程抽象：从字节流到内存语义
  - 把数据中心作为一台计算机
- **广域网**
  - 从“尽力而为”到“准确性”
  - 东数西算与全国一体化大数据中心
- **无线网络**
  - “1+8+N”与“5G to B”
  - 应用和数据无缝流转的“分布式超级终端”

# 传统数据中心网络：为容易并行的Web服务设计





# 传统Web服务对数据中心网络的时延不太敏感

The screenshot displays a web application interface for a course review community. The top navigation bar includes 'USTC评课社区', '点评', '课程', '关注', '导师', a search box for '搜索课程、老师', a notification bell with '37' messages, and a user profile 'boj'. The main content area is titled '课程列表' and features a filter section with '课程类别' (全部, 通识, 体育, 英语, 慕课, 双学位, 研究生课) and '排序方式' (课程评分, 点评数量). Below this, it shows '共 14288 门课 (当前第 1 页)'. Three course entries are listed: '几何学基础 (盛茂) 2021秋' with a 10.0 rating, '拓扑学(H) (王作勤) 2022春...' with a 10.0 rating, and '计算机程序设计A (白雪飞) 2022秋...' with a 9.9 rating. A dark-themed performance profiler overlay is visible on the right side, showing various metrics: 'Versions' (FLASK 1.1.4), 'Time' (CPU: 134.92ms (1082.73ms)), 'HTTP Headers', 'Request Vars', 'Config', 'Templates' (1 RENDERED), 'SQLAlchemy' (18 QUERIES), 'Logging' (0 MESSAGES), 'Route List' (99 ROUTES), and 'Profiler' (VIEW: 1081.68ms) with a green checkmark.

打开网页的时延在秒级以内，用户基本无感知，广域网时延在数十毫秒级，数据中心内网时延在毫秒级，主要时延消耗在业务上（业务处理、数据库查询）

# SQLAlchemy queries

(ms)	Action	Context	Query
79.5329	SELECT EXPLAIN	./views/course.py:66 (index)	<pre>SELECT anon_1.courses_id AS anon_1_courses_id, anon_1.courses_name AS anon_1_courses_name, anon_1.courses_dept_id AS anon_1_courses_dept_id FROM (SELECT courses.id AS courses_id, courses.name AS courses_name, courses.dept_id AS courses_dept_id, courses.introduction AS courses_ir FROM reviews) * (SELECT count(reviews.id) / count(distinct(reviews.course_id)) AS anon_2 FROM reviews)) / (course_rates.review_count + (SELECT count(reviews.id) / count(distinct(reviews.course_id)) AS anon_2 FROM reviews)), %s) AS `IF_1` FROM courses INNER JOIN course_rates ON courses.id = course_rates.id ORDER BY IF(course_rates.review_count &gt; %s, (course_rates._rate_total FROM reviews) * (SELECT count(reviews.id) / count(distinct(reviews.course_id)) AS anon_2 FROM reviews)) / (course_rates.review_count + (SELECT count(reviews.id) / count(distinct(reviews.course_id)) AS anon_2 FROM reviews)), %s) DESC LIMIT %s, %s) AS anon_1 LEFT OUTER JOIN depts AS depts_1 ON depts_1.id = anon_1.courses_dept_id LEFT OUTER JOIN (course_teachers AS course</pre>
19.3055		./views/course.py:66 (index)	<pre>SELECT count(*) AS count_1 FROM (SELECT courses.id AS courses_id, courses.name AS courses_name, courses.dept_id AS courses_dept_id, courses.introduction AS courses_ir FROM courses INNER JOIN course_rates ON courses.id = course_rates.id) AS anon_1</pre>
0.3834	SELECT EXPLAIN	./models/user.py:349 (load_user)	<pre>SELECT users.id AS users_id, users.username AS users_username, users.email AS users_email, users.password AS users_password, users.active AS FROM users WHERE users.id = %s</pre>
0.2325	SELECT EXPLAIN	./__init__.py:58 (inject_global_banner)	<pre>SELECT banner.id AS banner_id, banner.desktop AS banner_desktop, banner.mobile AS banner_mobile, banner.publish_time AS banner_publish_time FROM banner ORDER BY banner.publish_time DESC LIMIT %s</pre>
720.5443	SELECT EXPLAIN	./models/user.py:118 (latest_notifications_text)	<pre>SELECT notifications.id AS notifications_id, notifications.to_user_id AS notifications_to_user_id, notifications.from_user_id AS notificati FROM notifications WHERE %s = notifications.to_user_id ORDER BY notifications.time DESC</pre>
11.8165	SELECT EXPLAIN	./models/user.py:202 (info)	<pre>SELECT students.sno AS students_sno, students.name AS students_name, students.email AS students_email, students.dept_id AS students_dept_id FROM students WHERE %s = students.user_id</pre>
69.1741	SELECT EXPLAIN	./models/user.py:164 (courses_joined)	<pre>SELECT course_classes.id AS course_classes_id, course_classes.course_id AS course_classes_course_id, course_classes.term AS course_classes_ FROM course_classes, join_course WHERE %s = join_course.student_id AND course_classes.id = join_course.class_id ORDER BY course_classes.term DESC</pre>
60.9488	SELECT EXPLAIN	./models/course.py:362 (reviewed_by)	<pre>SELECT courses.id AS courses_id, courses.name AS courses_name, courses.dept_id AS courses_dept_id, courses.introduction AS courses_introduc FROM review_course, courses LEFT OUTER JOIN depts AS depts_1 ON depts_1.id = courses.dept_id LEFT OUTER JOIN (course_teachers AS course_tea WHERE %s = review_course.user_id AND courses.id = review_course.course_id ORDER BY teachers_1.id</pre>
0.6387	SELECT EXPLAIN	./models/course.py:569 (term_ids)	<pre>SELECT course_terms.id AS course_terms_id, course_terms.course_id AS course_terms_course_id, course_terms.term AS course_terms_term, course FROM course_terms WHERE %s = course_terms.course_id ORDER BY course_terms.term DESC</pre>
0.5608	SELECT EXPLAIN	./models/course.py:569 (term_ids)	<pre>SELECT course_terms.id AS course_terms_id, course_terms.course_id AS course_terms_course_id, course_terms.term AS course_terms_term, course FROM course_terms WHERE %s = course_terms.course_id ORDER BY course_terms.term DESC</pre>

Hide »

Versions

FLASK 1.1.4

Time

CPU: 134.92ms (1082.73ms)

HTTP Headers

Request Vars

Config

Templates

1 RENDERED

SQLAlchemy

18 QUERIES

Logging

0 MESSAGES

Route List

99 ROUTES

Profiler

VIEW: 1081.68ms

Calls	Total Time (ms)	Per Call (ms)	Cumulative Time (ms)	Per Call (ms)	Function
18	948.223277	52.6791	948.223277	52.6791	{function Connection.query at 0x7f2599b60a60}
2149	49.698313	0.0231	49.698313	0.0231	<sqlalchemy/orm/state.py:96(__init__)>
18	8.936239	0.4965	8.936239	0.4965	{method 'store_result' of '_mysql.connection' objects}
2340/1926	6.504395000000001	0.0028	68.982538	0.0358	<sqlalchemy/orm/loading.py:883(_instance)>
3479	5.332349	0.0015	5.720546	0.0016	<MySQLdb/times.py:66(DateTime_or_None)>
2295/1926	5.1335809999999995	0.0022	56.452625000000005	0.0293	<sqlalchemy/orm/loading.py:1108(_populate_full)>
18	3.2205660000000003	0.1789	8.941112	0.4967	{method 'fetch_row' of '_mysql.result' objects}
2149	2.055756	0.0010	52.710592	0.0245	<sqlalchemy/orm/instrumentation.py:412(new_instance)>
154	1.374729	0.0089	1.9258850000000003	0.0125	<sqlalchemy/sql/coercions.py:112(expect)>
168/18	1.1764080000000001	0.0070	2.889365	0.1605	<sqlalchemy/sql/traversals.py:159(_gen_cache_key)>
18	0.7589250000000001	0.0422	0.918012	0.0510	<sqlalchemy/engine/result.py:392(<listcomp>)>
2149	0.712991	0.0003	0.712991	0.0003	<sqlalchemy/orm/identity.py:163(_add_unpresent)>
2163	0.703816	0.0003	0.703816	0.0003	{method 'issuperset' of 'frozenset' objects}
17	0.693119	0.0408	69.675657	4.0986	<sqlalchemy/orm/loading.py:151(<listcomp>)>
1932	0.673772	0.0003	0.96818	0.0005	<sqlalchemy/orm/collections.py:1168(append)>
11	0.633395	0.0576	0.6384770000000001	0.0580	{built-in method builtins.exec}
23/17	0.625194	0.0272	2.233086	0.1314	<sqlalchemy/orm/loading.py:648(_instance_processor)>
1649/875	0.622604	0.0004	967.7866240000001	1.1060	{built-in method builtins.getattr}
2149	0.60466	0.0003	0.60466	0.0003	<string>:1(set)
2341	0.600523	0.0003	0.600523	0.0003	<sqlalchemy/orm/identity.py:168(get)>
2241	0.548297	0.0002	0.6094849999999999	0.0003	{built-in method builtins.isinstance}
2558	0.48285500000000003	0.0002	0.48285500000000003	0.0002	{built-in method __new__ of type object at 0x906da0}
6	0.475317	0.0792	1.025746	0.1710	<flask_sqlalchemy/_init__.py:413(iter_pages)>
1427	0.46500800000000003	0.0003	0.5514610000000001	0.0004	<flask_sqlalchemy/_init__.py:368(pages)>
7248/7245	0.44383500000000004	0.0001	0.44510000000000005	0.0001	{built-in method builtins.len}
769	0.428827	0.0006	953.395804	1.2398	<sqlalchemy/orm/attributes.py:466(__get__)>
130/101	0.411355	0.0032	2.2168970000000003	0.0219	<sqlalchemy/util/langhelpers.py:1216(__getattr__)>
215/173	0.404194000000000005	0.0019	1.7020400000000002	0.0098	<sqlalchemy/util/langhelpers.py:1177(__get__)>
18	0.400897000000000006	0.0223	970.986831	53.9437	<sqlalchemy/engine/base.py:1677(_execute_context)>
202/169	0.381079000000000006	0.0019	0.878789	0.0052	<sqlalchemy/orm/path_registry.py:510(__getitem__)>
4	0.355255	0.0888	949.671755	237.4179	<sqlalchemy/orm/strategies.py:934(_emit_lazyload)>
87	0.350777	0.0040	2.844432	0.0327	<sqlalchemy/sql/compiler.py:2976(_label_select_column)>
435	0.338601	0.0008	227.457977	0.5229	./templates/course-list.html:5(root)
5	0.33451	0.0669	0.5228520000000001	0.1046	<sqlalchemy/engine/result.py:409(<listcomp>)>
811	0.329048	0.0004	0.338312	0.0004	{method 'get' of 'dict' objects}
localhost:12345/course/		0.0037	1.5653730000000001	0.0180	<sqlalchemy/sql/compiler.py:1441(visit_label)>

## Versions

FLASK 1.1.4

## Time

CPU: 134.92ms (1082.73ms)

## HTTP Headers

## Request Vars

## Config

## Templates

1 RENDERED

## SQLAlchemy

18 QUERIES

## Logging

0 MESSAGES

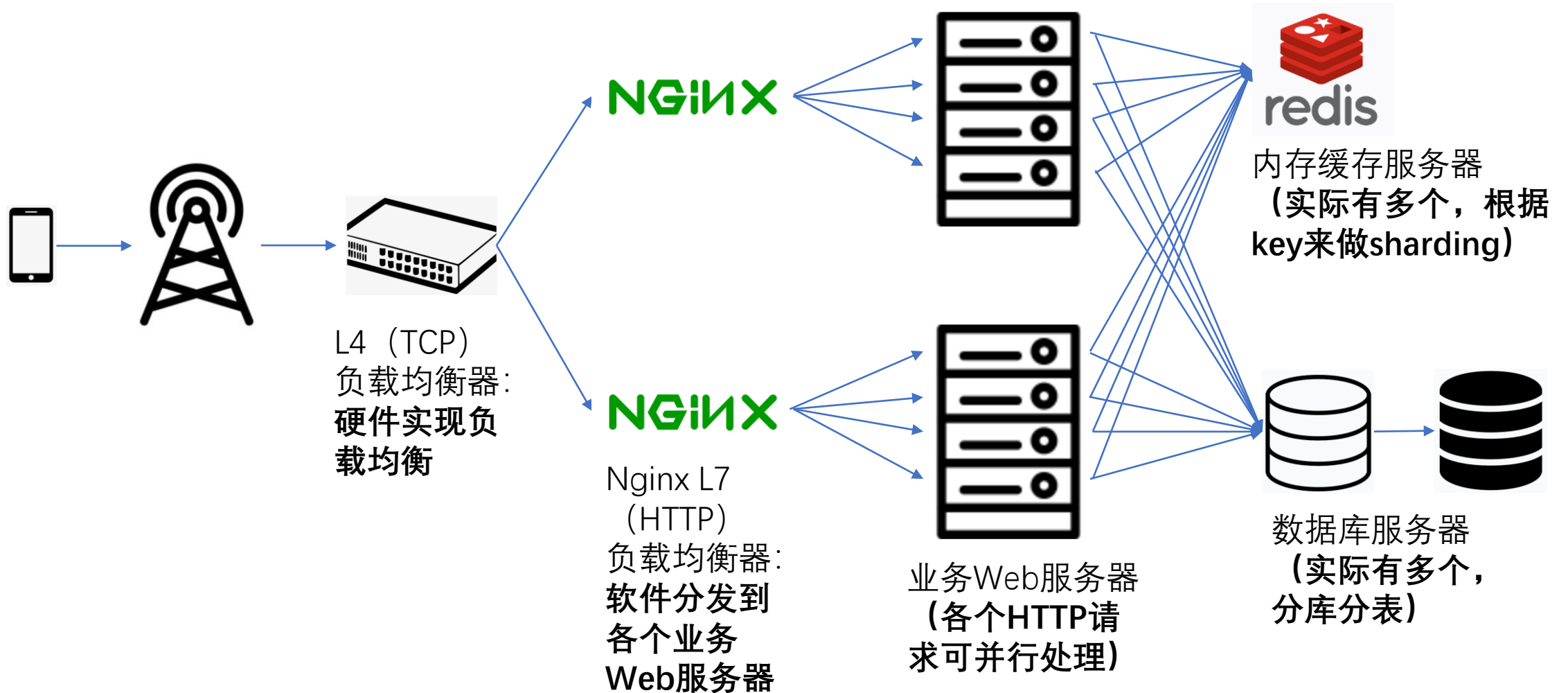
## Route List

99 ROUTES

## Profiler

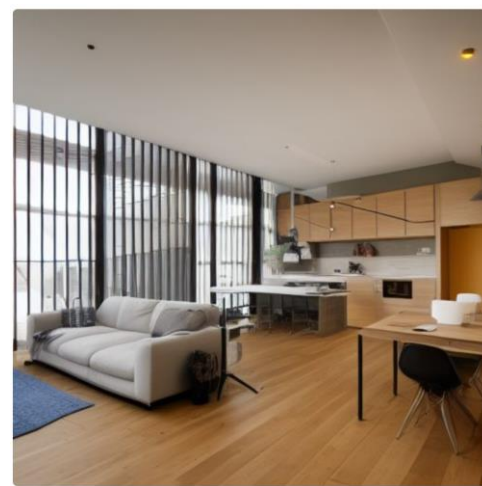
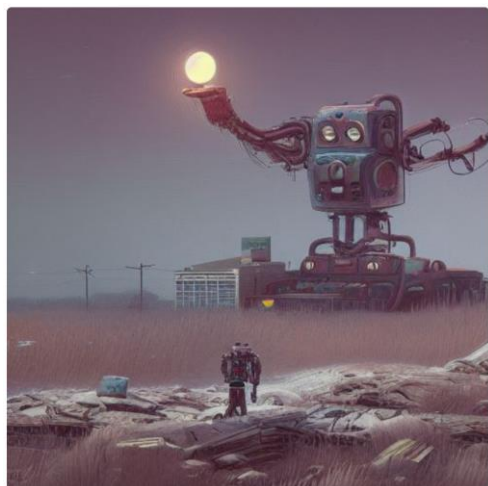
VIEW: 1081.68ms

# 传统Web服务容易扩容



# 数据中心网络需求的新变化

- AI、大数据、HPC对网络的带宽和时延提出了越来越高的诉求



CV生成模型：**Stable Diffusion**，2022，以上图片为根据提示词生成  
训练使用**256个NVIDIA A100 GPU**，150,000 GPU小时，耗资60万美元



# 数据中心网络需求的新变化

LA

```
for (int i=0; i<10; i++)
  for (int j=0; j<10; i++)
    if (a[i] < a[j]) {
      int tmp = a[i]; a[i] = a[j]; a[j] = tmp;
    }
```

There is a bug in the inner `for` loop. The loop control variable `j` is not being incremented, so it will never change value and the loop will run indefinitely. This can be fixed by changing the loop control statement to `for (int j=0; j<10; j++)`.

Additionally, the code as written does not properly swap the elements of the array `a` when `a[i]` is less than `a[j]`. Instead of swapping `a[i]` and `a[j]`, it is swapping the values of the temporary variables `tmp` and `a[j]`. To properly swap the elements of the array, the code should be changed to the following:

## ChatGPT, 2022, AI问答系统

模型基于GPT-3.5和InstructGPT, GPT-3就已经有1750亿个参数, 需要350 GB GPU内存分布式推理/训练, 单次训练成本1200万美金, ChatGPT只会更大

LA 能不能帮我在Protocol Buffers里面创建一个地址簿, 地址簿里面有若干联系人, 每个联系人有姓名、地址、电话等。

可以的, 可以使用Protocol Buffers定义一个地址簿, 在地址簿中定义若干联系人, 每个联系人包含姓名、地址、电话等信息。

定义示例如下:

```
// 地址簿
message AddressBook {
  // 联系人列表
  repeated Person person = 1;
}

// 联系人
message Person {
  // 姓名
  string name = 1;
  // 地址
  string address = 2;
  // 电话
  string phone = 3;
}
```

使用时, 可以创建一个AddressBook对象, 并向其中添加联系人。例如:

```
AddressBook address_book;

// 添加联系人
address_book.add_person()->set_name("John");
```

LA 能不能帮我在地址簿中增加一个联系人数量的字段

可以的, 可以在地址簿中增加一个联系人数量的字段。

定义示例如下:

```
// 地址簿
message AddressBook {
  // 联系人数量
  uint32 count = 1;
  // 联系人列表
  repeated Person person = 2;
}
```

使用时, 可以通过设置count字段来指定联系人数量。例如:

```
AddressBook address_book;

// 设置联系人数量
address_book.set_count(3);

// 添加联系人
address_book.add_person()->set_name("John");
address_book.add_person()->set_address("New York");
address_book.add_person()->set_phone("123-456-7890");
```

在序列化或反序列化时, count字段也会被保存或恢复。

LA

2100年，一种传染性很强的病毒席卷世界。一开始，这种病毒的致死率很高，各国政府都采取了严格的防疫措施。但是，数年之后，病毒的毒性显著降低，因此几乎所有国家都放松了防疫，除了一个与世隔绝的封闭国家。

一年后，全世界的大多数人都感染了这种病毒。这时，这种病毒突然产生了一个突变，致死率显著提高，全世界在几个月内尸横遍野。

你作为全世界唯一未受感染国家的首席生物学家，你的使命是集结人类残存的科研力量，消灭病毒。

请以此为背景，写一部小说。现在开始写前言部分。



在2100年，一种叫做“XX”的病毒席卷了世界。这种病毒最初具有高致死率，各国政府都采取了严格的防疫措施来遏制它的蔓延。

然而，随着时间的推移，“XX”病毒的毒性逐渐降低，各国政府也渐渐放松了防疫措施。除了一个与世隔绝的封闭国家之外，几乎所有国家都开始解除了封锁。

然而，这种局面并没有持续多久。一年后，“XX”病毒突然产生了一个突变，它的致死率再次提高。在短短几个月内，“XX”病毒席卷了全世界，导致人类灭绝。

作为世界上唯一一个未被感染的国家，我们继承了人类的遗产，成为了最后的希望。我们招募了来自世界各地的优秀科研人员，致力于消灭“XX”病毒，夺回人类的明天。

这是我们的故事。

LA

这部小说都有哪些章节？每个章节分别是哪些故事内容？



整部小说可以分为五个章节，分别是：

1. 引言：介绍“XX”病毒的起源和人类面临的威胁。
2. 集结：讲述我们如何集结了来致力于消灭“XX”病毒的科研人员。
3. 研究：展示我们如何进行病毒研究，寻找有效的抑制方法。
4. 战斗：描述人类如何和“XX”病毒展开激烈的战斗。
5. 胜利：讲述人类最终如何成功消灭了“XX”病毒，夺回了人类的明天。

LA

请撰写小说第一章的内容。



在2100年，人类面临着一场前所未有的灾难。这场灾难的名字叫做“XX”病毒。

“XX”病毒起源于一个被遗弃的军事实验室，它被设计用来制造一种能够抑制人体免疫系统的武器。然而，这个实验失败了，“XX”病毒在实验室内自由传播，最终逃出实验室，感染了整个城市的居民。

“XX”病毒的传染性非常强，它可以通过空气传播，也可以通过接触传播。一开始，它的致死率非常高，每一个感染者都有很大概率会死亡。为了遏制“XX”病毒的蔓延，各国政府都采取了严格的防疫措施，包括封锁城市，进行大规模的消毒和隔离患者。

然而，随着时间的推移，“XX”病毒的毒性逐渐降低，感染者的存活率也逐渐提高。因此，各国政府也渐渐放松了防疫措施，除了一个与世隔绝的封闭国家之外，几乎所有国家都开始解除了封锁。

人类以为自己已经成功地控制住了“XX”病毒，

LA

请撰写小说第二章的内容。



在第一章中，我们已经了解了“XX”病毒的起源和威胁。在第二章中，我们将讲述人类是如何集结起来，共同对抗“XX”病毒的。

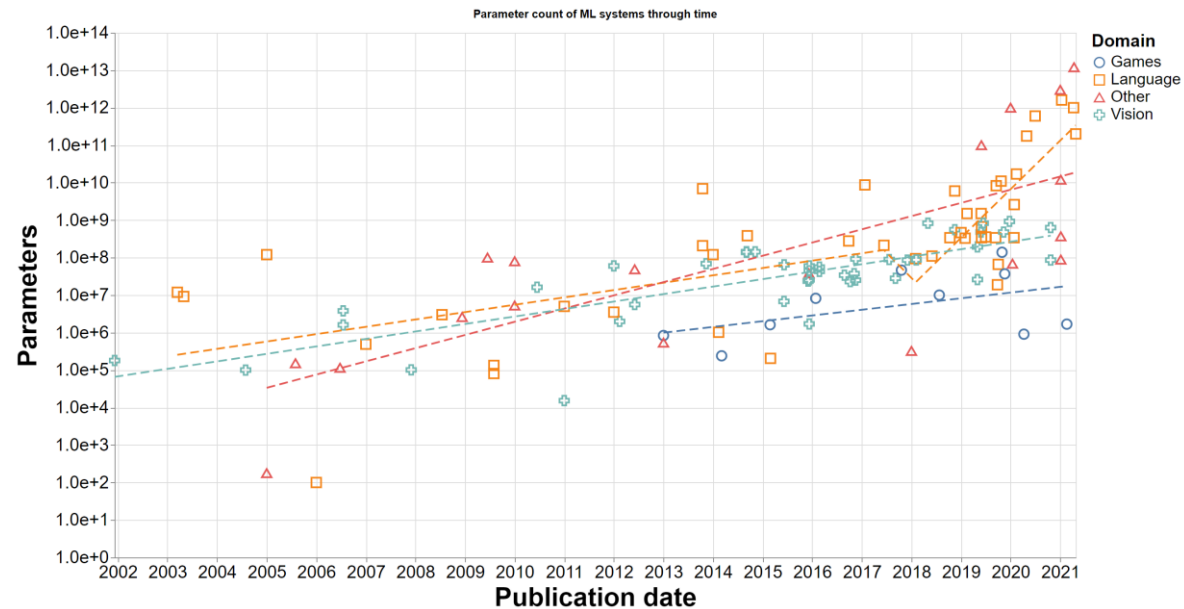
作为世界上唯一一个未被感染的国家，我们继承了人类的遗产，成为了最后的希望。我们招募了来致力于消灭“XX”病毒的优秀科研人员，集中在一个专门设立的研究所内。

在研究所内，我们首先进行了大量的病毒样本收集工作。我们在世界各地的病毒搜集站进行样本采集，并通过严格的实验室安全措施进行分析。

随着样本的不断积累，我们开始进行病毒基因测序，以了解“XX”病毒的遗传结构和变异情况。我们还建立了一个超级计算机，用于分析病毒数据，寻找有效的抑制方法。

此外，我们还设立了一个专门的实验室，用于研究“XX”病毒对人体的作用机制，并寻找有效的抑制方法。我们还设置了一个高安全性的医院，用于治疗患者。

# AI算力和内存需求远超摩尔定律 分布式异构并行计算是必由之路

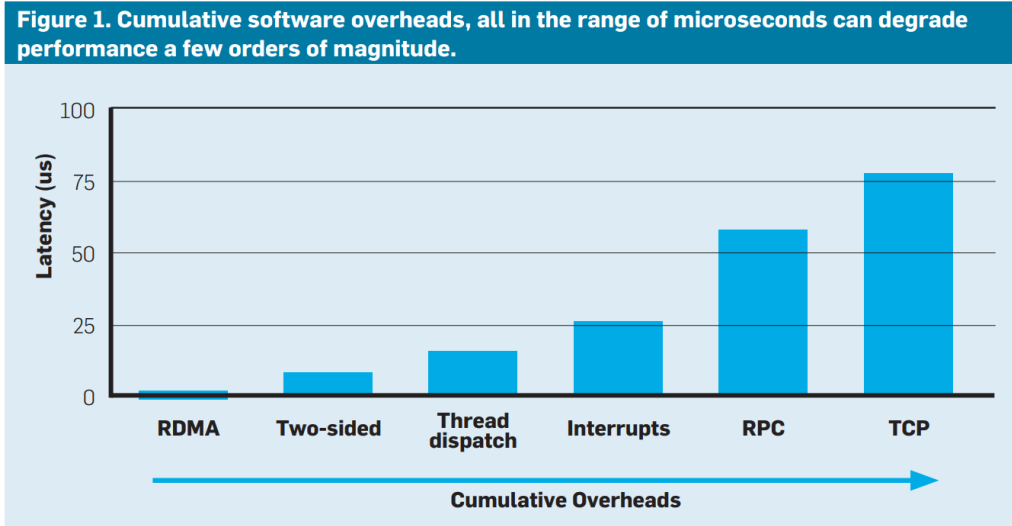


模型大小（内存需求）的增长趋势  
（纵坐标为对数）

模型训练算力需求的增长趋势（纵坐标为对数）



# 传统数据中心网络软件栈厚重，开销巨大



Attack of the Killer Microseconds, CACM, April 2017.

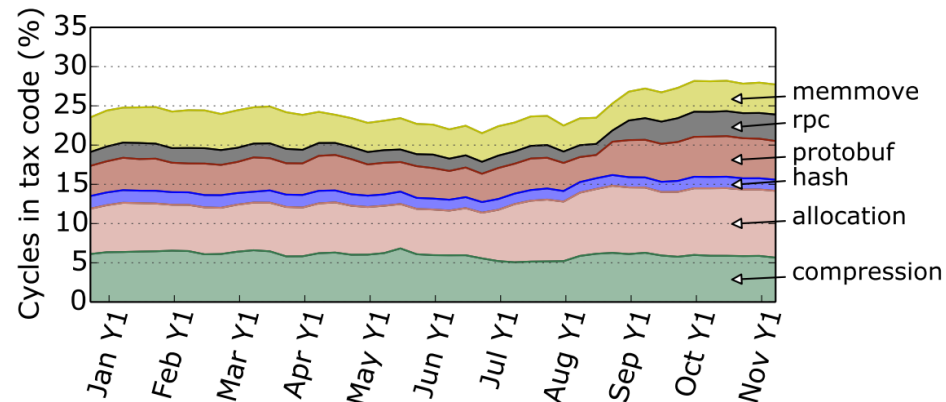


Figure 4: 22-27% of WSC cycles are spent in different components of "datacenter tax".

Profiling a warehouse-scale computer, ISCA'15.

- 数据中心网络硬件的时延是微秒级的，但由于厚重的网络协议栈，到了TCP层次就是数十微秒，到RPC层次甚至可达百微秒级
- 就像是北京到天津高铁只要30分钟，但坐地铁需要2个小时，结果端到端的时间长达4个小时
- 谷歌的统计表明，约25%的CPU被浪费在了网络协议栈中，包括内存拷贝、RPC、序列化、哈希、内存分配、压缩等

# 微秒级时延如何隐藏也是数据中心的难题

**Table 1. Events and their latencies showing emergence of a new breed of microsecond events.**

<b>nanosecond events</b>	<b>microsecond events</b>	<b>millisecond events</b>
register file: 1ns–5ns	datacenter networking: $O(1\mu s)$	disk: $O(10ms)$
cache accesses: 4ns–30ns	new NVM memories: $O(1\mu s)$	low-end flash: $O(1ms)$
memory access: 100ns	high-end flash: $O(10\mu s)$	wide-area networking: $O(10ms)$
	GPU/accelerator: $O(10\mu s)$	

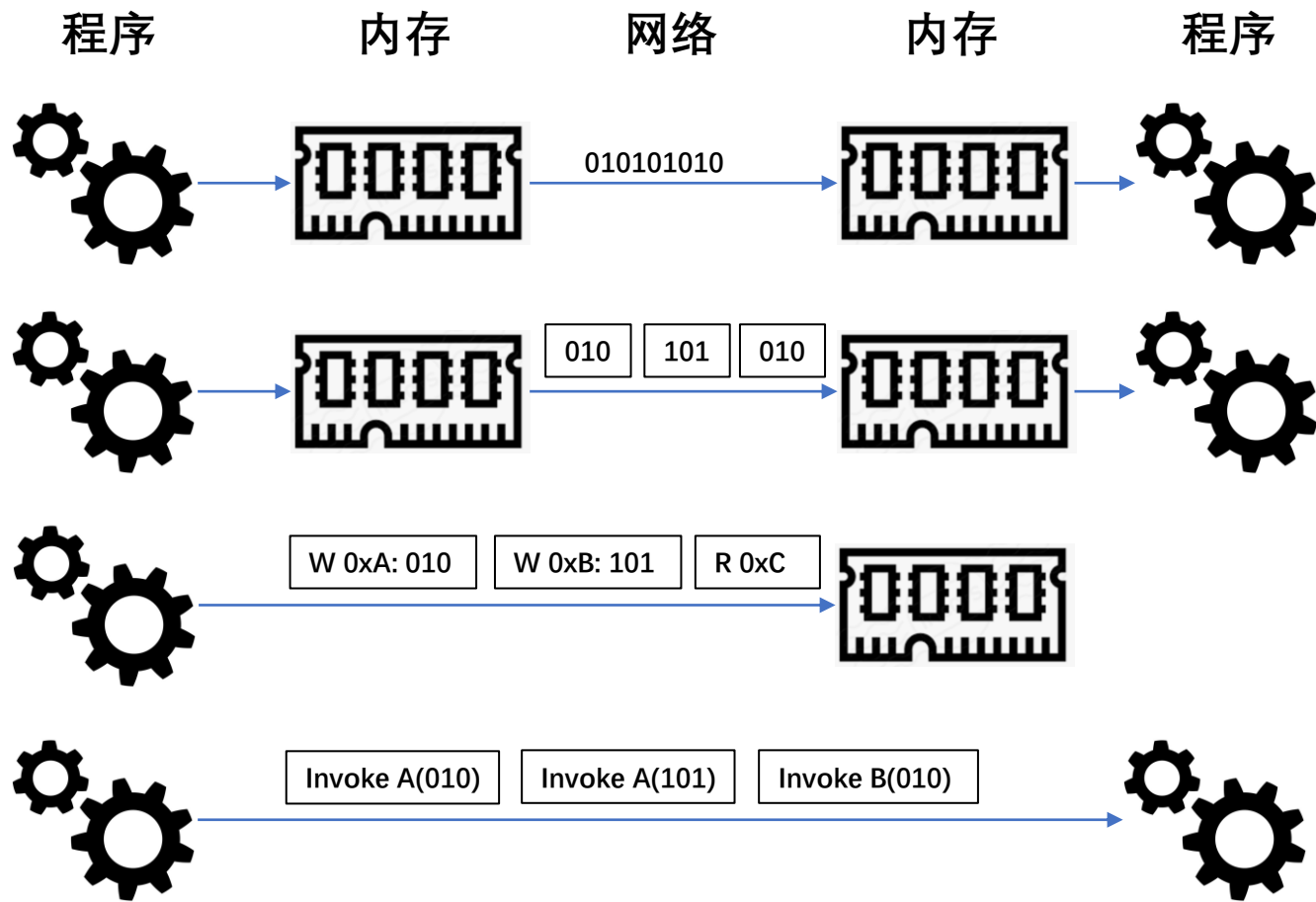
Attack of the Killer Microseconds, CACM, April 2017.

- 除了网络硬件时延，存储设备的访问也从毫秒级降到微秒级
- 等待微秒级的事件（网络通信、存储读写）时，CPU该干什么？
- 对于纳秒级的事件，CPU硬件流水线足以隐藏时延，无需软件干预
- 对于毫秒级的事件，操作系统可以切换到其他进程/线程
- 但对于微秒级的事件，操作系统切换的开销就已经是微秒级，可能得不偿失

# 目录

- 数据中心网络
  - 从Web服务到大规模异构并行计算
  - **编程抽象：从字节流到内存语义**
  - 把数据中心作为一台计算机
- 广域网
  - 从“尽力而为”到“准确性”
  - 东数西算与全国一体化大数据中心
- 无线网络
  - “1+8+N”与“5G to B”
  - 应用和数据无缝流转的“分布式超级终端”

# 字节流 vs. 内存语义



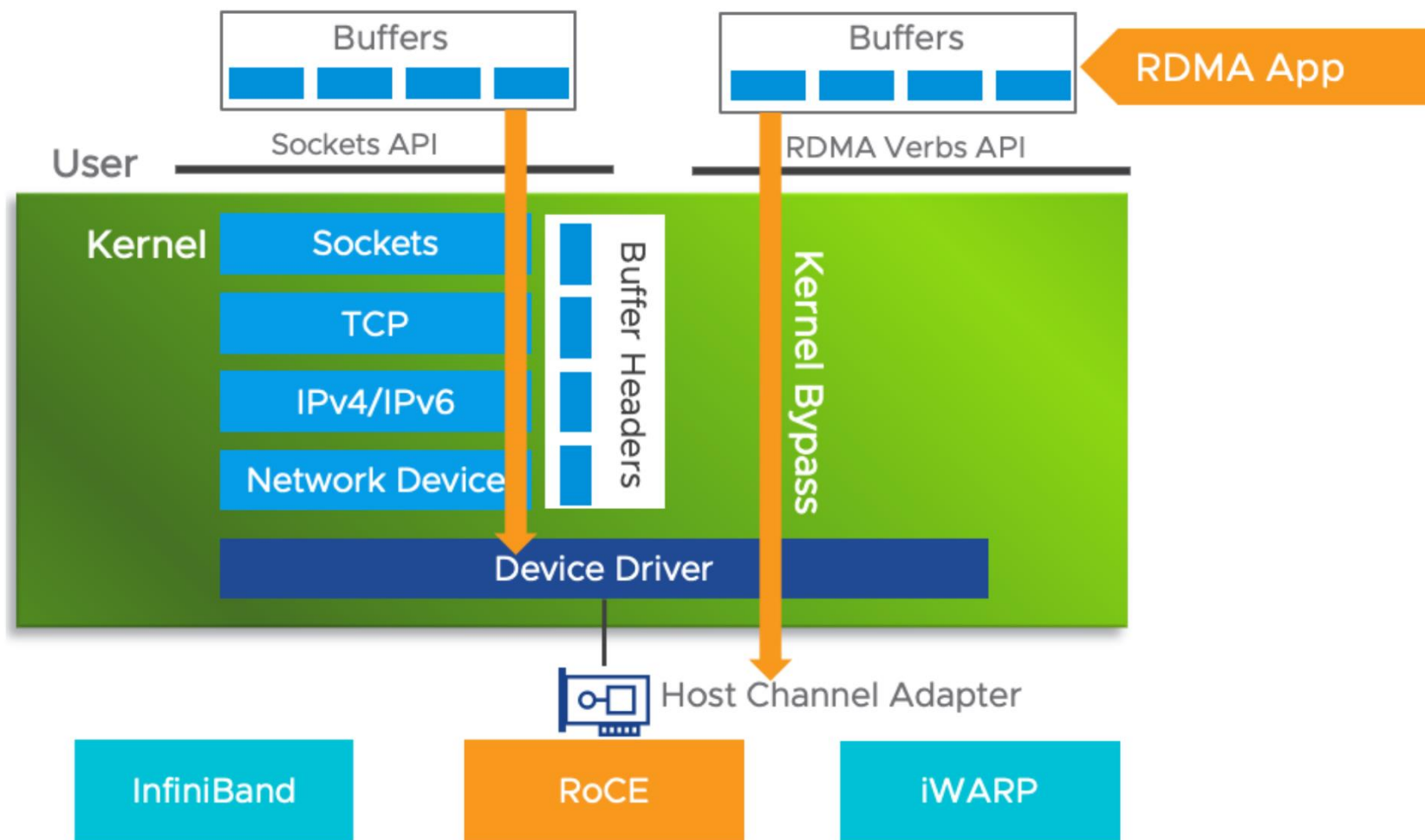
语义	例子
字节流	Socket send/recv
消息语义	ZeroMQ / Kafka RDMA send/recv
远端内存访问	RDMA read/write /atomic CXL Load/Store
RPC	gRPC, Thrift

内存语义

# 为什么是内存语义而非字节流？

- 字节流的抽象层次太低，没有消息边界的概念，需要封包拆包，增加额外的内存拷贝
- 字节流是有序的，无法利用通信中的并行性
  - 多条物理链路、选择性重传等
  - 现实中的多个函数调用（RPC）、多个远端内存访问（RMA）之间未必有强的顺序依赖，使用内存语义可以充分利用通信中的并行性
- 字节流不区分消息的重要性和QoS需求，无法按优先级传输
- 字节流是面向连接的，需要维护大量连接状态，为每个连接预留缓冲区，内存占用量大

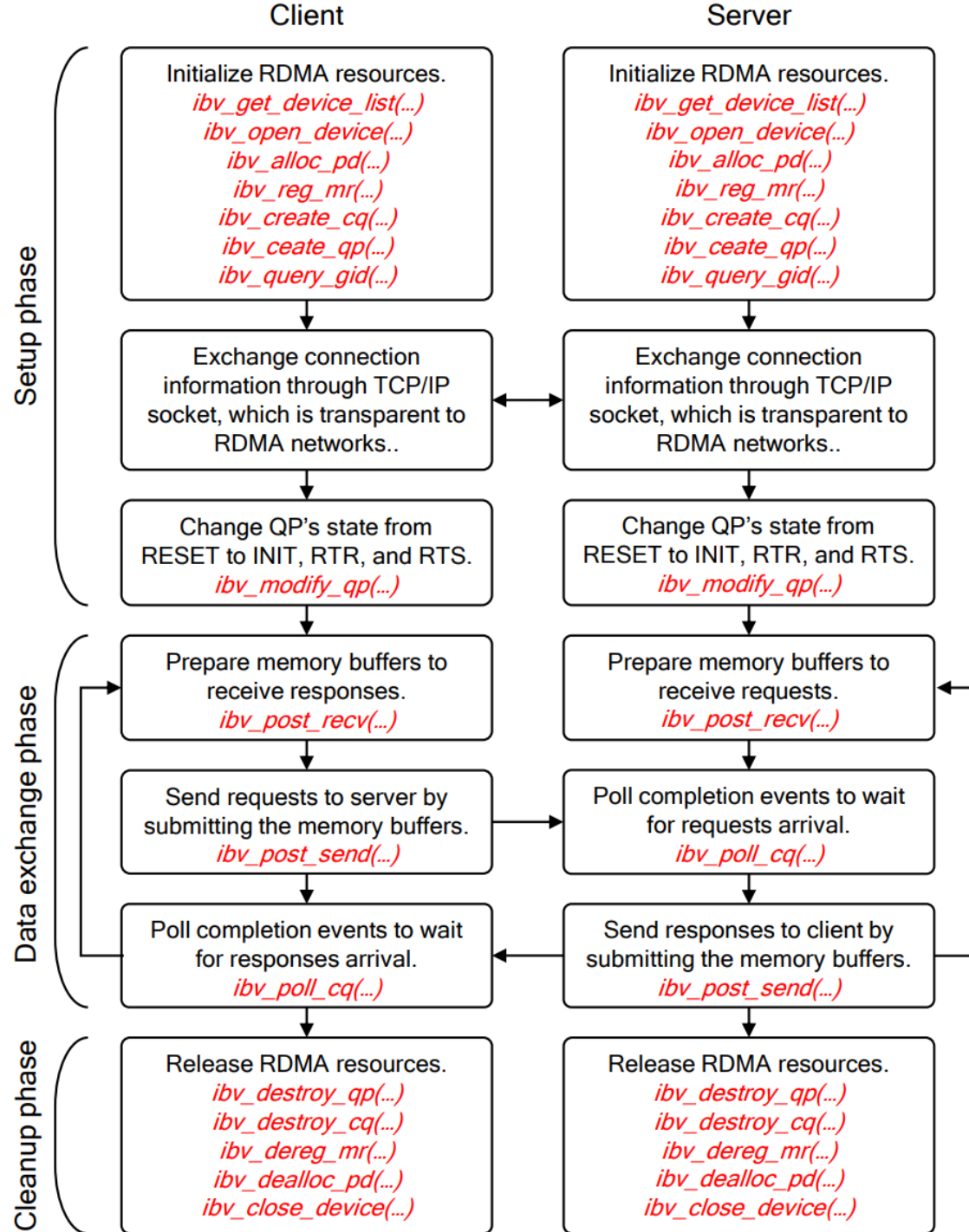
# 软硬结合，RDMA高效实现异步内存语义



相同Mellanox网卡条件下，TCP/IP协议栈时延为30 us，RDMA时延仅为2 us，快15倍

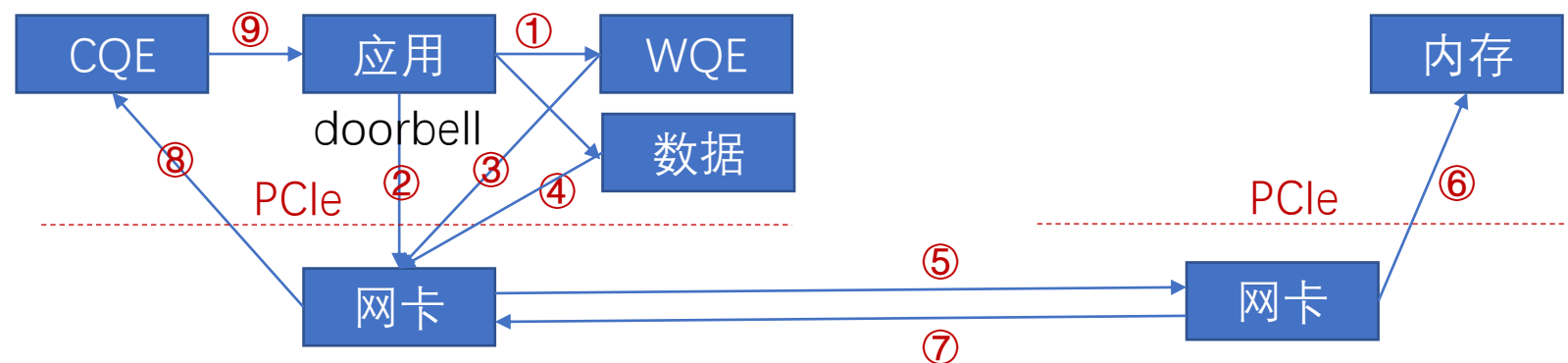
# RDMA的使用

- RDMA API比TCP/IP socket复杂，其主要原因是：
- 内存需要事先注册到网卡才能访问，这些内存需要pin住，即不能换入换出，由于网卡需要使用物理地址而非虚拟地址访问内存；
- 为了实现零拷贝，发送和接收操作都是异步的，发送后发送端可异步等待发送完成事件；接收端需要事先指定接收缓冲区，异步等待网卡生成的接收完成事件；
- 为了实现kernel bypass，传输协议栈在网卡硬件中实现，需要创建发送和接收队列，在用户态与网卡直接交互；

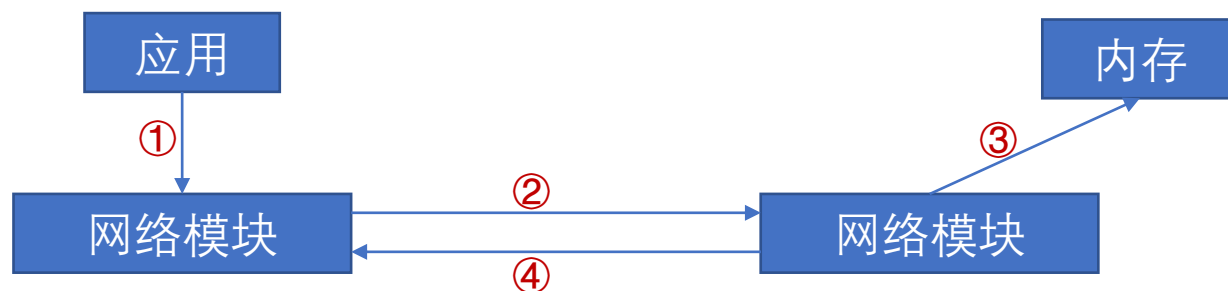


# CXL Load/Store: 同步远端内存访问

- RDMA是**异步**远端内存访问，每次访问需要多次PCIe交互，时延最低也需要1.6 us



- CXL Load/Store是**同步**远端内存访问，CPU直出网络，指令直接访问远端内存，无需经过PCIe，无需WQE、CQE、doorbell开销，时延 $<0.5$  us



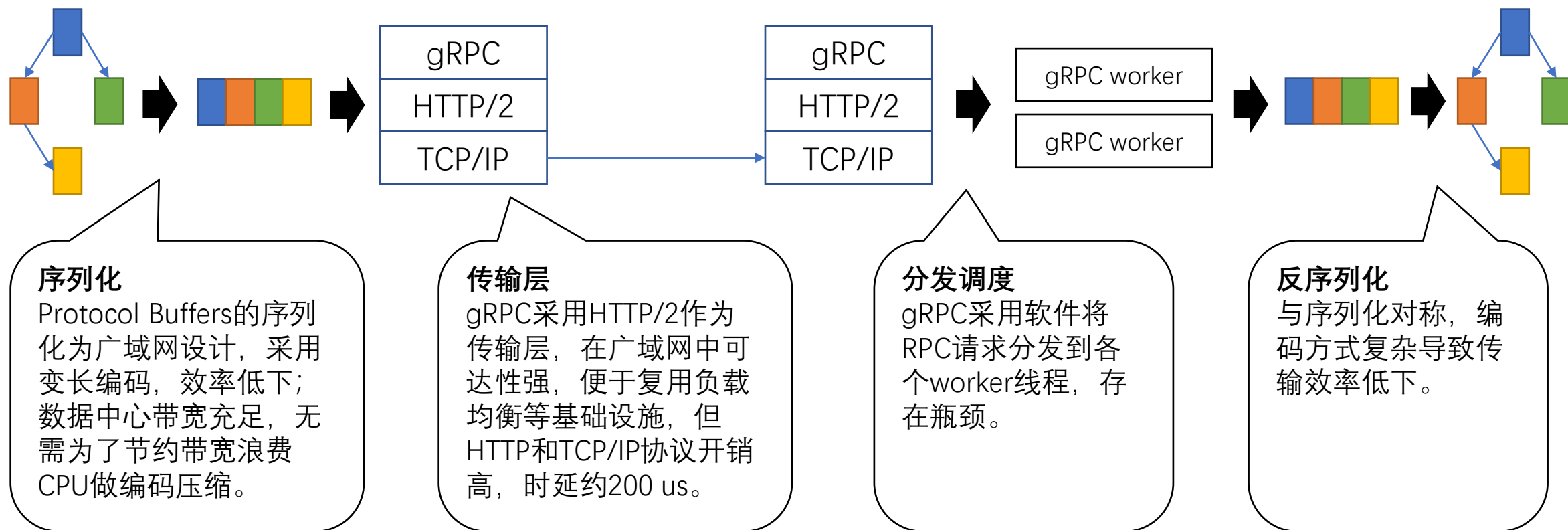


# 同步和异步远端内存访问的比较

	同步远端内存访问	异步远端内存访问
粒度	Cache line	用户指定的消息大小
时延	低	高
大块数据的访问效率	低	高
应用透明性	应用无感，可用于扩展本地内存，实现应用无感的内存池化	应用需要显式访问远端内存，如果用于内存扩展，应用需要修改
对硬件的要求	高，需要网卡与CPU紧密配合	低，网卡可以是分离式的形态
可靠性	爆炸半径大，一个节点故障会影响使用了该节点远端内存的所有节点，异步指令错误难以捕获	容易通过应用捕获异步远端访问异常，将爆炸半径缩小到受影响的应用
缓存一致性	取决于硬件是否支持，但在大规模下硬件支持缓存一致性的开销高	不支持，软件显式在远端和本地内存之间拷贝，在有共享的情况下需要配合分布式锁保证一致性

# RPC: 跨主机、跨进程、跨语言的函数调用

gRPC是当前应用最广泛的RPC框架，但其性能较差，Hello World需要250 us，是数据中心网络硬件时延的50倍



# 连马斯克都关心起RPC了：RPC是微服务间的通信方式



Elon Musk @elonmusk · 11月14日

Btw, I'd like to apologize for Twitter being super slow in many countries. App is doing >1000 poorly batched RPCs just to render a home timeline!

2.4万 1.4万 17.3万



@EricFrohnhoefer@mastodon.social  
@EricFrohnhoefer

Hot take

I have spent ~6yrs working on Twitter for Android and can say this is wrong.

翻译推文



Elon Musk @elonmusk · 11月14日

Btw, I'd like to apologize for Twitter being super slow in many countries. App is doing >1000 poorly batched RPCs just to render a home timeline!



Elon Musk @elonmusk · 11月14日

回复 @EricFrohnhoefer

Then please correct me. What is the right number?

3,966 3,969 10.8万



Elon Musk @elonmusk · 11月14日

回复 @elonmusk 和 @EricFrohnhoefer

Twitter is super slow on Android. What have you done to fix that?

2,689 2,275 3.2万



Elon Musk @elonmusk

Replying to @lsngdon @EricFrohnhoefer and @pokemoniku

He's fired

9:32 PM · Nov 14, 2022 · Twitter for iPhone



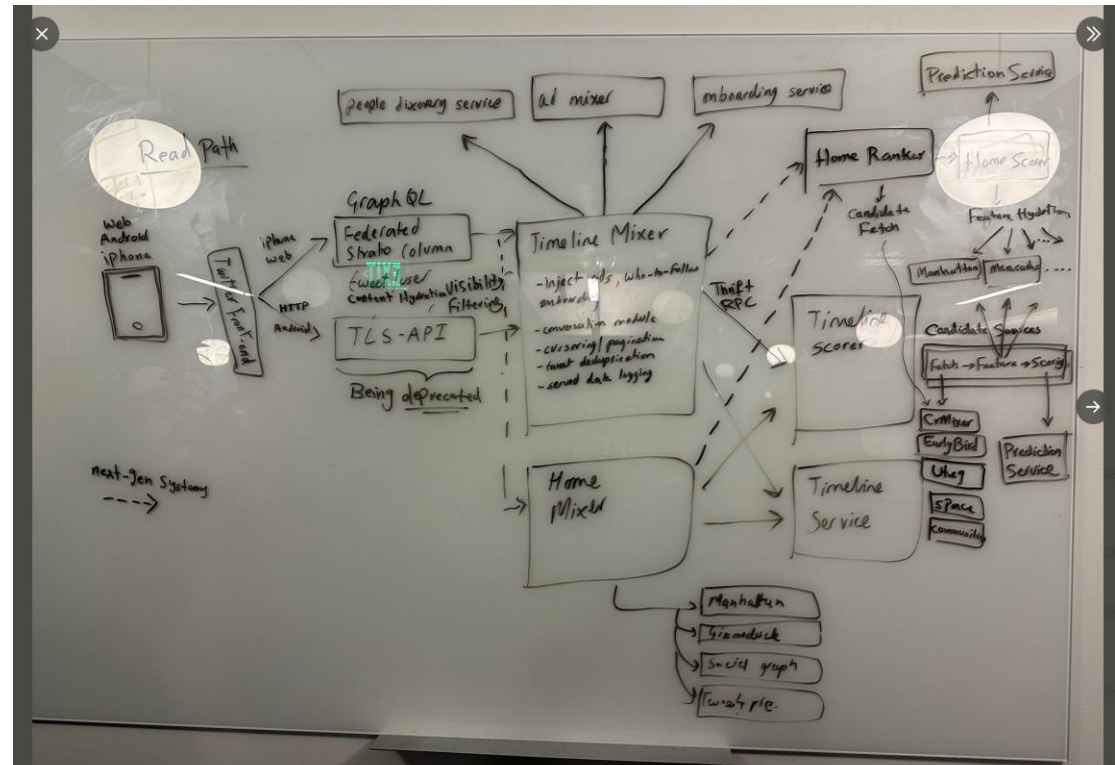
Elon Musk @elonmusk · 11月14日

回复 @sampullara

I was told ~1200 **RPCs** independently by several engineers at Twitter, which matches # of microservices. The ex-employee is wrong.

Same app in US takes ~2 secs to refresh (too long), but ~20 secs in India, due to bad batching/verbose comms. Actually useful data transferred is low.

603 859 8,226



Elon Musk @elonmusk

Just leaving Twitter HQ code review

下午5:28 · 2022年11月19日

3.6万 转推 1.6万 引用推文

46.2万 喜欢次数

发布你的回复

Edward ... @... · 11月20日

回复 @elonmusk

Trust me man, you don't want to lose this - write it down somewhere. Like on a napkin or something. That's what all the real disruptive geniuses do.

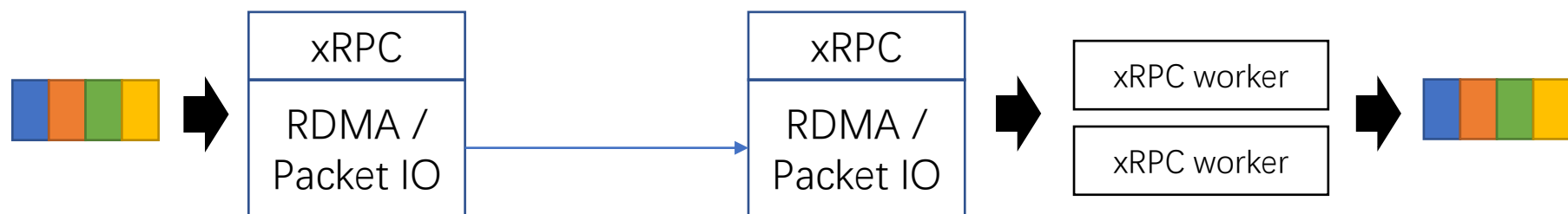


62 77 1,180

TRILLY @SaitaTrilly · 11月19日

# 业界的高性能内存语义RPC

- FaSST RPC: RPC over **RDMA**
  - eRPC: RPC over **Packet I/O**
- } 可达到5 us级的端到端时延, 比gRPC低50倍



## 序列化

Flat Buffers等序列化工具无需对数据做额外的编码, 数据免拷贝, 无需额外格式转换, 效率高。

## 传输层

采用为数据中心优化的RDMA传输层或者基于Packet IO的用户态RPC协议栈, 消除内核TCP/IP协议栈开销, 时延降低到us级。

## 分发调度

使用硬件的RSS能力 (Packet IO) 或RDMA QP分发能力, 请求直通worker线程, 避免软件集中式分发RPC请求。

## 反序列化

Flat Buffers无需反序列化, 数据免拷贝, 无需额外格式转换, 效率高。

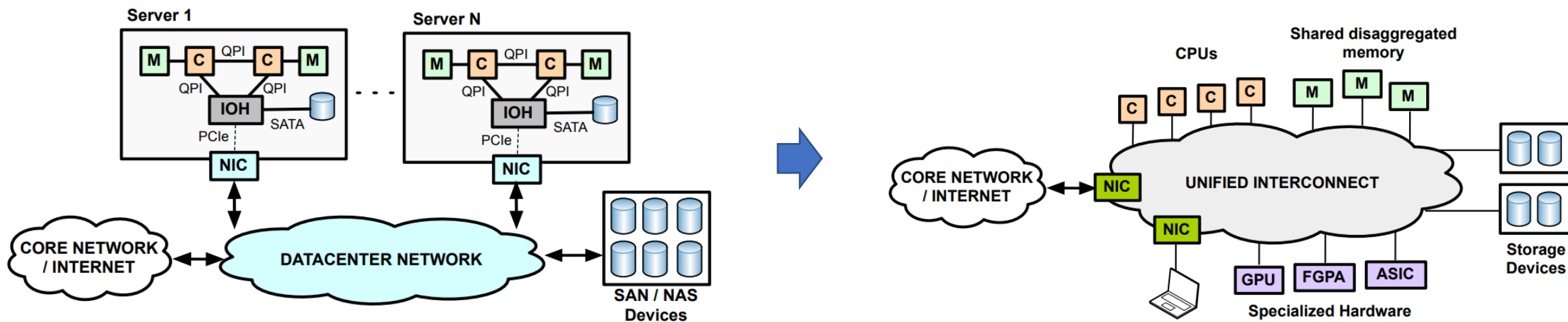
# 目录

- 数据中心网络
  - 从Web服务到大规模异构并行计算
  - 编程抽象：从字节流到内存语义
  - **把数据中心作为一台计算机**
- 广域网
  - 从“尽力而为”到“准确性”
  - 东数西算与全国一体化大数据中心
- 无线网络
  - “1+8+N”与“5G to B”
  - 应用和数据无缝流转的“分布式超级终端”

# 把数据中心作为一台计算机

- 数据中心互联像一台计算机的内部总线一样高性能
- 数据中心内分布式系统的编程像单机编程一样便捷

# 从以CPU为中心的体系结构到异构对等互联



Network Requirements for Resource Disaggregation, OSDI'16

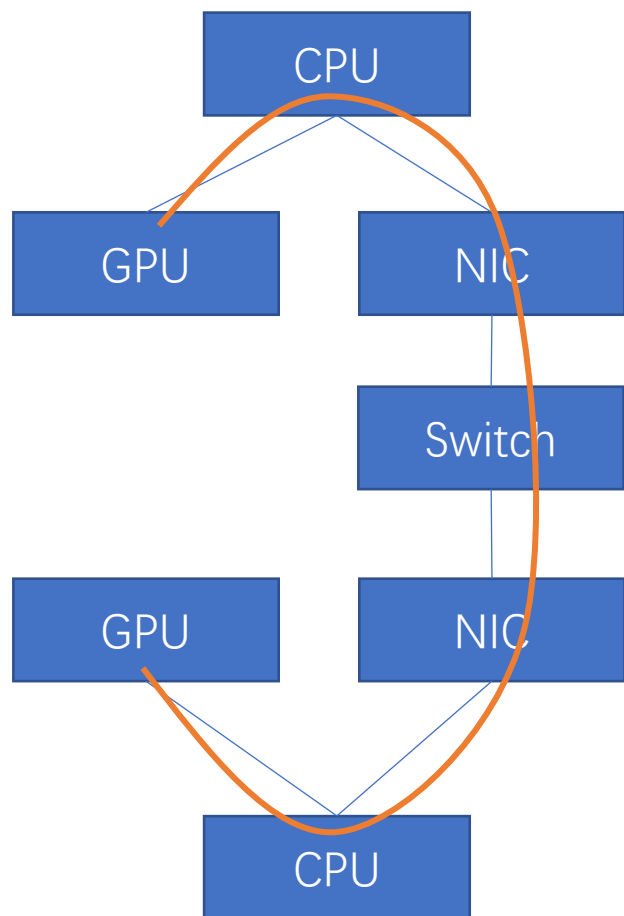
有哪些异构设备？

- 异构计算设备：CPU、GPU、NPU（AI芯片）、DPU（数据处理器）、FPGA……
- 异构存储设备：HBM、DDR、NVM、Flash、磁盘……

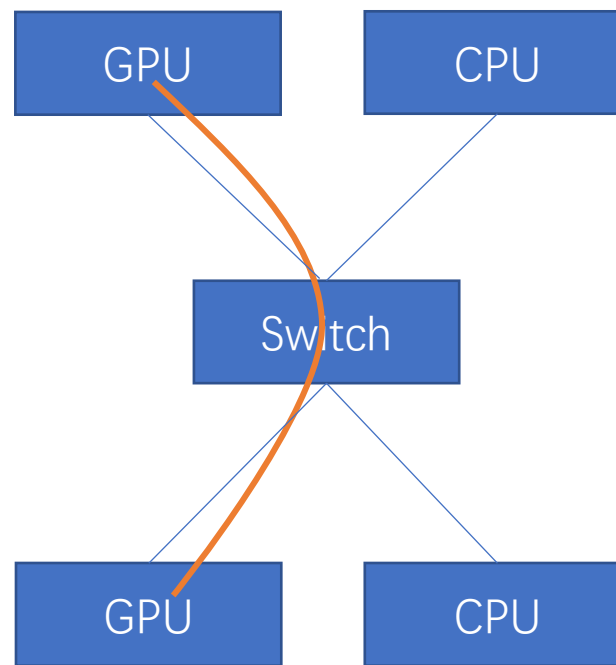
当前数据中心的存储集群与计算集群已经实现了分离，一些先进的数据中心内，存储和计算集群之间使用RDMA进行高效通信。但其他设备的异构对等互联尚未实现。

# 从以CPU为中心的体系结构到异构对等互联

传统以CPU为中心的体系结构：需要CPU转发



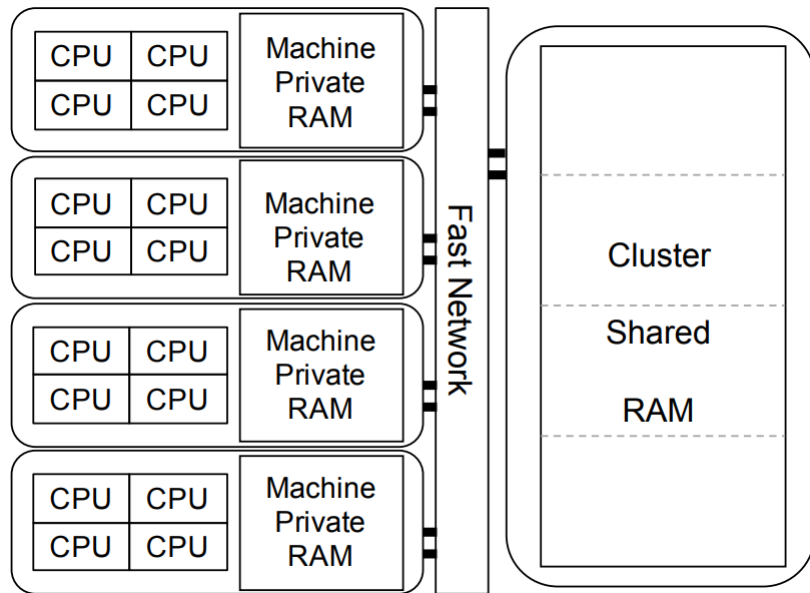
异构对等互联：无需转发，设备直通



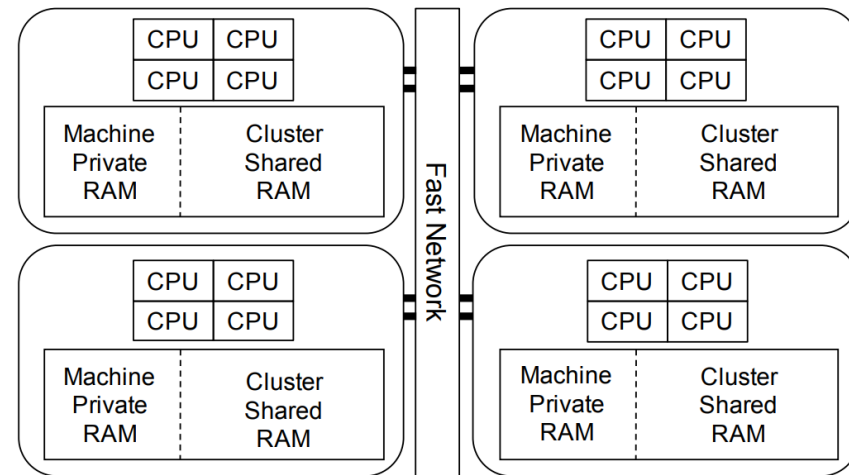


# 分离式内存 (Disaggregated Memory)

- 谷歌的统计表明，数据中心内存的成本占到服务器总成本的40%，未来内存成本可能成为CPU成本的5倍。
- 内存池化（分离式内存）、多种存储介质融合是降低硬件成本的趋势。

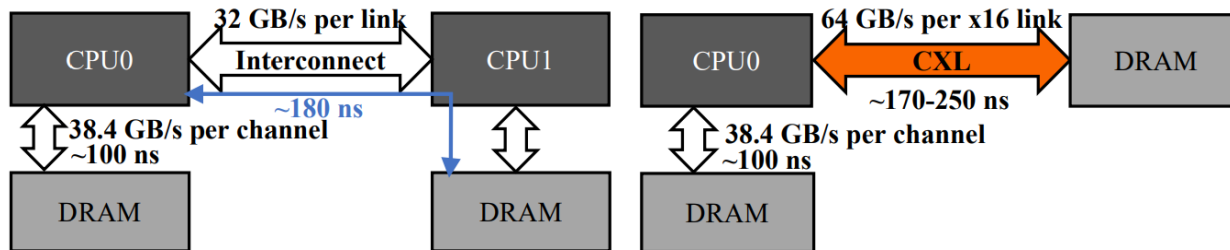
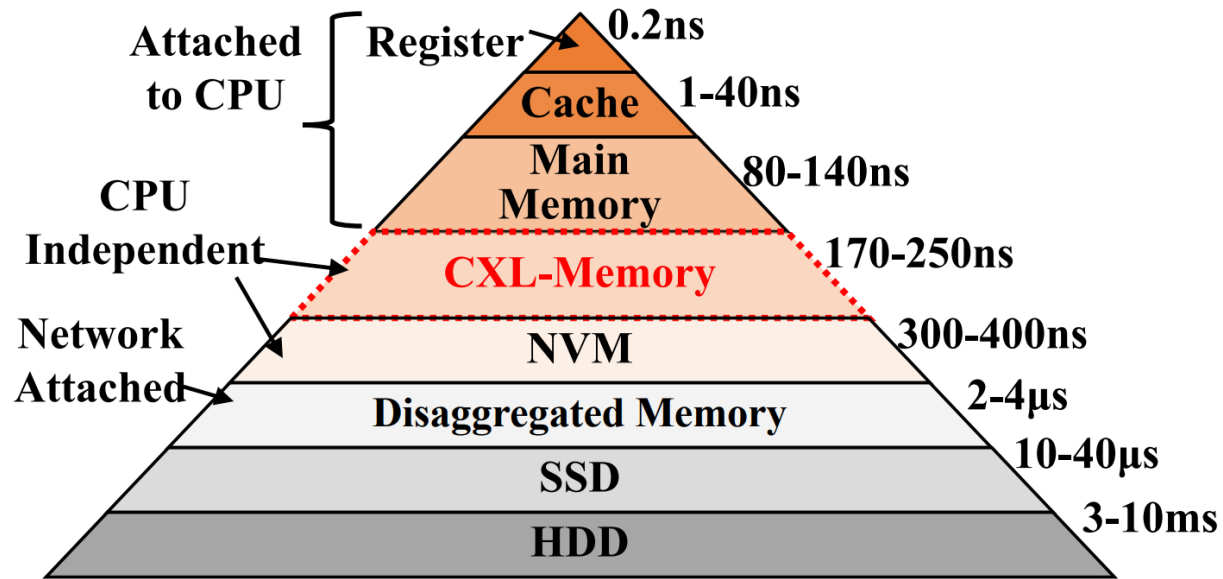


分离式内存



远端内存

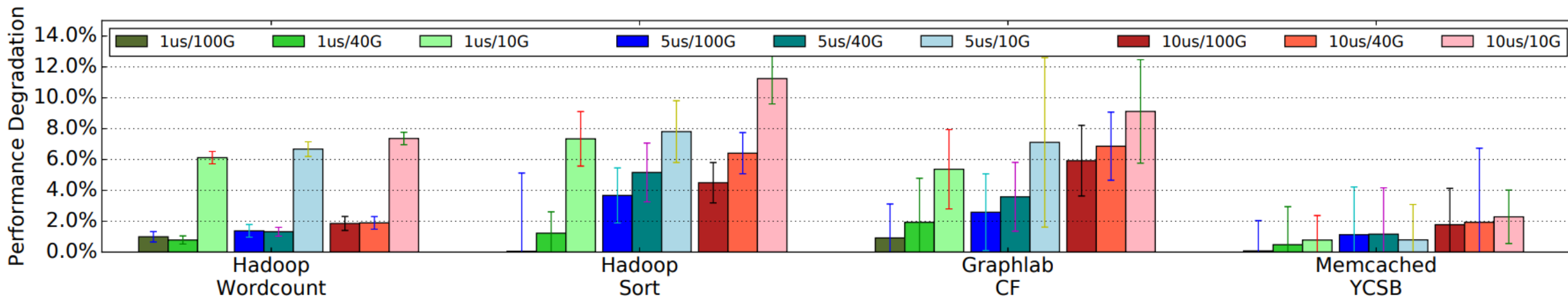
# 分级内存 (Tiered Memory)



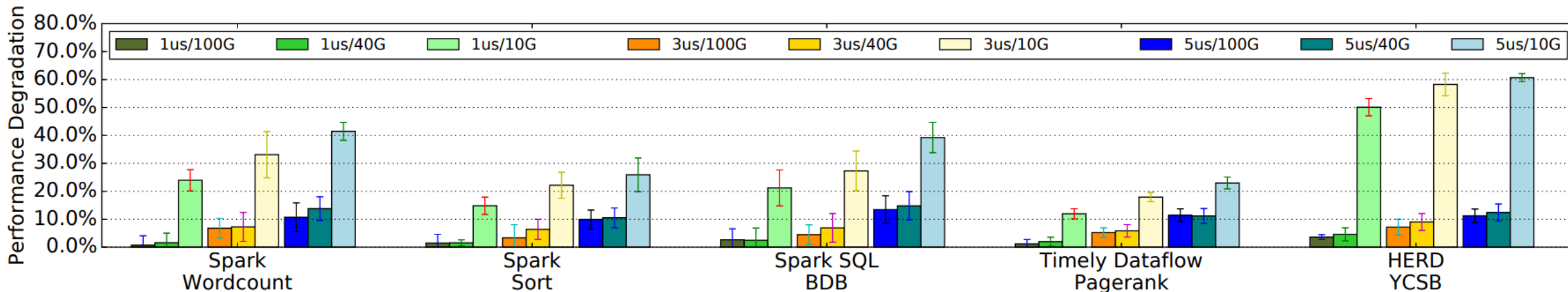
(a) Without CXL

(b) With CXL

# 分离式/分级内存会对应用性能造成多少影响？

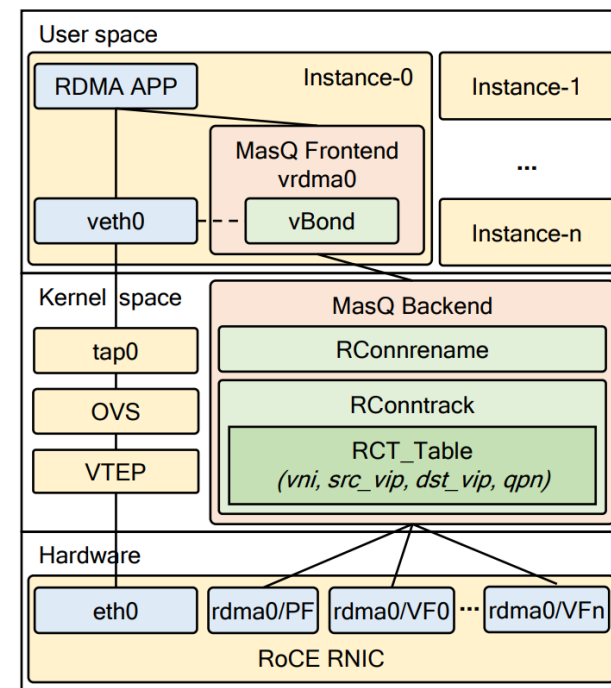


在本地内存25%的情况下，应用性能下降5%，上图需要5 us/40 Gbps，下图需要3 us/100 Gbps

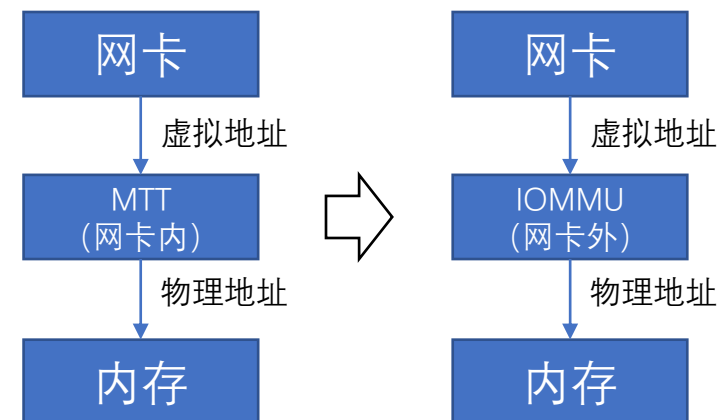


# 内存语义原生支持虚拟化

- 为了提高资源的利用率，云化是必然趋势，从而需要原生支持虚拟化和多租户
- 传统RDMA网卡不支持虚拟化
- FreeFlow、MasQ等学术界工作实现了RDMA的虚拟化，但性能有一定损失
- 下一代硬件需要原生支持内存语义的虚拟化，这将在硬件中引入额外的查表开销
- 通过IOMMU/SMMU与MMU共页表（或自动同步页表），实现网卡可以使用虚拟内存访问，无需pin住内存，使内存可以在分级内存中动态迁移



MasQ



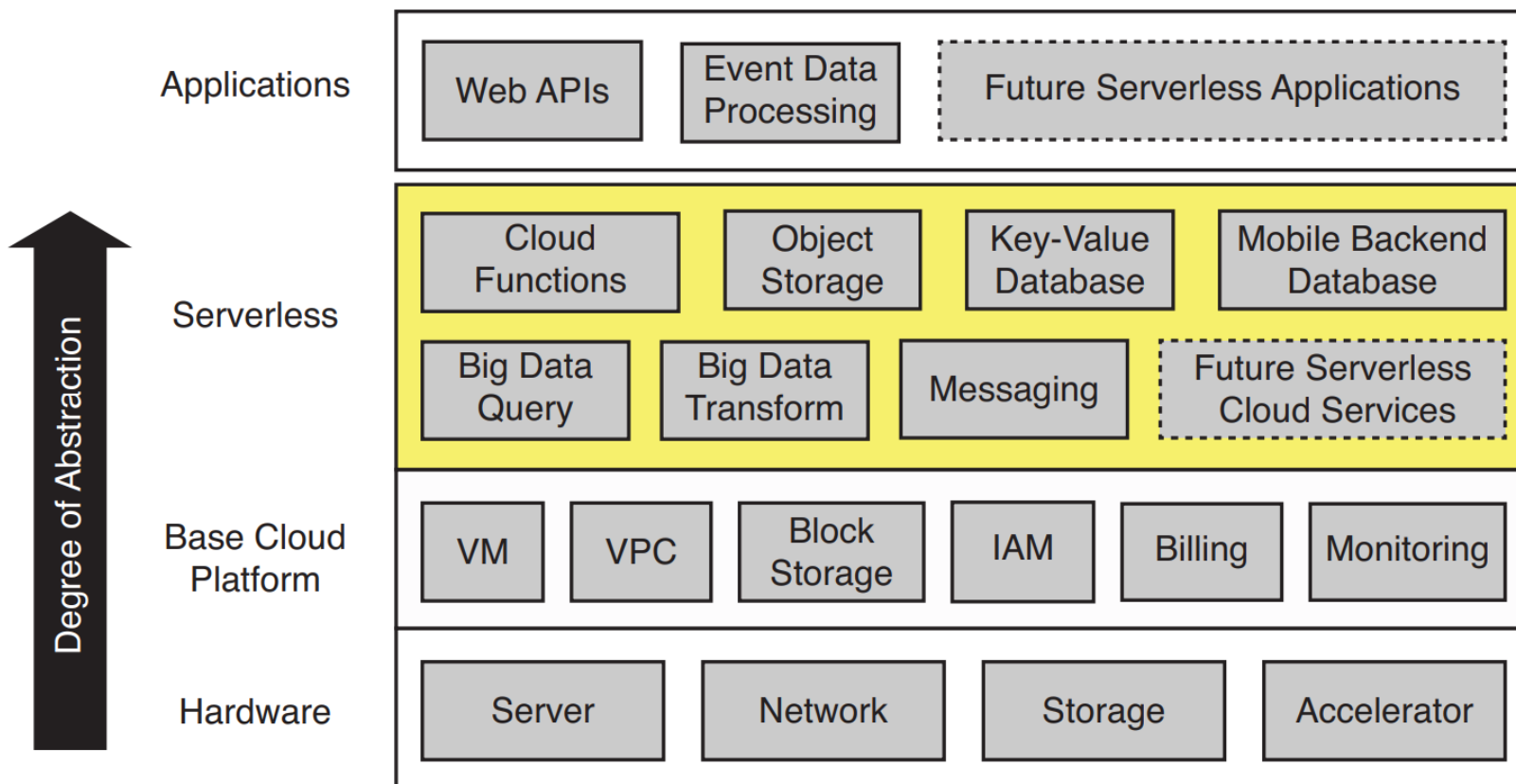
# Serverless: 让分布式系统编程像单机一样便捷

- 回顾：2009年Berkeley针对云计算的预测：
  - （理论上）无限可用的计算资源
  - 用户再也不需要承担服务器运维的工作和责任
  - 服务的按需付费成为可能
  - 超大型数据中心的使用成本显著降低
  - 通过可视化资源管理，运维操作的难度大大降低
  - 得益于分时复用，物理硬件的利用率大大提高
- 2019年，Berkeley预测Serverless将成为下一个重要的应用范式
- 现有云计算的IaaS模式有如下问题：
  - 为可用性做到冗余，这样一台机器的故障不会导致服务中断
  - 在发生灾难时保留服务的冗余拷贝的地理分布
  - 通过负载均衡、请求路由来高效利用资源
  - 根据负载变化自动伸缩系统
  - 监控服务确保它们一直健康地运作
  - 记录日志用于debug和性能调优
  - 系统升级，包括安全补丁
  - 在新实例可用时迁移到新实例

# Serverless: 让分布式系统编程像单机一样便捷

角色	特征	AWS Serverless	AWS Serverful (传统IaaS模式)
开发者	程序运行	用户选择的事件	持续地运行, 直到明确指出需要停止
	编程语言	JavaScript, Python, Java, Go, C#等	任意
	程序状态	持久化到存储中 (无状态)	任意 (无状态或有状态)
	最大内存	0.125到3g	0.5-1952g
	本地存储	0.5g	0-3600g
	最大运行时间	900s	无
	最小计费单元	0.1s	60s
	每个计费单元的价格	0.0000002美元	0.0000867-0.408美元
	操作系统和库	云厂商选择	用户选择
系统管理	服务实例	云提供商选择	用户选择
	伸缩	云提供商选择	用户选择
	部署	云提供商职责	用户职责
	容错	云提供商职责	用户职责
	监控	云提供商职责	用户职责
	日志	云提供商职责	用户职责

# Serverless云的架构



Cloud Programming Simplified: A Berkeley View on Serverless Computing

# Serverless适合的应用类型和目前的局限

百分比	场景
32	Web和API
21	数据处理, 比如ETL (抽取, 转换, 加载)
17	集成三方应用
16	内部工具
8	聊天机器人 比如Alex Skills (Alexa AI助手的SDK)
6	IoT

描述	挑战	Workarounds	性价比	
实时视频压缩	视频编码	对象存储太慢无法支持细粒度通信; 函数对于任务来说太粗粒度	函数到函数通信, 避免对象存储; 函数执行替代任务	VM的60倍速度, 六分之一价格
MapReduce	大数据处理 (100TB)	Shuffle不会因为对象存储延迟和IO的限制而伸缩	低延迟, 高IOPS的小存储来加速shuffle	比VM排序100TB快1%, 花费多15%
线性代数	大规模线性代数	需要很大的问题规模来克服存储延迟, 很难实现高效的广播	使用低延迟高吞吐的存储处理小规模问题	比原本慢3倍, 但是CPU消耗节省1.26到2.5倍
ML 流水线	规模化的ML训练	缺少告诉存储来实现参数服务器, 很难实现高效广播, 聚合	低延迟存储, 高IOPS的参数服务	比VM快3-5倍, 成本高7倍
数据库	有主状态的应用	缺少共享内存, 对象存储延迟太高, 缺少链接管理的支持	如果没有过多的写入操作, 可以通过共享文件系统解决	每个事务的成本高3倍, 读规模相当但是写不是



# 目录

- 数据中心网络
  - 从Web服务到大规模异构并行计算
  - 编程抽象：从字节流到内存语义
  - 把数据中心作为一台计算机
- 广域网
  - 从“尽力而为”到“准确性”
  - 东数西算与全国一体化大数据中心
- 无线网络
  - “1+8+N”与“5G to B”
  - 应用和数据无缝流转的“分布式超级终端”

# 实时互动广域网应用对时延和带宽的需求

## 传统应用

- Web网页
- 微信语音
- 网络游戏
- 下载文件
- 视频点播
- 100 Mbps带宽、100 ms~1s时延已经足够

## 新型应用

- 实时高清视频 (1 Gbps)
  - 例如: 卡塔尔世界杯8K高清直播
- 视频会议 (100 ms)
- 直播 (1s内端到端时延)
- 远程协同办公 (50 ms)
- 远程桌面 (10 ms)
- 云游戏 (10 ms)



## 实时音视频

### Real Time Communications (RTC)

#### 基本释义：

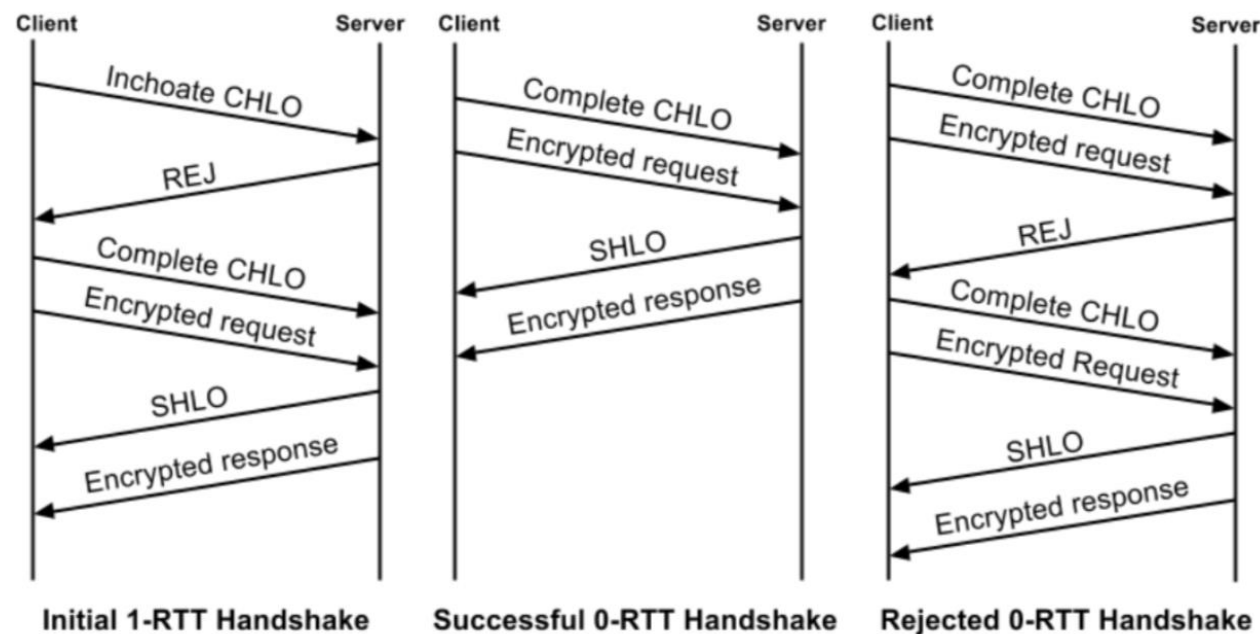
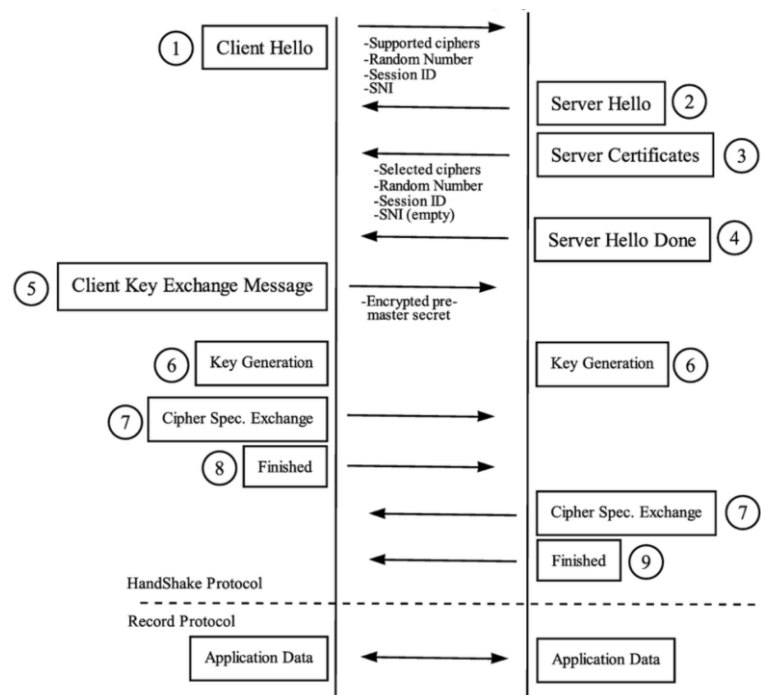
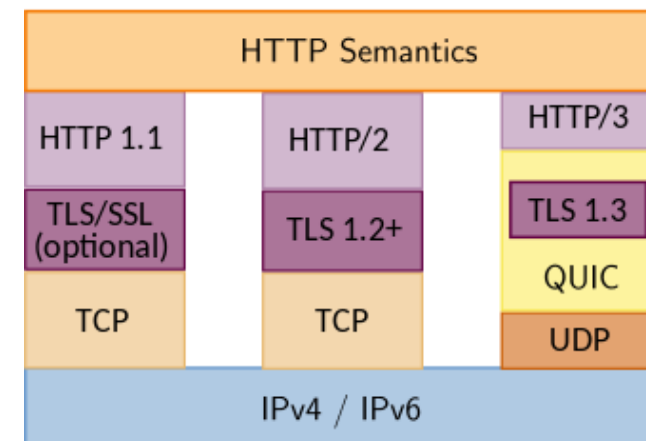
在**远程条件**下，以**接近实时/可忽略延迟**交换信息的通信方式，包括固定通话、移动通话、音频会议、视频会议、网真等形式。

#### 本篇报告研究的实时音视频（RTC）服务特指：

- **通信方式**：以音频、视频的形式赋能行业；
- **服务模式**：基于云计算技术理念提供，包括实时音视频的底层通用能力，以及垂直行业的解决方案；
- **底层技术**：采用软件定义的实时音视频传输网络（Real-time Network, RTN）和基于UDP的传输协议；
- **时延区间**：本篇报告聚焦于广义的实时音视频，包含时延400ms内的强互动，以及800ms内的超低延时互动。

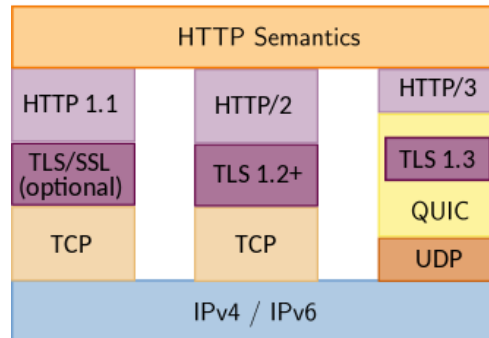
# QUIC改进通信协议栈，提升性能 (1)

- **连接建立**：HTTP/3基于QUIC/UDP，将TCP三次握手、TLS连接建立过程从TLS 1.2的3个RTT简化为0/1个RTT（初次建链1 RTT，二次建链0 RTT），大大缩短连接建立时间
- 对短HTTP请求效果尤其显著



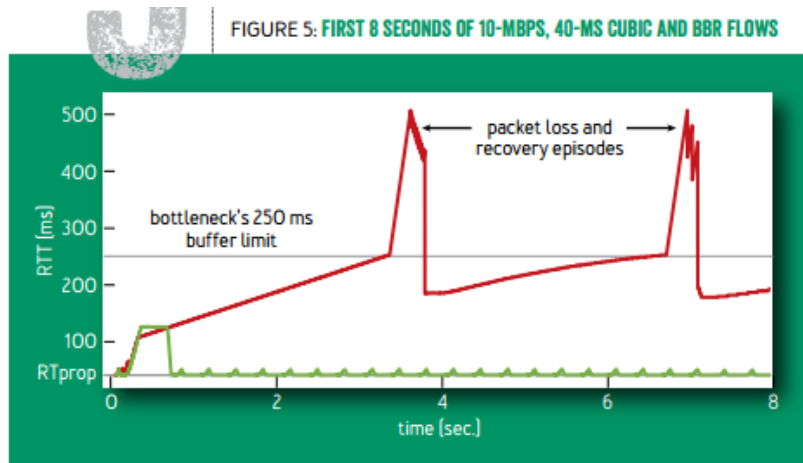
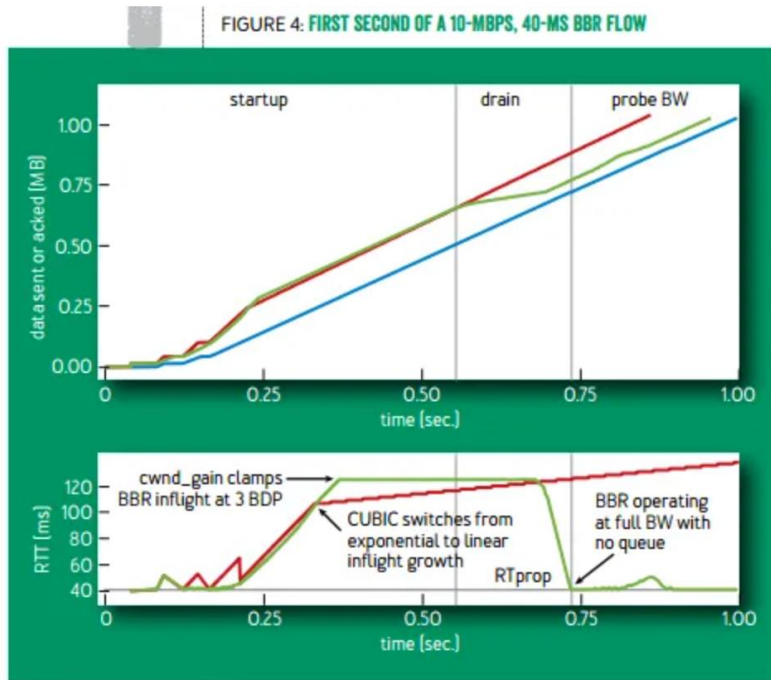
# QUIC改进通信协议栈，提升性能 (2)

- **拥塞控制**：HTTP/3可选多种拥塞控制算法，BBR在高带宽、高延迟、高丢包场景下表现较好



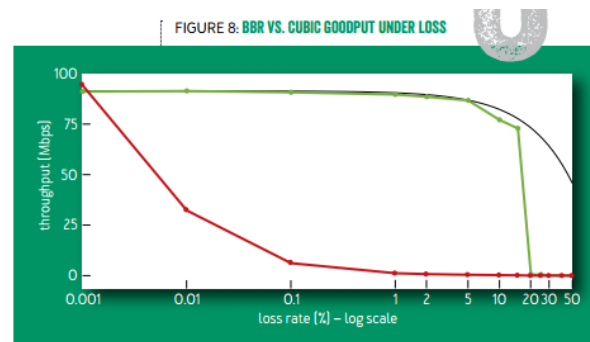
如何解决带宽和延迟无法同时测准？

- **交替测量带宽和延迟**；用一段时间内的带宽极大值和延迟极小值作为估计值。
- BBR稳定状态的绝大多数时间处于带宽探测阶段。



## TCP BBR如何解决？

- 既然不容易区分拥塞丢包和错误丢包，就干脆**不考虑丢包**。
- 既然灌满水管的方式容易造成缓冲区膨胀，TCP BBR就**分别估计带宽和延迟，而不是直接估计水管的容积**。



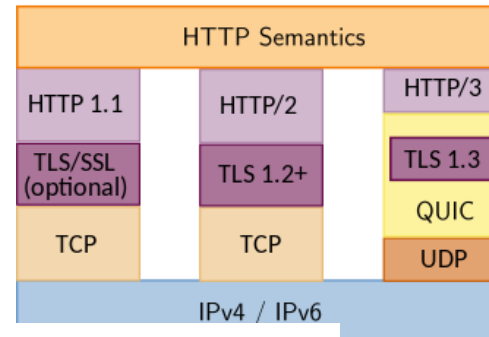
只要有万分之一的丢包率，标准TCP的带宽就只剩30%；千分之一丢包率时只剩10%；有百分之一的丢包率时几乎就卡住了。而BBR在丢包率5%以下几乎没有带宽损失，在丢包率15%的时候仍有75%带宽。

Cubic拥塞控制算法的**缓冲区膨胀**问题：

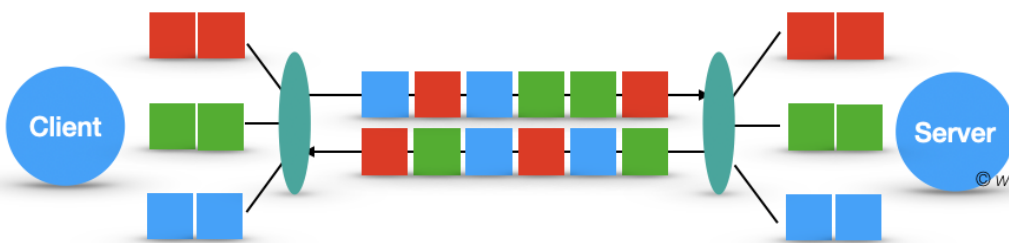
- 增加网络时延
- 共享连接数多时，缓冲区填满而丢包

# QUIC改进通信协议栈，提升性能 (3)

- 无队头阻塞的多路复用:



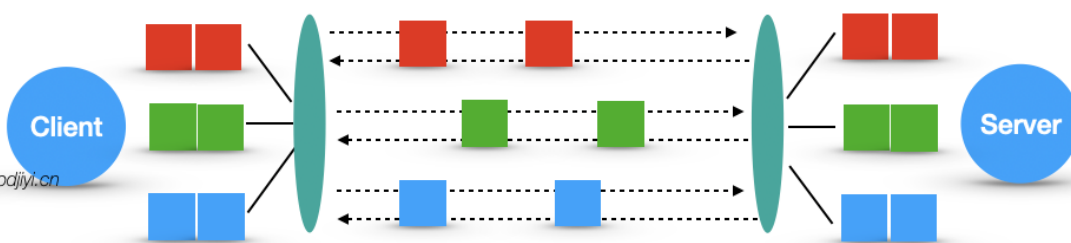
红绿蓝块分别代表不同连接里的数据帧



1. 复用一个 TCP 连接;
2. 基于流的数据帧传输, 请求可同时发送;
3. 一般同一个域名只需 1 个 TCP 并发;

## HTTP2 多路复用

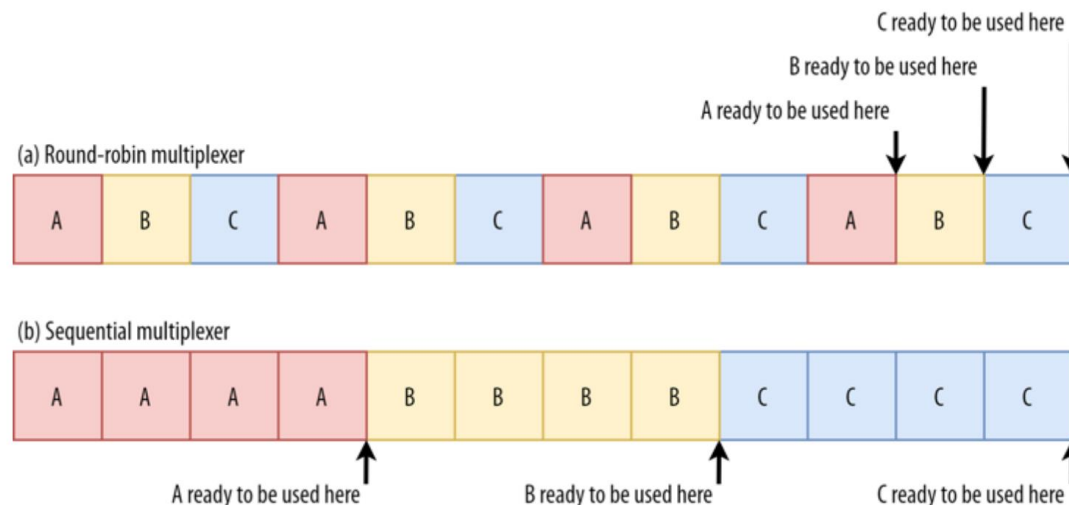
© www.godjiji.cn



1. 无 TCP 连接, 但QUIC实现了有序性;
2. QUIC 的有序性每个请求只会阻塞自身, 而不会影响其他请求;

## QUIC UDP传输

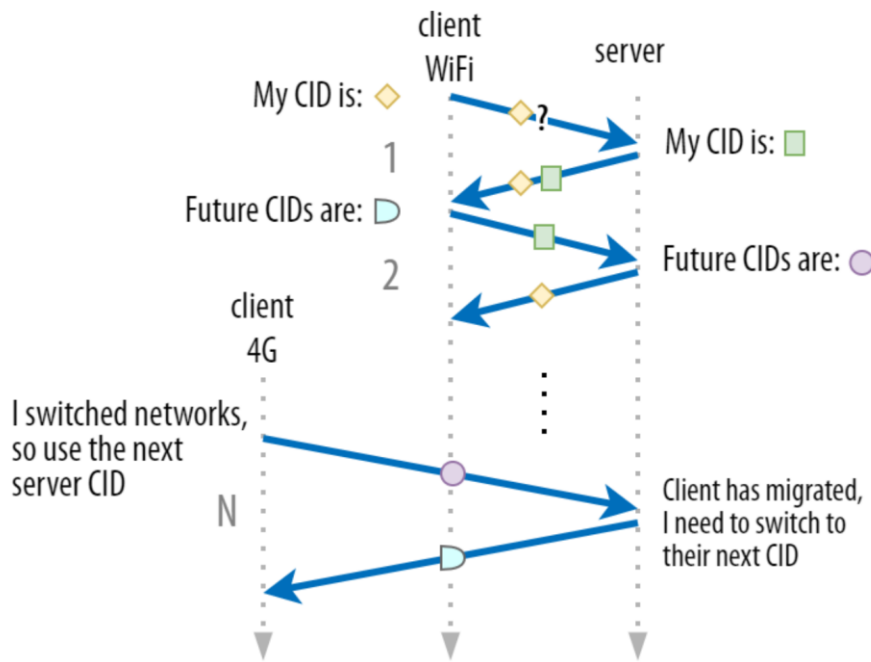
- 类似数据中心内存语义, QUIC可以充分利用通信中的并行性, 避免慢速请求阻塞其他请求。
- 通过调度, 可以缩短请求的平均完成时间。
- 当发生丢包时, 可以仅影响发生丢包的请求, 其他请求如果没有保序要求仍然可以快速完成。



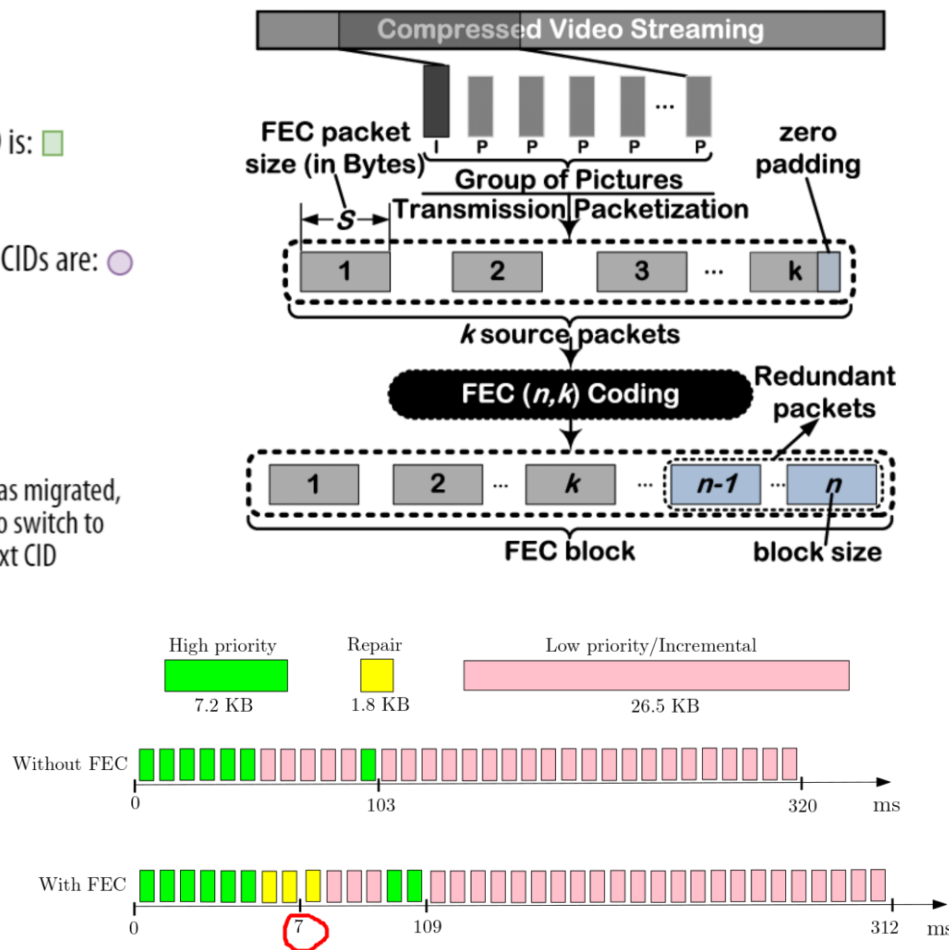


# QUIC改进通信协议栈，提升性能 (4)

## 连接迁移:



## 前向纠错 (FEC) :



HTTP Semantics		
HTTP 1.1	HTTP/2	HTTP/3
TLS/SSL (optional)	TLS 1.2+	TLS 1.3
TCP	TCP	QUIC
		UDP
IPv4 / IPv6		

## 精细流量控制:

- window\_update 帧, 告诉对端自己“可以且最多可以”接收的字节数绝对偏移量;
- blocked 帧, 告诉对端数据发送, 由于流量控制被阻塞, 暂时无法发送;
- stop\_waiting 帧, 用于通知对端, 不应该继续等待包小于特定值的包。
- 对于直播而言, 在直播发起时, 在转码跟不上的时候, 告诉服务端停止发送直播流或者做出一些应急处理, 推动定制的帧去做, 保证服务可用性。



# QUIC可改进的空间

- **拥塞控制：**

- **慢启动：** BBR的慢启动过程仍然需要多个RTT，但很多请求仅需传输少量数据，慢启动占了大部分甚至全部传输时间，需要更快速地探测到可用带宽；
- **突发流量：** 很多应用的流量不符合长稳流量模型，如果预留带宽则会影响带宽利用率，如果不预留则可能出现丢包或延迟骤增；
- **时延抖动：** 时延抖动会导致BBR的BDP计算不准确；

- **丢包重传：**

- **非随机丢包：** 前向纠错（FEC）对随机丢包较为友好，但一些场景下可能出现突发丢包；
- **伪丢包：** 时延抖动被当作是丢包，导致冗余的重传；
- **可靠传输：** 所有数据都必须到达，没有提供类似UDP的非可靠传输，一些实时数据事实上超时后没有必要重传；

# 目录

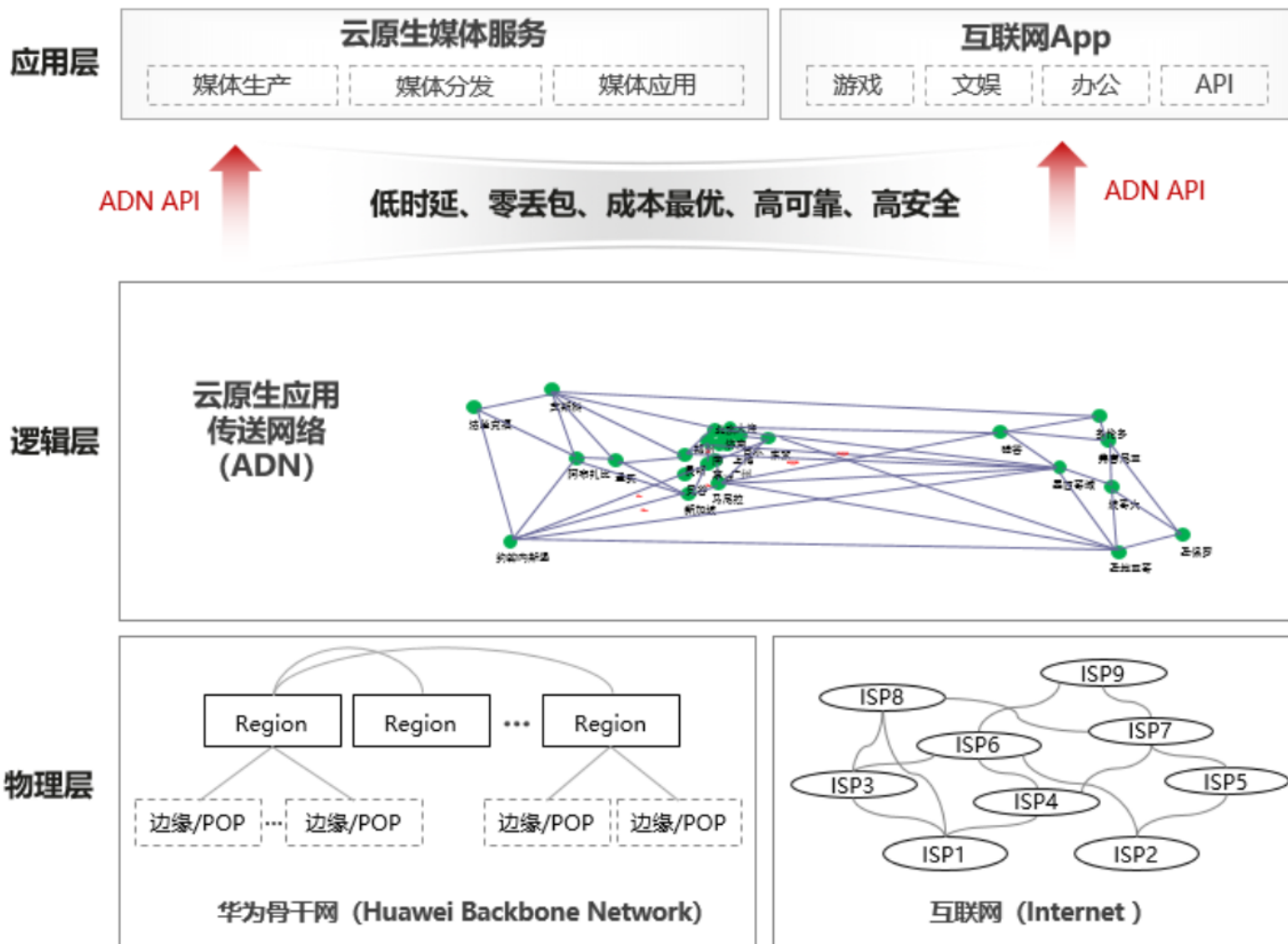
- 数据中心网络
  - 从Web服务到大规模异构并行计算
  - 编程抽象：从字节流到内存语义
  - 把数据中心作为一台计算机
- 广域网
  - 从“尽力而为”到“准确性”
  - **东数西算与全国一体化大数据中心**
- 无线网络
  - “1+8+N”与“5G to B”
  - 应用和数据无缝流转的“分布式超级终端”

# 东数西算：国家战略



- 数据+算力释放数据要素的生产力，成就数字经济底座
- 数据资源与算力资源分布不均衡，“东数西算”助力实现国家算力规模化和集约化
- 构建绿色算力途径，实现绿色能源与东西部的协调发展，是“东数西算”的战略考量
- “东数西算”带动西部数字经济，促进东西部协调发展

# 云原生应用传送网络 (ADN) : 应用驱动、软件定义、敏捷智能



## 网络拓扑: 广覆盖、高弹性、全互联

- 广覆盖: 全球互联互通, 2500+节点, 一跳入网
- 高敏捷: 分布式容器化部署, 分钟级节点增删
- 全互联: 逻辑/物理分层解耦, 基于Full Mesh点到点测量

## 路由与传输: 多目标驱动智能调度与控制

- 智能路由计算: 分钟级端到端路由图优化算法
- 多路径冗余并发: 单流分多流, 多流合单流
- 智能拥塞控制: 抗弱网协议增强、高可靠传输
- 高性能转发: DPDK、用户态驱动等转发模式

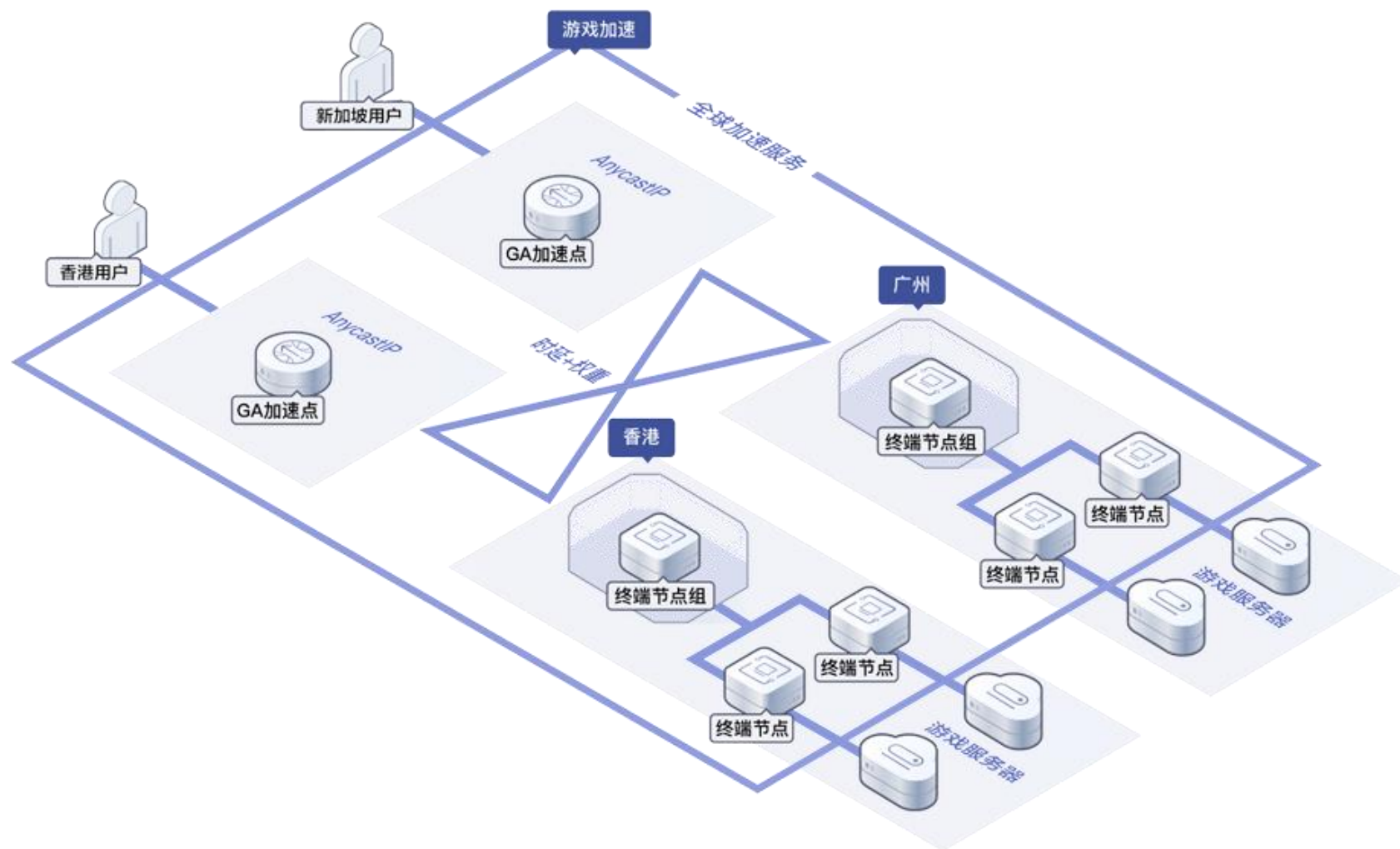
## 应用驱动: 软件定义SLA, 多协议

- 可编程SLA: 软件API定义的网络层/传输层/应用层QoS/SLA, 如时延、丢包、抖动、音视频MOS等
- 多协议灵活接入: TCP/UDP/域名DNS/SDK
- 应用感知: AI流量画像, 分时错峰, 端边云协调感知

# Regionless云计算：跨Region的全球部署、自动调度

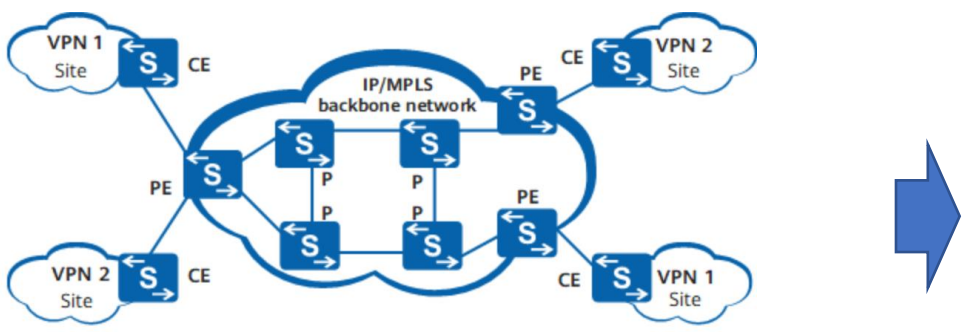
- 通常情况下，**用户购买资源前都是先选Region（地区）**，而他们对云服务的全球部署、网络拓扑的连接并没有整体概念，所以云厂商需要为用户揭开迷雾，将资源的分布、价格、使用现状一一呈现出来。
- 在架构设计上打破Region级服务的约束，引入全域调度能力，**基于对算力成本最优化、特定云服务及业务负载接入时延，以及应用/应用群之间的通信耦合关系，为用户提供最佳选择**。至于具体云服务的资源实例发放到哪一个地理区域，完全由云的智能调度系统动态确定。
- 这个过程就是Regionless化，由华为云来完成调度策略，屏蔽底层资源调度的复杂性。**用户无需自己选择地理Region，就能享受全局服务的全球部署能力**。
- 通过这种方式解决东西地区的平滑引流，使得用户在几乎无感知的情况下，将业务负载从东部城市平滑地迁移到西部，比如华为云的乌兰察布数据中心、贵安数据中心。其中涉及到地区层面的架构分层以及全域调度，乃至东部和西部资源的定价差别等等。

# 全球加速网络 (Overlay network)





# SD-WAN: 软件定义广域网



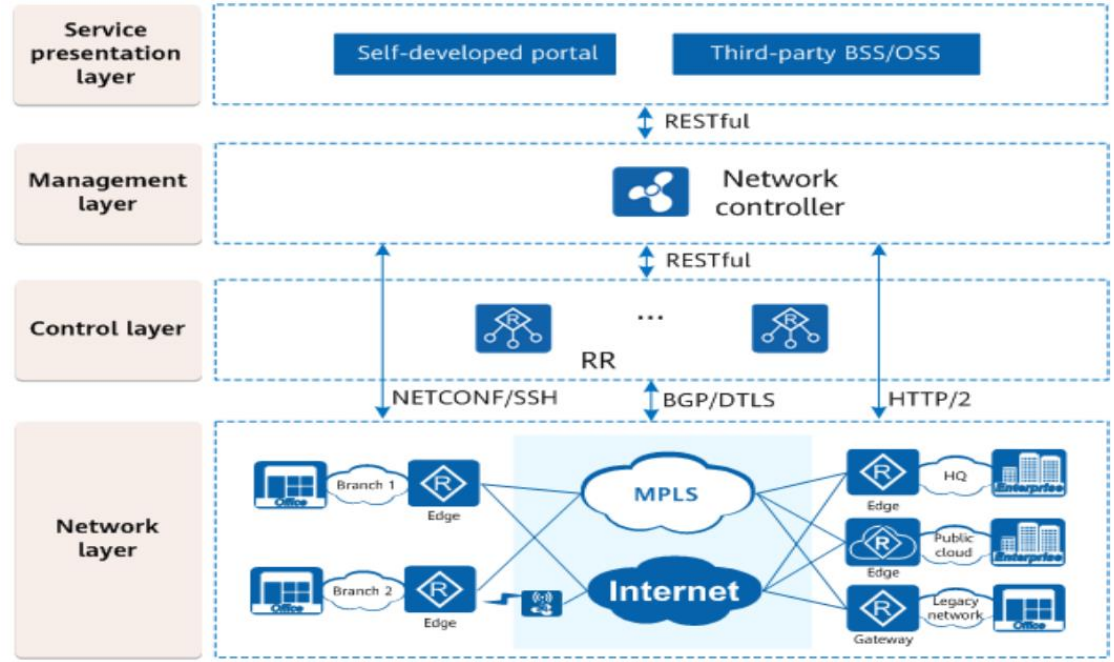
传统MPLS VPN的缺点:

- 成本昂贵, 例如10Mbps的跨国专线高达8万元/月
- 部署周期长, 故障排查难



SD-WAN: 软件统一管理网络设备

- 灵活的路由调度能力, 选择最佳路径
- 支持根据服务质量需求进行负载均衡
- 支持专线、公网等多种接口, 降低成本
- 部署、运维简单



来源: 艾瑞咨询研究院自主研究及绘制。



# 目录

- 数据中心网络
  - 从Web服务到大规模异构并行计算
  - 编程抽象：从字节流到内存语义
  - 把数据中心作为一台计算机
- 广域网
  - 从“尽力而为”到“准确性”
  - 东数西算与全国一体化大数据中心
- 无线网络
  - **“1+8+N”与“5G to B”**
  - 应用和数据无缝流转的“分布式超级终端”

# 1

# +

# 8

# +

# N



Smartphone

HUAWEI  
Share



AR/VR



Pad



PC



HD



Watch



Speakers



Vehicle  
Terminal



Earphone

4G/5G  
HiLink



CPE



Router



T-Box



Module



Intelligent  
Scale



Video



Game



Projector



Sweeping  
Machine



Camera



Vehicle  
Info.



Map



Sphygmomanometer



Printer



1=手机

8=PC、平板、TV、音响、眼镜、手表、车机、耳机

N=移动办公、智能家居、运动健康、影音娱乐、智能出行

# 5G在港口行业中的应用



- 时延要求高，10毫秒级
- 上行带宽要求大，传统5G下行比上行带宽高一个数量级

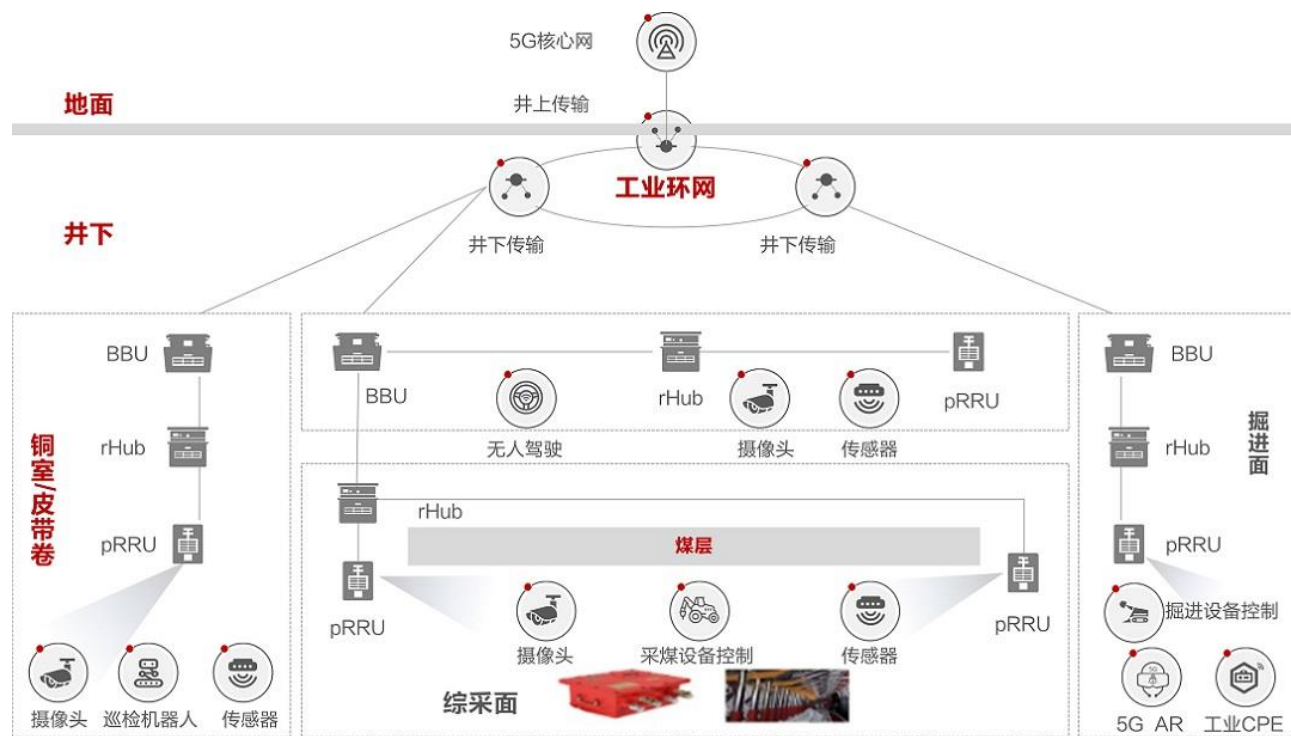




# 5G在矿山行业中的应用

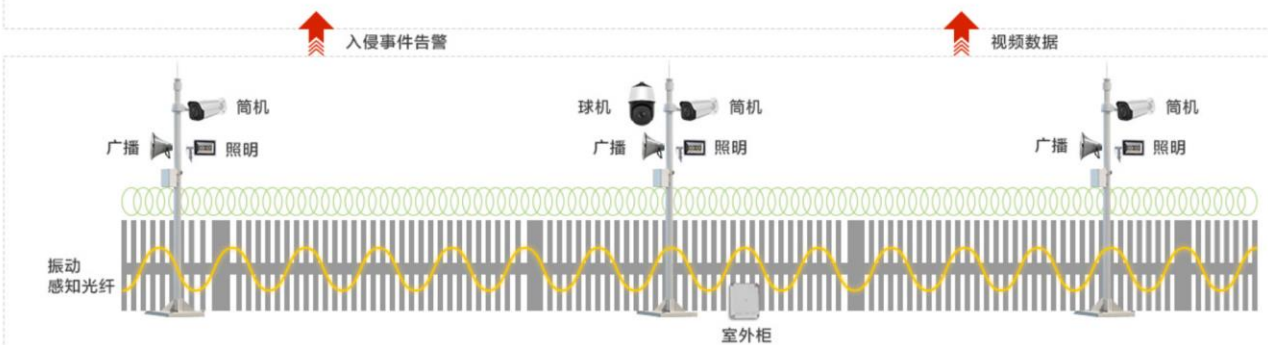


5G智慧矿山 – 无人开采车



## 5G煤矿 – 让矿工“穿西装打领带”远程操控作业

- 时延要求高，10毫秒级
- 上行带宽要求大，传统5G下行比上行带宽高一个数量级

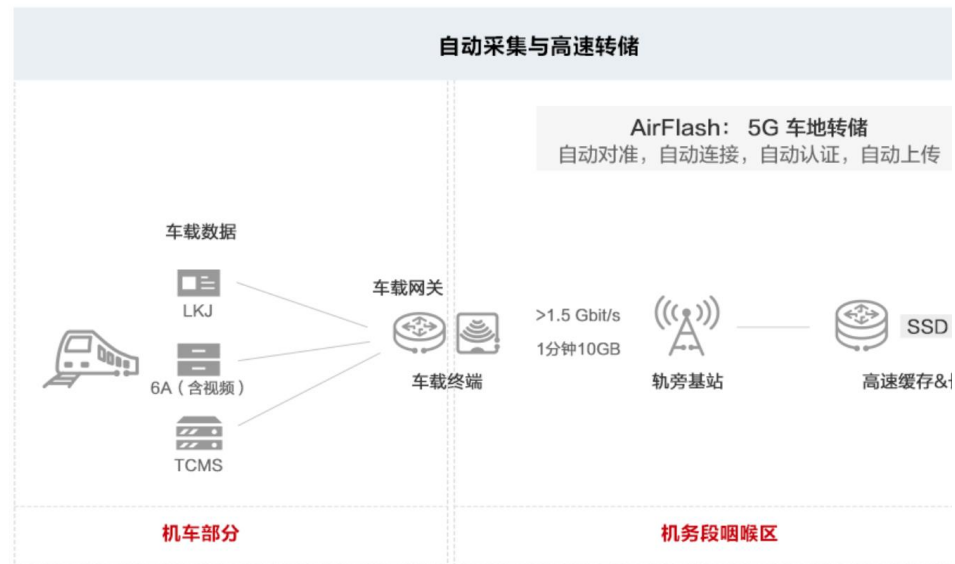


光感围界安防监控

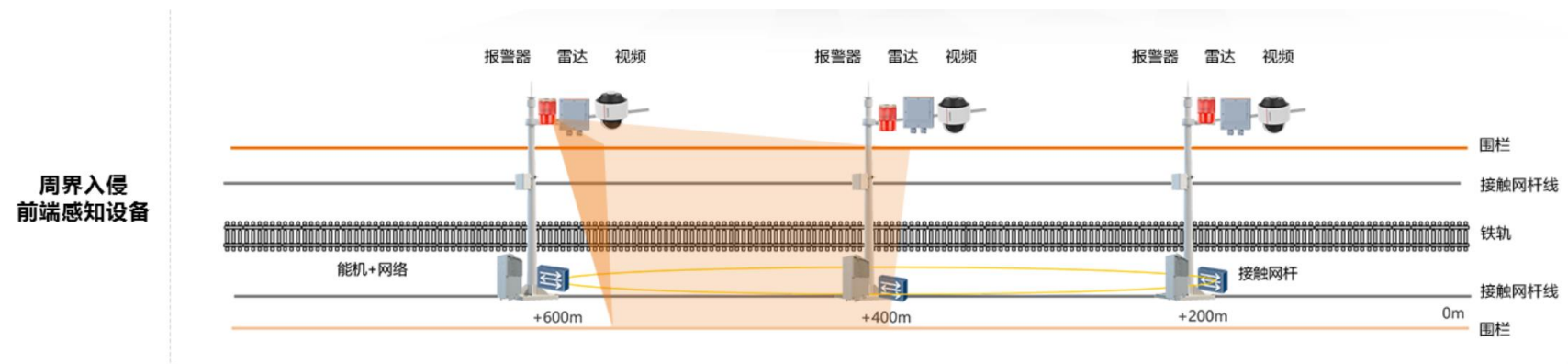
# 5G在铁路行业中的应用



- 时延要求高
- 上行带宽要求大
- 高速移动网络下对无线网络的稳定性要求高

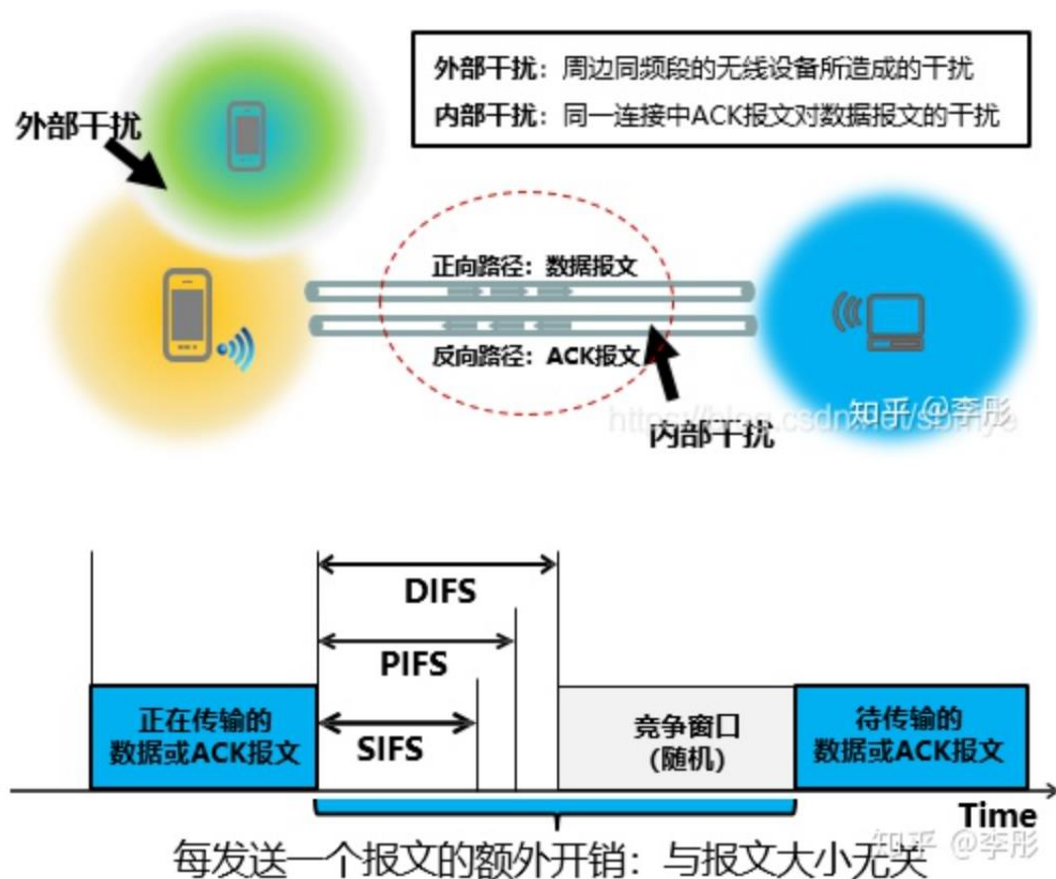


## 智慧机车



## 周界入侵检测

# 优化无线网络 (1): TACK

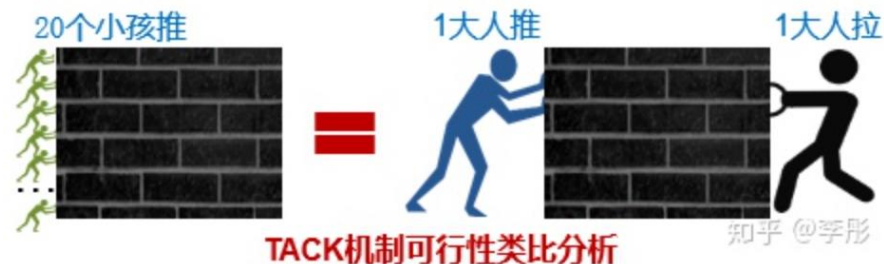


WiFi的半双工特性和冲突避免机制造成ACK报文与和数据报文形成了直接的资源竞争, 传统TCP的ACK几乎要占用将近一半的可用频谱资源

TACK:

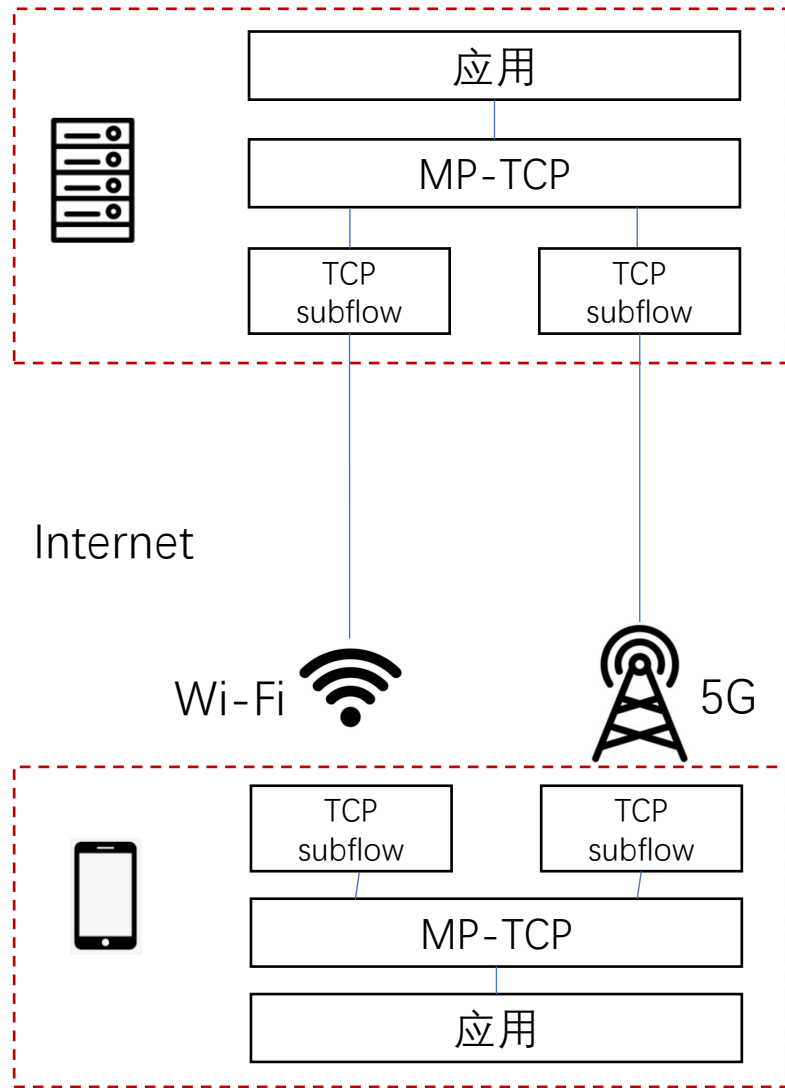
- 1) 更多ACK种类适应不同场景需求;
- 2) ACK按需携带更多必要的信息;
- 从而实现: 3) 更少但足够的ACK数目。

- “即时ACK”报文加快传输控制对即时事件（例如丢包）的反馈和响应;
- 高度自适应的“周期ACK”报文保证反馈的鲁棒性和可靠性, 以时钟周期触发的, 而非以包接收或者超时等事件触发, 也就不会造成“内部干扰”随着吞吐增大而增大。

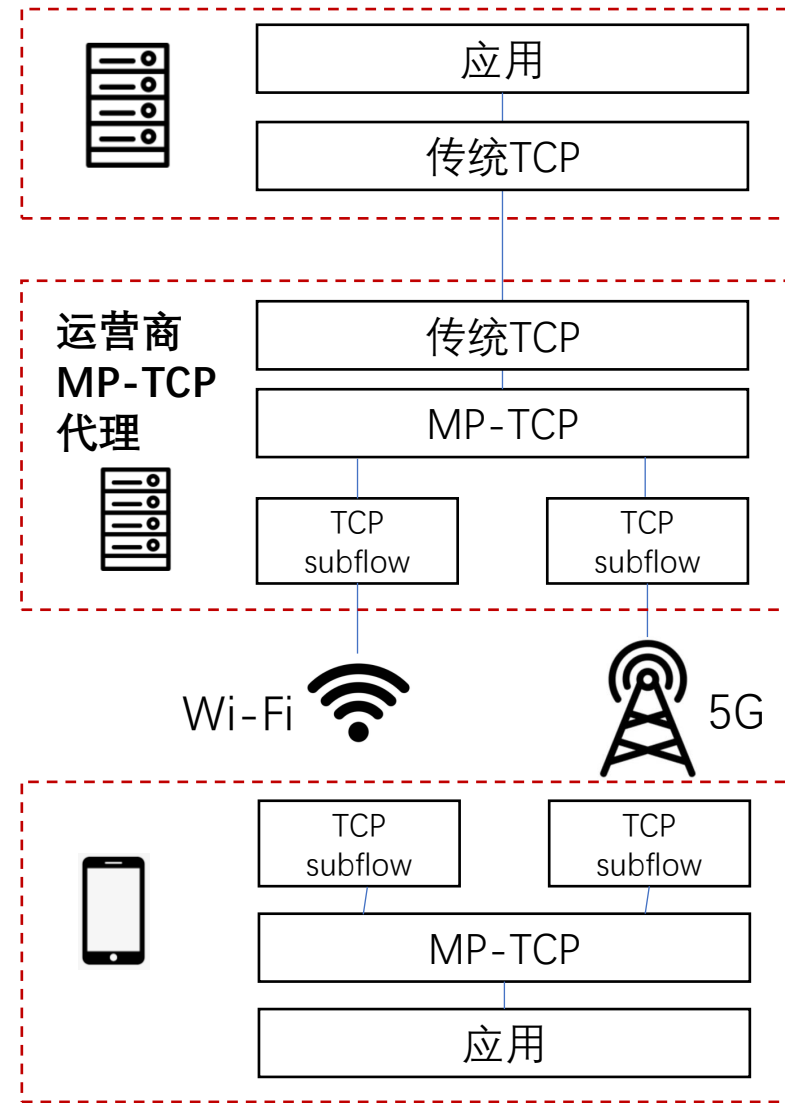


# 优化无线网络 (2): Link Turbo

部署场景1: 任意网络+特定应用



部署场景2: 特定网络+任意应用





# 无线网络性能达到甚至超越有线网络

Channel Bandwidth	Number of Spatial Streams	Maximum Speed		Target Device or Application
		Wi-Fi 5	Wi-Fi 6/6E	
First Wave Products				
80 MHz	1	433 Mb/s	540 Mb/s	Dual-band smart phone, VoIP handset, or tablet
80 MHz	3	1.3 Gb/s	1.6 Gb/s	High-end laptop/ digital cinematography
Second Wave Products				
80 MHz	2	867 Mb/s	1.1 Gb/s	Netbook/low-end laptop
160 MHz	3	2.6 Gb/s	3.6 Gb/s	High-end laptop/digital cinematography
Possible Future Implementations				
160 MHz	4	3.5 Gb/s	4.8 Gb/s	Outdoor or low coverage areas
160 MHz	8	n/a	9.61 Gb/s	Specialized

## Wi-Fi 5/6的理论性能

- 5G全国平均下行334.98Mbps, 上行70.21 Mbps
- 4G全国平均下行39.02Mbps, 上行21.63 Mbps
- 固定宽带用户平均下行94.01Mbps, 上行38.68Mbps

5G/4G数据: 《全国移动网络质量监测报告》2022年第一季度;

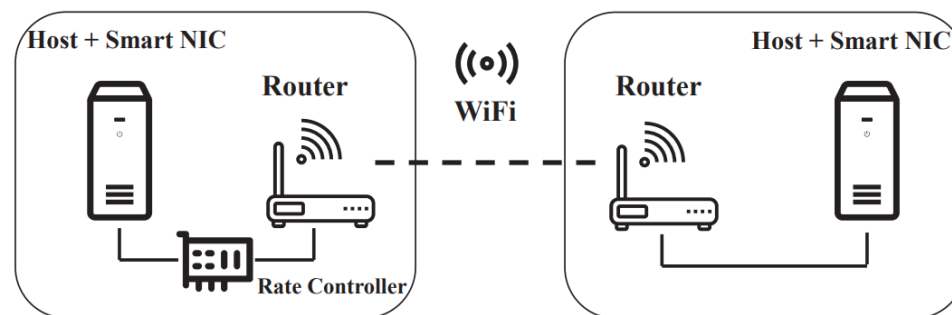
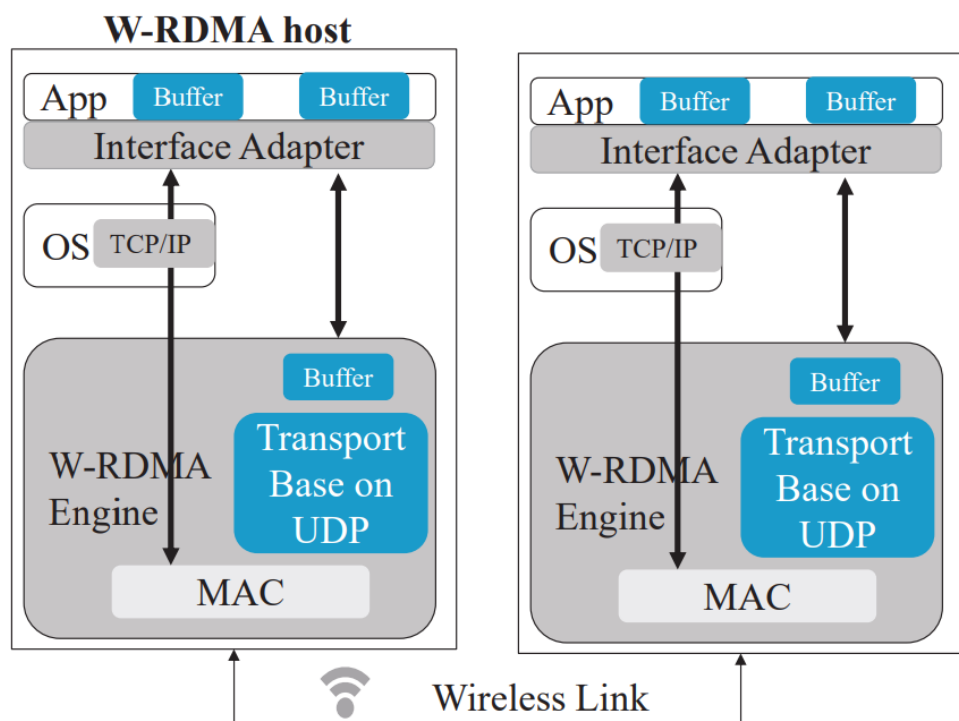
固定宽带数据: <https://www.speedtest.cn/article/VXjrPAKZ3Vx8wMILBpW>

Key performance indicators	5G parameter values	6G parameter values
Maximum transmission rate	20/(DL)/10/(UL)	>1Tbit/s
Experience rate	0.1-1	10-100/
Flow density	10 Tb/s/km <sup>2</sup>	10 <sup>2</sup> -10 <sup>4</sup> Tb/s/km <sup>2</sup>
Positioning accuracy	1 m (indoor) 10 m (outdoor)	0.1 m (indoor) 1 m (outdoor)
End-to-end delay	1 ms	< 0.1 ms
Error rate	500 km/h	1000 km/h
Mobility	450MHz-6GHz(FR1)	95 GHz-3 THz
Spectrum bandwidth	FR1:100MHz FR2:400MHz	20 GHz
Spectrum efficiency	30-100bps/Hz	200-500 b/s/Hz
Connection density	10 <sup>6</sup> /Km <sup>2</sup>	10 <sup>8</sup> /km <sup>2</sup>
Network energy consumption	100 bits/J	200 b/J
Base station computing capacity	100-200T ops (operation per second)	1000T ops

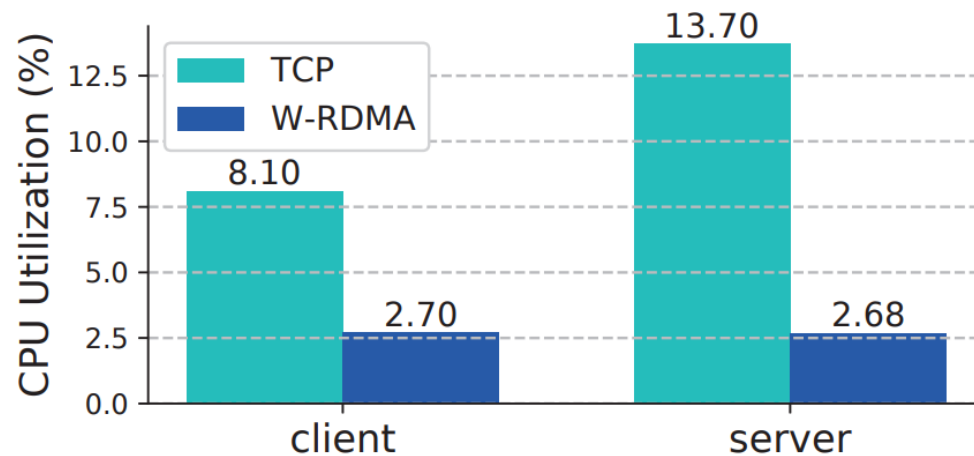
## 5G/6G的理论性能

# 无线内存语义：将内存语义从数据中心扩展到终端

是否有可能让无线网卡支持内存语义（消息语义、远程内存访问），从而减少软件协议栈的开销？



实验环境



显著降低终端功耗

# 目录

- 数据中心网络
  - 从Web服务到大规模异构并行计算
  - 编程抽象：从字节流到内存语义
  - 把数据中心作为一台计算机
- 广域网
  - 从“尽力而为”到“准确性”
  - 东数西算与全国一体化大数据中心
- 无线网络
  - “1+8+N”与“5G to B”
  - 应用和数据无缝流转的“分布式超级终端”

# 鸿蒙分布式软总线



## 挑战：

- 设备数量众多；
- 设备间的连接方式复杂，并且由于房屋结构的关系，无线信号干扰、衰减、遮掩的问题也较多；
- 使得设备之间的互联互通更加可靠、安全；
- 基于业务和网络状态进行质量优化和合理调度。

# 鸿蒙分布式软总线



## 发现

- 支持通过Wi-Fi、蓝牙、以太网等不同的媒介发现设备。
- 支持根据不同设备的能力，选择合适的发现媒介。
- 支持根据设备特点和业务需求提供合适的发现频次、扫描周期等发现策略。

## 连接

- 通过分布式软总线的连接技术，连接周边的分布式设备。
- 分布式软总线根据分布式设备的能力和业务需求，选择合适的通信媒介和最恰当的连接技术，建立通信链路，为后续的组网和传输提供基础能力。

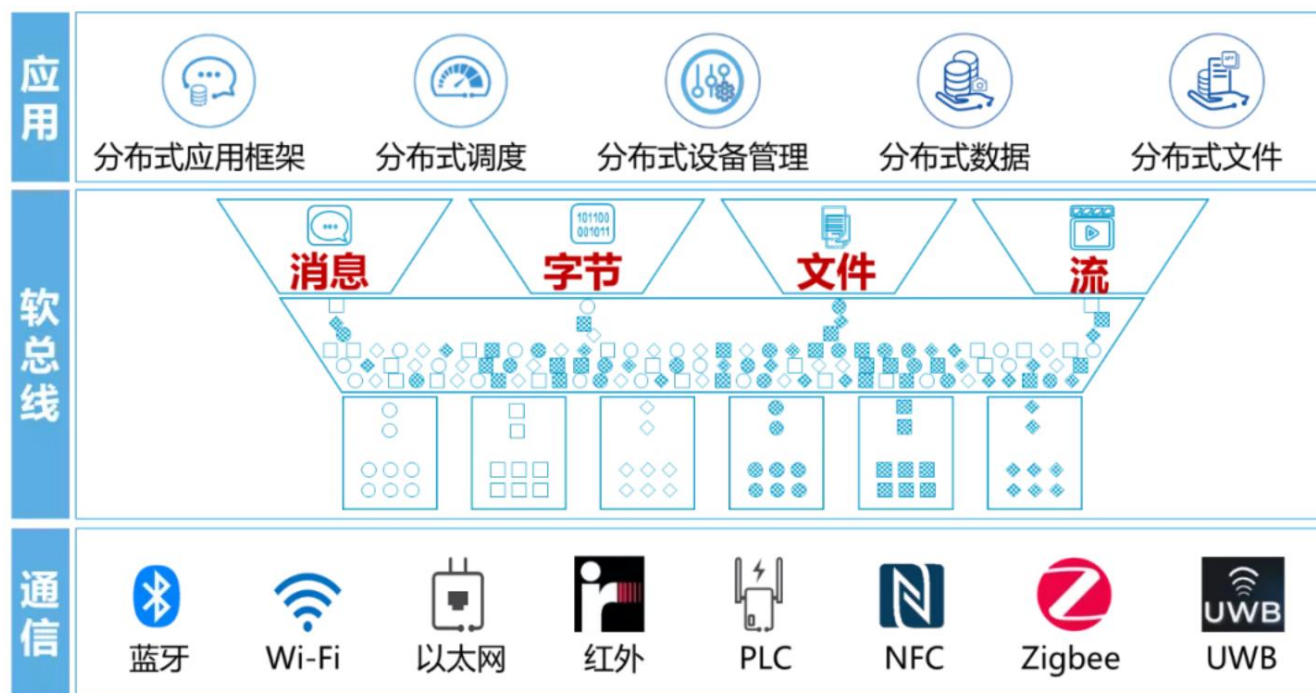
## 组网

- 将不同能力、不同特征的分布式设备组成一张网络，组成一张动态网络。
- 管理每个设备的通信能力、业务能力。
- 当业务需要时，通过分布式软总线的网络，可以随时提供业务需要的设备能力信息，支撑建立业务通道。

## 传输

- 对业务数据和QoS要求进行抽象，并根据网络负载和设备能力为业务提供合适的传输技术。
- 既保证单业务的通信诉求，又保证整个分布式网络内多业务的传输质量。

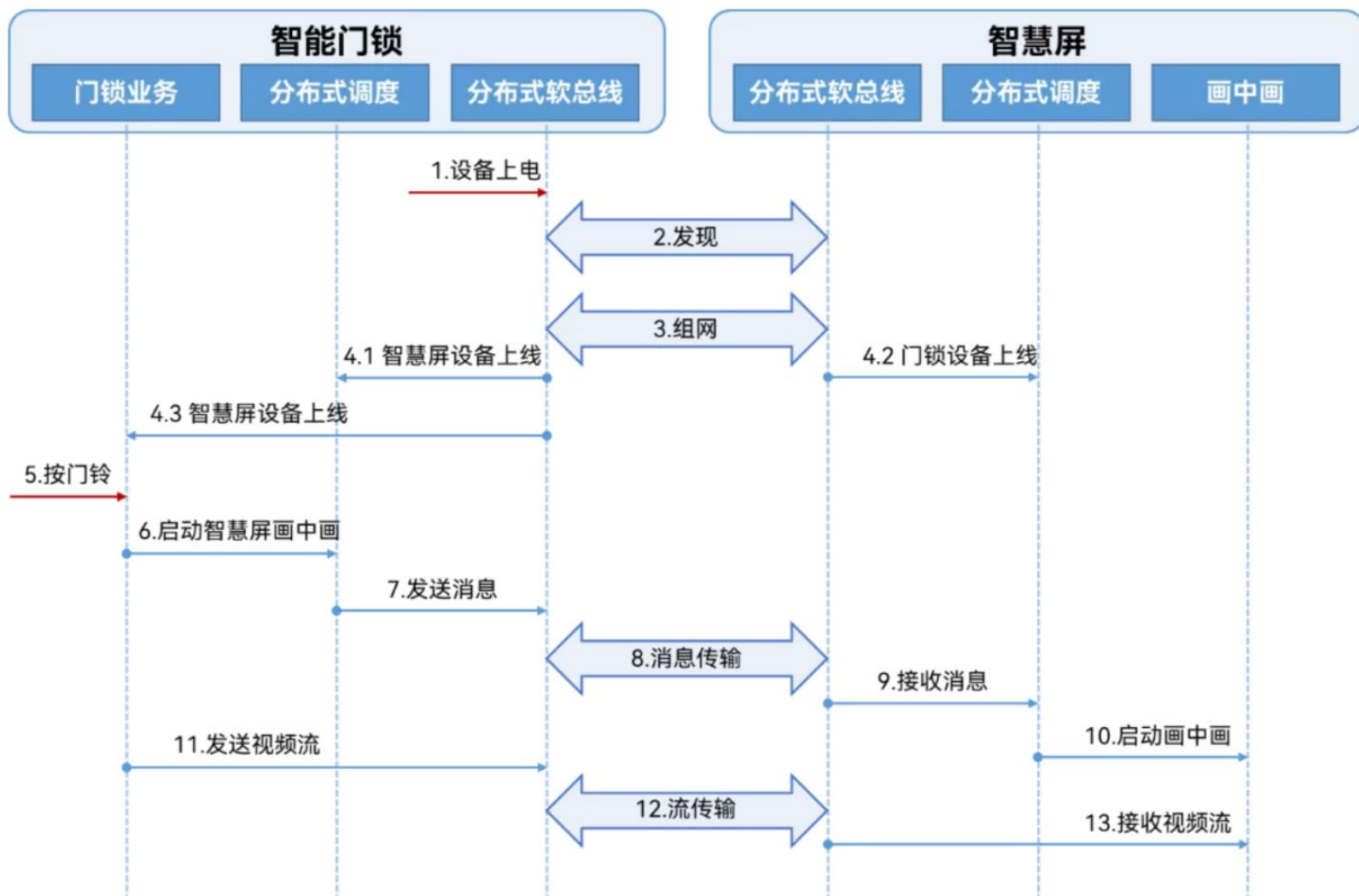
# 鸿蒙分布式软总线的通信原语



- **消息:** 用于实时性和可靠性要求极高短数据（比如控制类指令）的传输。
- **字节:** 用于时延要求不高的基本业务数据传输。
- **文件:** 主要用于设备间文件的传输和同步。通常要求较大的传输带宽, 但实时性要求不高。
- **流:** 一般用于音视频流的传输。既要求高带宽, 又要求低时延。



# 鸿蒙分布式软总线的应用场景示例



**场景描述：智慧屏的画中画用作猫眼。**当门铃按下时，门锁发现支持画中画功能的电视，建立门锁上摄像头到电视屏幕中画中画的高速传输通道。

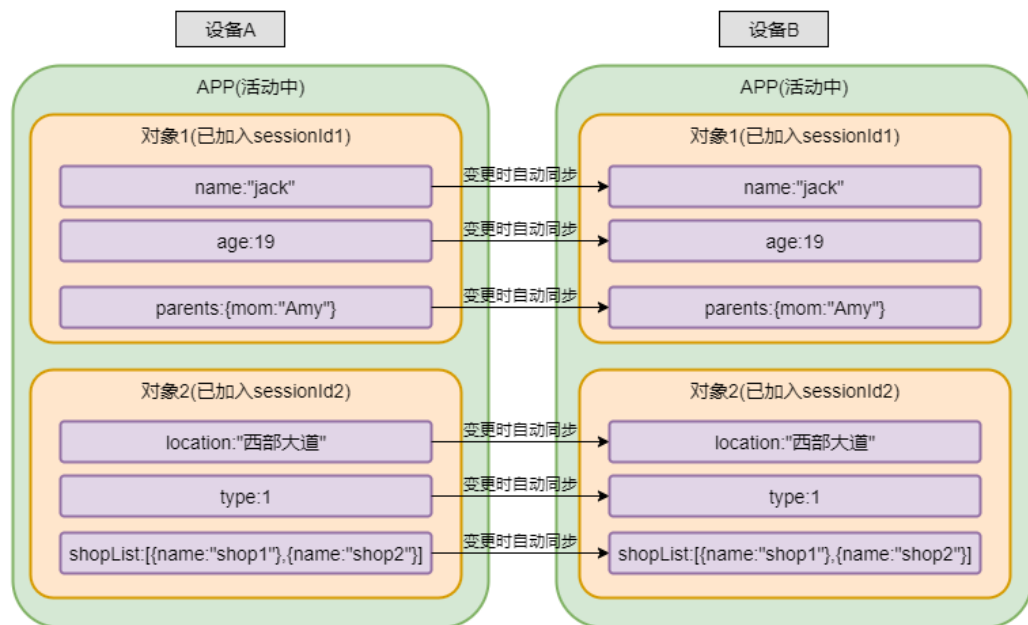
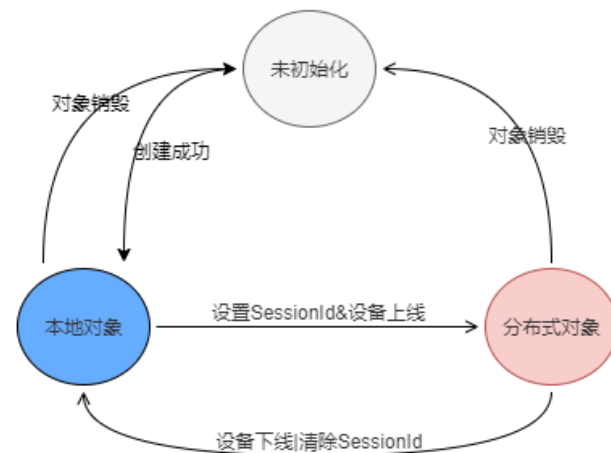
- **步骤1：**智能门锁上电后，分布式软总线启动发现流程。
- **步骤2、3：**分布式软总线发现智慧屏设备后，启动组网流程，完成智能门锁与智慧屏之间的可信认证。
- **步骤4.1、4.2、4.3：**分布式软总线分别向智能门锁和智慧屏上报对方设备上线。
- **步骤5、6：**当客人按下门铃时，智能门锁的门锁业务请求分布式调度启动智慧屏画中画。
- **步骤7：**智能门锁的分布式调度将“启动画中画”的指令封装为消息，请求分布式软总线将该消息发送至智慧屏的分布式调度。
- **步骤8、9：**分布式软总线通过消息传输功能将“启动画中画”指令发送到智慧屏的分布式调度。
- **步骤10：**智慧屏的分布式调度收到“启动画中画”指令后，启动画中画FA。
- **步骤11：**智慧门锁的门锁业务请求分布式软总线将捕获的摄像头画面传输至智慧屏画中画。
- **步骤12、13：**分布式软总线通过流传输功能，将门锁侧摄像头画面发送至智慧屏，智慧屏的画中画收到门锁摄像头画面后，在画中画FA中播放。



# 分布式对象：鸿蒙分布式超级终端的“全局变量”



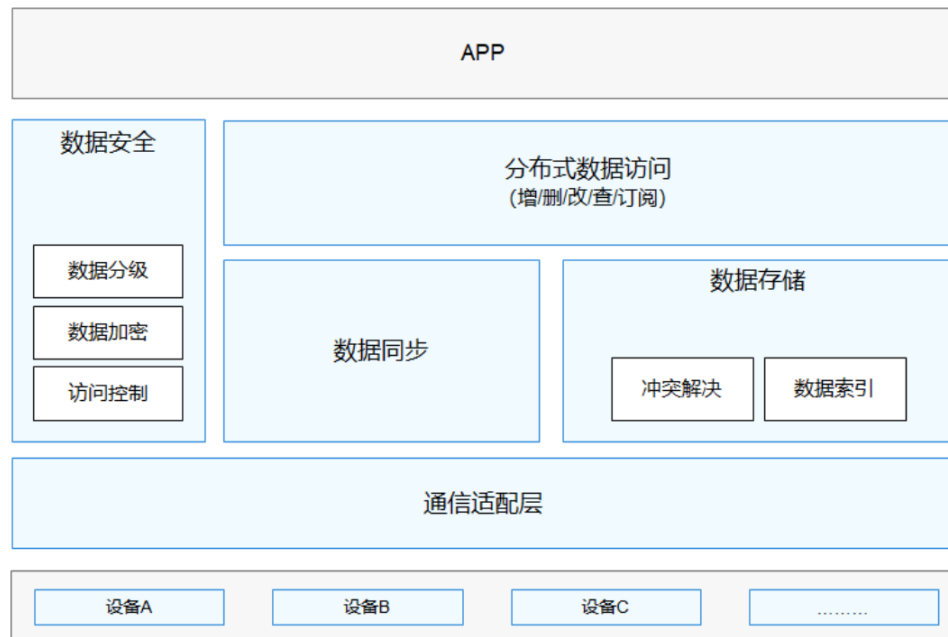
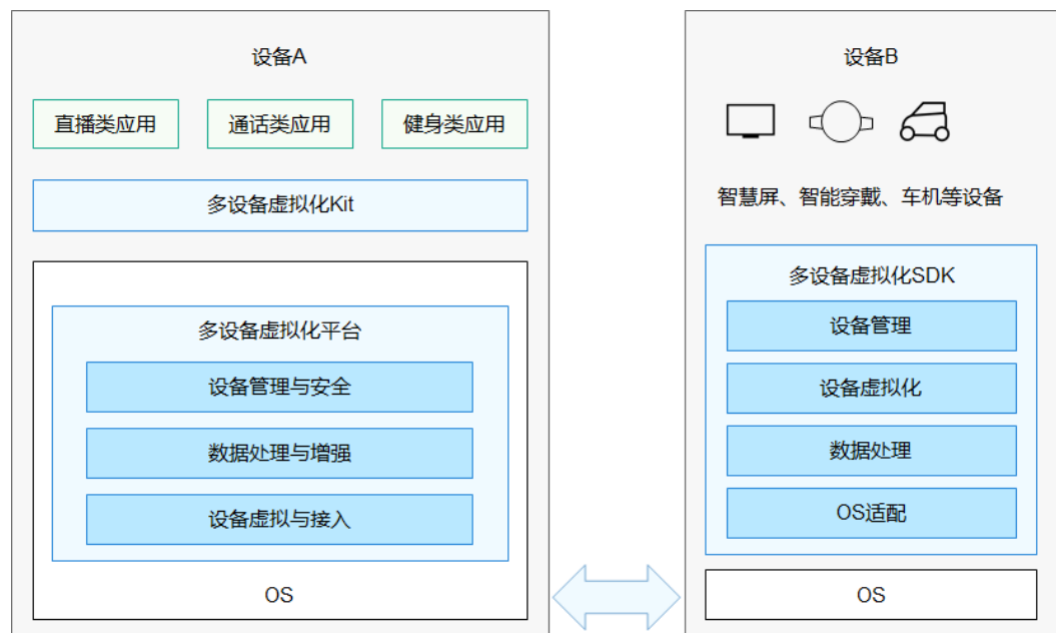
在 TV/PAD 上观看视频时，在可触屏设备上控制 TV/PAD 上的播放状态、进度、音量和速度等，实现多端控制



加入了同一个 session 的对象，支持以下操作：

- 读取/修改 session 中的数据；
- 监听数据变更，感知其他对象对共享对象数据的修改；
- 监听状态变更，感知其他对象的加入和离开。

# 鸿蒙分布式超级虚拟终端 (1)



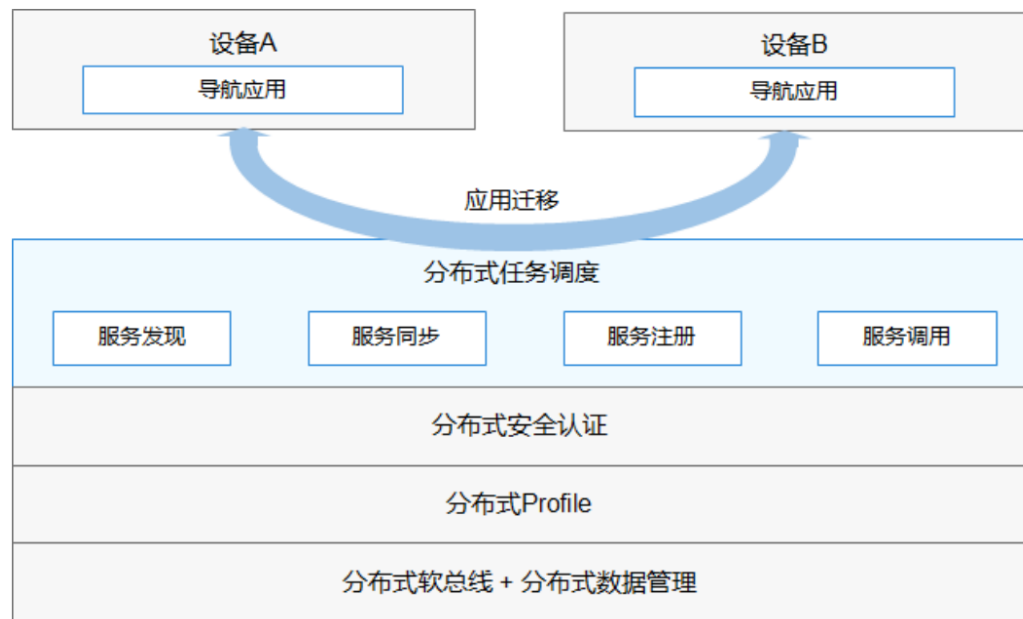
## 分布式设备虚拟化

- **视频通话场景**：在做家务时接听视频电话，可以将手机与智慧屏连接，并将智慧屏的屏幕、摄像头与音箱虚拟化为本地资源，替代手机自身的屏幕、摄像头、听筒与扬声器，实现一边做家务、一边通过智慧屏和音箱来视频通话。
- **游戏场景**：在智慧屏上玩游戏时，可以将手机虚拟化为遥控器，借手机的重力传感器、加速度传感器、触控能力，为玩家提供更便捷、更流畅的游戏体验。

## 分布式数据管理

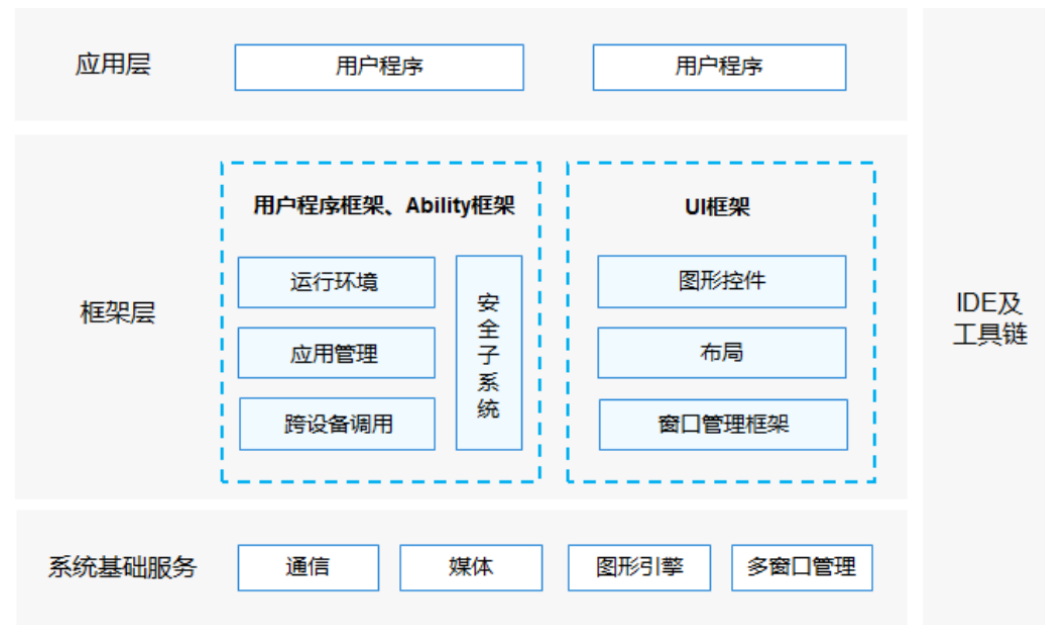
- **协同办公场景**：将手机上的文档投屏到智慧屏，在智慧屏上对文档执行翻页、缩放、涂鸦等操作，文档的最新状态可以在手机上同步显示。
- **照片分享场景**：出游时，使用手机拍摄的照片，可以在登录了同帐号的其他设备，比如平板上更方便地浏览、收藏、保存或编辑，也可以通过家中的智慧屏上同家人一起分享记录下的快乐瞬间。

# 鸿蒙分布式超级虚拟终端 (2)



## 分布式任务调度

- **导航场景**：如果用户驾车出行，上车前，在手机上规划好导航路线；上车后，导航自动迁移到车机和车载音箱；下车后，导航自动迁移回手机。如果用户骑车出行，在手机上规划好导航路线，骑行时手表可以接续导航。
- **外卖场景**：在手机上点外卖后，可以将订单信息迁移到手表上，随时查看外卖的配送状态。



## 一次开发，多端部署

- **UI框架**：支持使用ArkTS (TypeScript)、JavaScript、Java语言进行开发，并提供了丰富的多态控件，可以在手机、平板、智能穿戴、智慧屏、车机上显示不同的UI效果。采用业界主流设计方式，提供多种响应式布局方案，支持栅格化布局，满足不同屏幕的界面适配能力。
- **按需裁剪**：通过组件化和小型化等设计方法，支持多种终端设备按需弹性部署，能够适配不同类别的硬件资源和功能需求。支撑通过编译链关系去自动生成组件化的依赖关系。

# 总结

- 数据中心网络

- 大数据、大模型下，分布式异构并行计算是必由之路
- 从字节流到内存语义：消息语义、异步和同步远端内存访问、RPC
- 把数据中心作为一台计算机：高速互联+Serverless便捷编程

- 广域网

- 实时互动应用：QUIC传输协议+Overlay全球加速网络
- 东数西算：Regionless使能全国一体化大数据中心

- 无线网络

- “1+8+N”与“5G to B”：无线网络优化、无线内存语义
- 鸿蒙分布式超级终端：分布式软总线、设备虚拟化、数据管理、任务调度

A large steel truss bridge spans across a body of water at night. The bridge's structure is illuminated from below, and its reflection is visible in the water. In the background, the Oriental Pearl Tower is lit up, and other city lights are visible against a dark blue sky.

# 通途协议栈： 连接算力孤岛，天堑变通途

Thanks!