摘要：

- 多样化的移动计算设备（智能手机、平板电脑、可穿戴设备）可以适应不同的应用场景，但在性能和可用性方面存在各种限制。
- 可以将设备整合并互联形成个人移动云，通过协作和资源共享使这些设备相互补充。
- 但由于移动设备在硬件和软件方面的异质性，这一方案颇具挑战。
- 基本思路是利用现有的移动操作系统服务作为资源共享的接口，来掩盖移动系统中的硬件和软件异质性，并将资源共享框架进一步发展为移动操作系统中的中间件。

1介绍

背景：
移动用户通常配备多种类型的移动计算设备满足了不同应用场景的独特需求，但也限制了这些移动设备在其他方面的性能或可用性。如可穿戴设备以小巧的外形实现了身体感知，但在计算、通信和电池寿命方面能力有限。可行方案是构建一个个人移动云，通过无线链路将用户拥有的所有移动设备整合并互联。可穿戴设备和智能手机可以通过利用附近更强大设备（如智能汽车）的计算能力或GPS读数极大地延长其本地电池寿命。另一方面，智能汽车也可以利用可穿戴设备上各种身体传感器的数据来推断用户的实时行为模式，并相应地做出反应。

挑战：
主要在移动计算设备的异构性，这种异构性体现在硬件和软件两个方面。

- 当今移动设备上搭载的硬件组件种类日益增多，导致这些硬件所使用的驱动程序、I/O堆栈和数据访问接口存在根本性差异。
- 当今移动应用程序的复杂性大幅增加，导致它们对移动系统资源的需求类型及其访问这些资源的具体方式存在异构性。

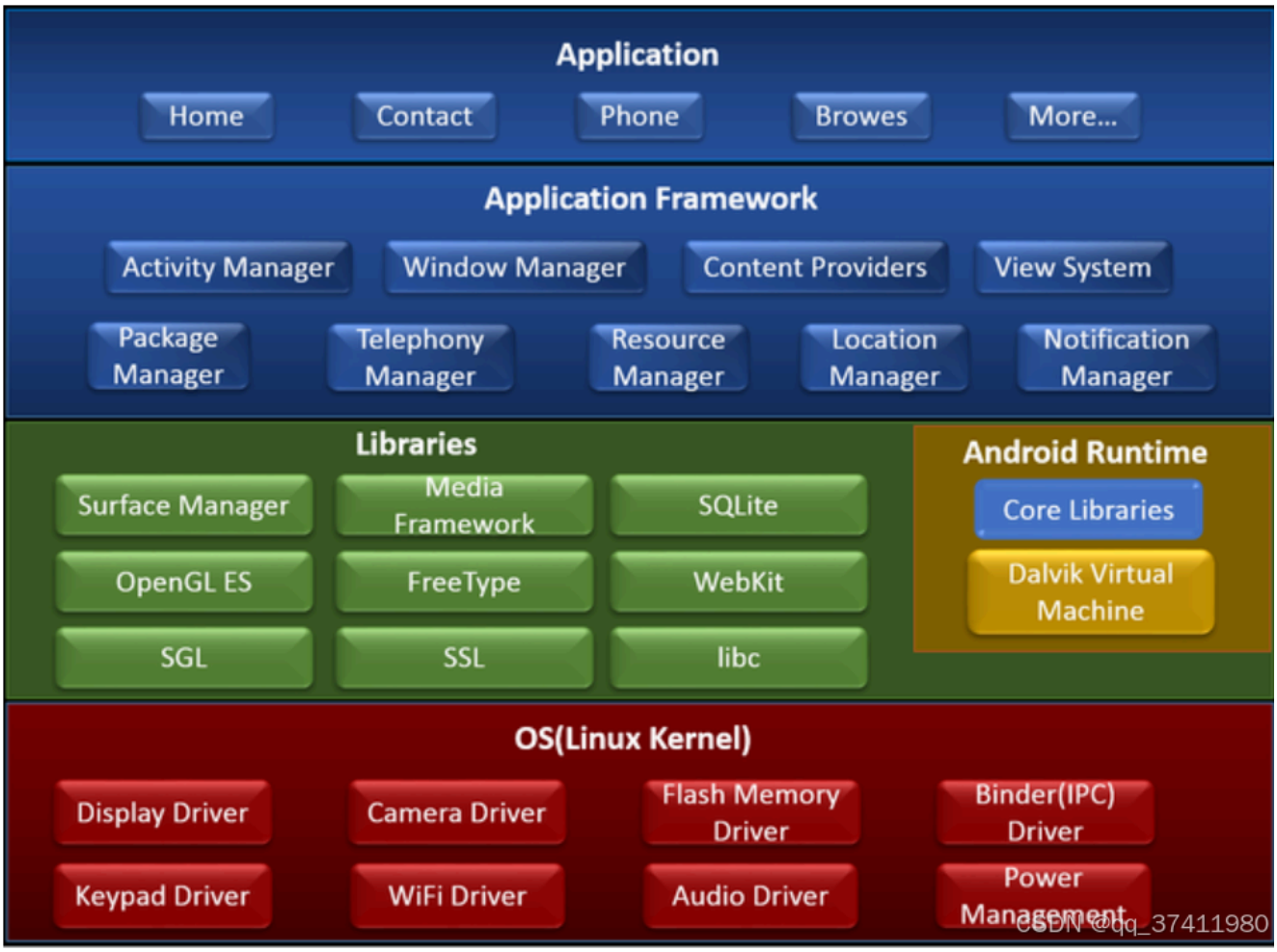
解决异构性现有方案：**仅限于针对单个移动应用程序或特定类型的共享硬件来互联移动设备**。因此，它们需要大量重编程工作来互联异构移动设备，每次都需要为这些设备的每个硬件或软件组件重新设计。这种重编程工作不仅降低了移动计算系统在多样化环境中的可用性，还增加了移动操作系统的额外开销，从而降低了移动系统性能。

解决上述问题关键和挑战：
实现资源共享框架，适当地掩盖移动系统中硬件和软件的异构性。在移动操作系统层次结构的较低层，框架必须处理以本质上不同方式操作的各个硬件驱动程序，因此需要大量的重新工程工作。另一方面，在应用程序层共享系统资源能够通过通用操作系统接口访问移动硬件，但必须与特定的数据传输协议相关联，因此通用性有限。

本文方法：
设计一种移动系统框架，将资源共享框架作为移动操作系统中的中间件进行开发，该中间件利用现有的移动操作系统实现资源共享，为用户应用程序提供统一的数据访问API。移动设备之间的互连可以通过远程访问和调用这些操作系统服务来实现。由于这些服务作为操作系统内核的独立系统进程执行，并与应用程序进程分离，因此可以通过移动系统之间的进程间通信（IPC）实现远程服务调用，而无需涉及复杂的内存引用和同步问题。因此，任何新设备都可以通过将其操作系统插入我们的框架中来加入移动云，而无需修改操作系统内核、我们的框架本身或任何移动应用程序的源代码。

2概述

移动操作系统的分层架构：



移动操作系统通常将用户应用程序与底层系统实现隔离开，使得系统资源管理更加高效，保护移动系统免受因设计不当的应用程序或移动恶意软件的恶意攻击而导致的资源耗尽。移动应用程序不直接访问系统资源，相反，资源访问由系统服务通过一组通用且预定义的API提供，这些API分别通过Android中的Binder机制（从IPC（进程间通信）角度来说，Binder是Android中的一种跨进程通信方式）和iOS中的消息传递机制由用户应用程序调用。例如，在Android和

iOS中，应用程序不是直接访问GPS或WiFi网络接口，而是调用操作系统提供的位置服务获取设备的位置信息。

现有远程资源共享方案的不足：
1.重复的编程工作
现有方案通过移动应用程序二进制文件或操作系统底层驱动程序实现，因此无法在移动系统中提供通用且自适应的资源共享（1中解决异构的现有方案？）。
另外在操作系统驱动层级进行资源共享需要大量的重新编程工作，如虽然移动操作系统提供了一个统一的系统服务来访问所有板载传感器，但现有的操作系统级资源共享方案未能利用这种统一接口，必须为每个驱动程序编写共享功能。
此外也可能是由于定制的SoC设计（系统级芯片设计）以及制造商之间硬件驱动程序的不兼容性所导致的。
2.操作系统驱动底层忽略了移动应用程序的执行模式
例如，每个用户应用程序通常会指定其请求位置数据的间隔，尽管Android中的GPS设备每秒都会报告位置信息。因此，为远程的Google GMS和Facebook应用程序共享GPS设备将分别浪费80%和98%的数据传输，因为它们分别每5秒（4/5浪费）和60秒（59/60浪费）请求一次GPS数据。
底层驱动操作在硬件状态发生变化时同步移动系统之间的资源数据，但这种变化频率（1s一次）通常远高于应用程序实际资源请求的频率（5s或60s一次）。

文章方案：
由于系统服务是用户应用程序访问系统资源的唯一接口，因此只要该框架能够拦截用户应用程序的资源访问请求并将其重定向到远程系统，就可以通过相同的通用框架提供对远程系统上任何类型资源的访问（如下图）。
此外，不同类型的系统服务遵循相同的调用机制（例如，Android 中的 binder 和 iOS 中的消息传递），因此我们不会遇到服务操作的异构性。
其次，这些系统服务对用户应用程序隐藏了硬件操作的细节。更重要的是，用户应用程序将以与访问本地资源相同的方式访问远程系统资源，这种透明性因此使得大量现有的移动应用程序能够访问远程资源而无需任何重新编程工作。

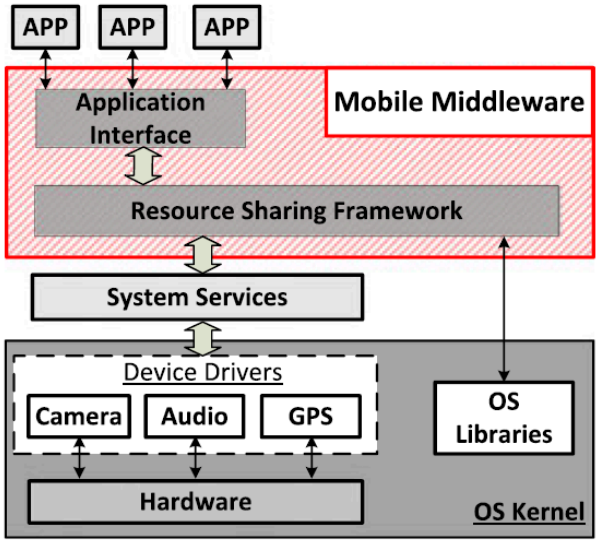


Fig. 2. The big picture of interconnecting heterogeneous mobile devices.

3 资源共享框架

资源共享框架设计：

框架拦截本地移动应用程序生成的资源访问请求，并将这些请求转发到作为服务器并提供共享资源的远程移动系统。

访问可分为主动调用和被动回调：

主动调用：框架创建一个本地服务代理对象，当应用程序请求访问远程服务时，该代理对象会触发一个远程调用事件，该事件在服务器端被捕获以调用相应的服务方法。

被动回调：允许应用程序注册并监听系统事件，并使用回调句柄，如位置更新。当系统事件发生时，服务器端的服务通过回调代理调用该句柄，然后用于将资源数据传输回客户端。

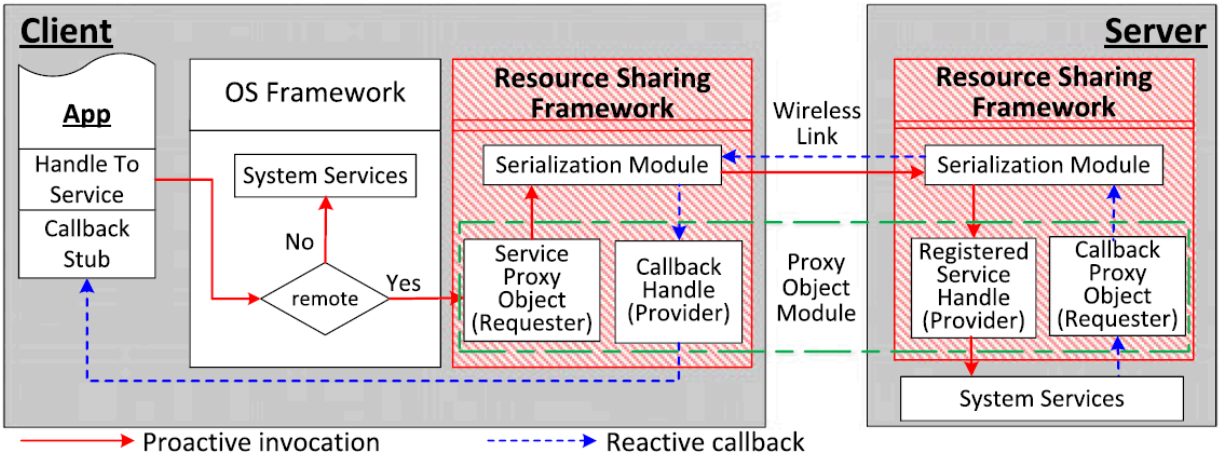


Fig. 3. Design of resource sharing framework.

3.1远程系统服务的调用

3.1.1基于java的系统服务:

一方面，为了设计一个通用的序列化模块来共享这些服务，我们利用 **Java 反射机制**（Java 反射机制是在运行状态中，对于任意一个实体类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能调用它的任意方法和属性）来最小化编程工作量，这使得开发者能够在运行时检查类、接口、字段和方法，而无需提前知道类和方法的名称。具体来说，我们通过反射访问任何内存对象的所有字段值，并将它们转换为二进制数据。类似地，这个特性也可以用于反序列化，通过在运行时创建对象实例并设置所有字段值，从而从接收到的二进制数据中重建内存对象。序列化和反序列化需要遍历内存对象，使用广度优先搜索遍历对象树来遍历和重建内存对象的内容。此外，当我们遍历内存对象的字段时，如果字段是基本类型，我们就获取或设置其值。否则，字段是对象引用类型，我们在处理完其所有兄弟节点后递归遍历该字段的内容。
另一方面，代理对象模块最直接的解决方案是使用 Java 的内置**动态代理**（在程序运行时动态生成一个代理对象，拦截方法调用，干额外工作）来实现一个接口，从而利用现有的 Java 反射机制来实例化代理对象，但 Android 中的用户应用程序必须通过 Android IPC binder 机制访问系统服务，其中用户应用程序和系统服务分别作为 binder 代理和 binder 存根，如图 4 所示。这个机制要求任何能够响应 binder 内核驱动程序调用的 binder 存根必须是 Binder 类的子类。因此，代理对象也必须继承自 Binder 类，以便注册到 binder 内核驱动程序，从而正确拦截应用程序对本地服务 binder 存根的调用。

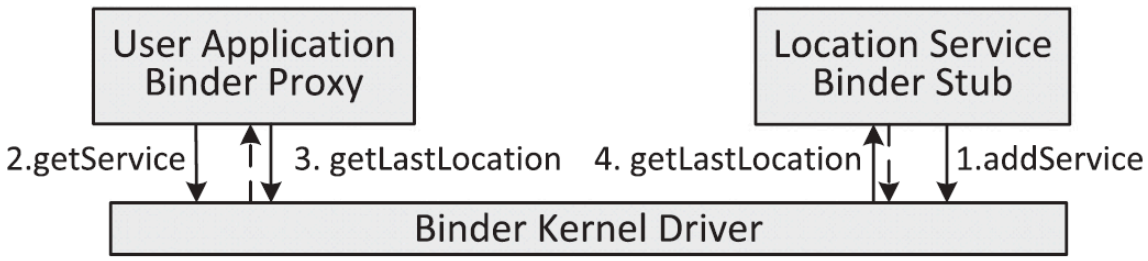


Fig. 4. Android binder mechanism with the example of location service.

文章方法是通过从现有系统服务类中开发代理对象，通过**动态织入**（一种织入方法，在程序运行期间将横切的方面代码织入到纵切的核心代码中）将远程调用操作的字节码附加到服务类中。新织入的类将是Android Binder类的子类，并且可以注册到Binder内核驱动程序中，以接收和拦截应用程序对服务方法的调用。
框架还支持回调到注册了系统事件的用户应用程序，如图5，用户应用程序利用Android系统库将此监听器包装成一个Binder桩，该桩与系统服务中的Binder代理进行通信以监听事件。为了实现两个移动设备之间的这种回调，如图3所示，我们允许应用程序通过回调代理在远程系统中注册并监听事件，并利用客户端的事件监听器Binder代理作为回调句柄。之后，当系统事件

在服务器端发生时，回调代理将发起对客户端回调句柄的远程调用。

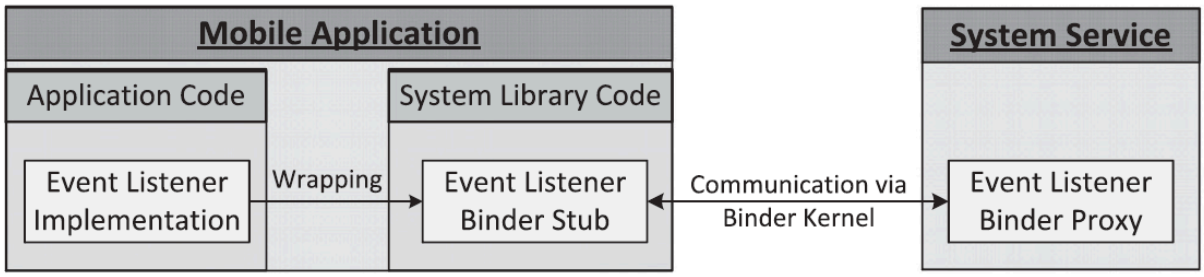


Fig. 5. Resource sharing through callbacks.

3.1.2 基于C/C++的系统服务

问题：
Android中的传感器和图形服务以及iOS中的所有系统服务在运行时作为编译后的二进制文件执行，很难在运行时定位原生方法的入口和出口点，因此难以动态地将编织指令附加到服务类上。其次，这些原生服务类中的机器指令依赖于硬件架构，因此只能静态地操作和编译。
解决：
手动修改每个系统服务类的源代码，以实现序列化和反序列化功能，以及服务方法的远程调用。在每个系统服务上产生的工程开销很小，不到300行代码。需要注意的是，这种手动修改的方法是通用的，适用于所有其他操作系统服务。与现有方案不同，现有方案通过设备驱动程序共享系统资源，并且必须为每个移动系统重新编程驱动程序实现，而我们的修改是一次性工作，适用于所有具有异构硬件组件的移动系统。

3.1.3支持并发资源共享：

并发共享多个资源：
不同类型的资源可能在两个移动设备之间同时共享，导致**多个服务方法的并发调用**，需要避免调用错误的提供者对象。确保提供者和请求者之间调用一致性的方法是标识符映射方案，如图7。代理对象模块中维护一个索引表，其中包含标识符与提供者之间的映射。当请求者向远程提供者发出请求时，请求消息中将包含一个标识符，指定目标提供者。另一端的代理对象模块将根据接收到的标识符查找提供者，并对该提供者进行方法调用。

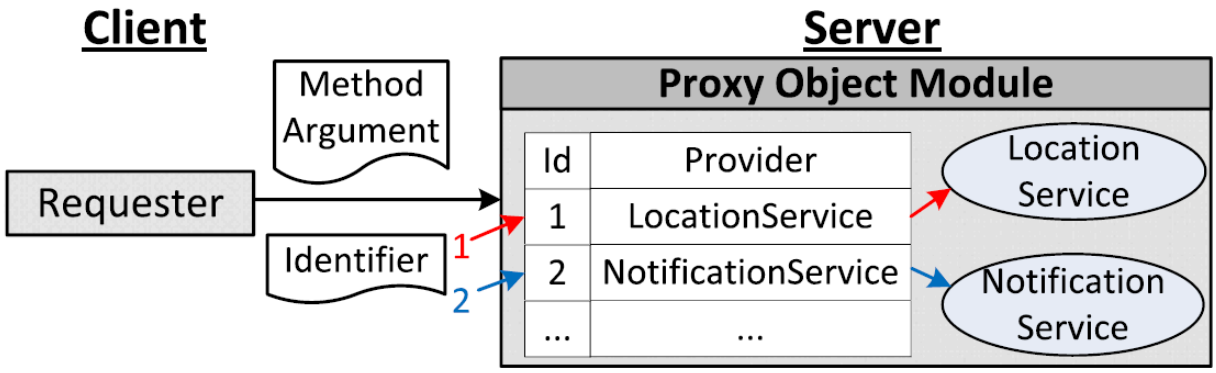


Fig. 7. Identifier mapping in the proxy object module.

并发访问同一资源：
本地移动应用程序和多个共享客户端可能同时访问同一资源。对于固有支持多个应用程序的系统服务（如位置服务和传感器服务），可以通过上述标识符映射方案轻松实现并发访问。相比之下，多媒体系统服务仅能由一个移动应用程序独占访问，并且禁止对这类资源的并发访问，当新的共享客户端需要访问已被占用的多媒体资源时，我们的框架允许移动用户通过应用程序接口组件中的配置决定是等待还是抢占资源访问。

移动设备同时作为共享服务器和客户端：
框架允许移动设备自由地同时充当共享服务器和共享客户端。
作为共享服务器时的守护线程处理远程资源访问请求，并与相应的系统服务交互以提供资源数据。
作为共享客户端时的守护线程连接到共享服务器，并为远程调用系统服务创建相应的服务代理。

3.2Unix域套接字：

框架中支持的另一种IPC方案是Unix域套接字。因为在Android中，传感器数据并不是通过Binder方法而是通过传感器服务和应用程序之间的Unix域套接字传递，以减少频繁数据传输的系统开销。更具体地说，服务器中的系统服务进程将监听连接请求，并建立分布式数据通道。当应用程序请求远程系统服务时，客户端中的代理对象将建立与服务器的连接，并将该连接的文件描述符传递回应用程序，服务器中的系统服务便可以通过TCP与客户端应用程序可靠地交换数据。由于移动操作系统使用相同的API来操作这两种套接字，因此数据通道套接字的实际类型可以对使用数据通道的系统隐藏，并且用于数据交换的现有IPC代码可以保持不变。

3.3错误处理：

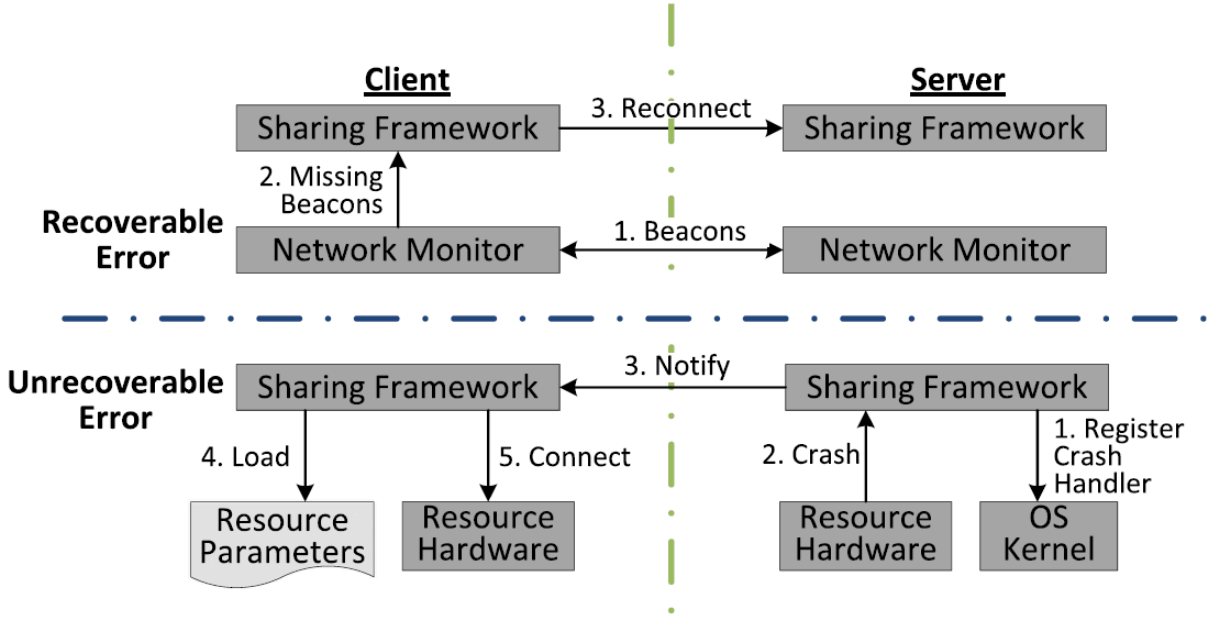


Fig. 8. Error handling in the sharing framework.

对于可恢复的系统错误（例如，不稳定的网络连接），我们的框架会在错误恢复后重新连接到远程共享服务器，并恢复使用远程资源，如图上半部分，客户端的框架会定期向服务器发送信标。如果信标超时，客户端的框架会发起与服务器的重新连接。对于不可恢复的系统错误，如资源运行时故障和不可用性，我们的框架选择回退到本地资源访问。如图下半部分，共享服务器在资源访问期间监听到错误，并向操作系统内核注册处理程序，该程序会在终止资源共享之前通知共享客户端。同时，在调用远程系统服务期间，框架会记录移动应用程序访问远程资源时使用的参数。当客户端收到不可恢复的系统错误通知时，框架会使用记录的参数调用本地系统服务，并将本地资源数据返回给移动应用程序。

4应用程序接口

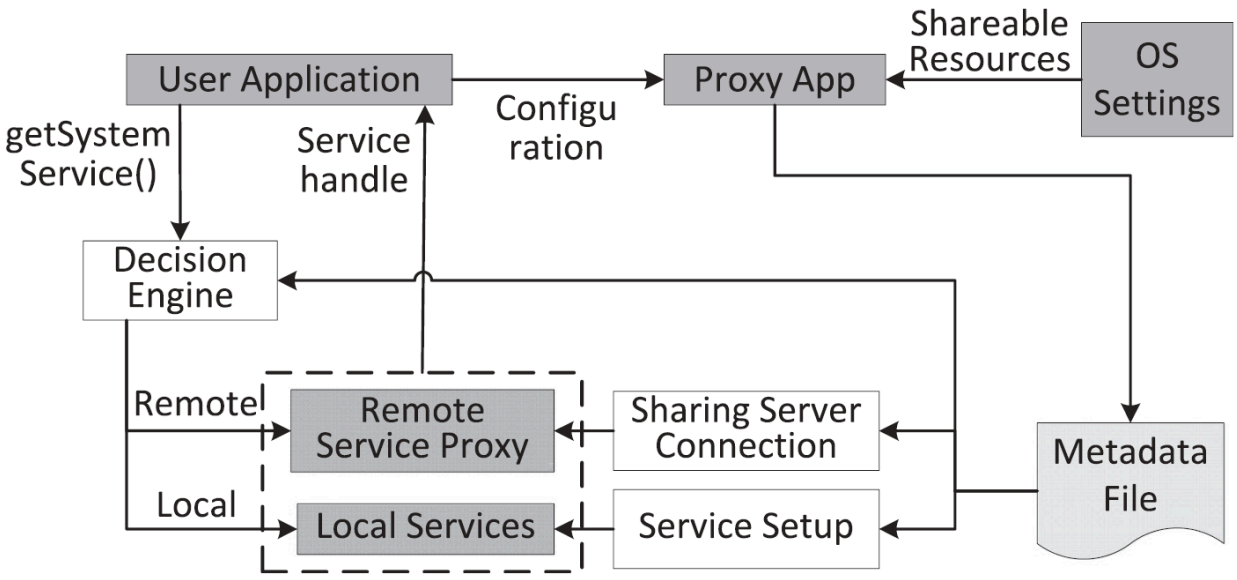


Fig. 9. Design of application interface.

应用程序接口设计如图9所示。基本上，每当用户应用程序请求访问系统服务时，应用程序接口会拦截该请求并返回适当的系统服务句柄，该句柄可能位于本地系统或远程系统中。由于这两个句柄都向应用程序提供了相同的编程接口，因此移动系统之间资源共享的过程对用户应用程序是完全透明的。

当用户应用程序请求访问系统服务时，框架（Decision Engine）会从**元数据文件**（Metadata File）中加载配置。如果配置指示将访问本地系统资源，则将本地服务句柄返回给用户。否则，我们的框架将创建一个远程服务代理，并建立与远程系统的TCP连接以迁移资源访问。这个元数据文件由一个特殊的代理应用程序(proxy App)操作，该应用程序作为应用程序接口的一部分嵌入，管理用户所有应用程序的资源访问配置，还接收来自移动操作系统设置的共享资源信息，所有这些信息都将由代理应用程序写入元数据文件。

5 多媒体操作

多媒体资源的操作通常涉及大量的批量数据，并且需要利用共享内存来实现应用程序之间的数据交换，本节介绍如何设计以支持访问远程移动系统之间的多媒体服务的共享内存。然而，主要的挑战在于如何在不改变原始移动操作系统结构和接口的情况下，确保远程进程间通信（IPC）的可靠性。

根据共享内存的操作方式，我们将共享内存分为两种情况并分别处理：通用缓冲区和图形缓冲区。

5.1通用缓冲

通用缓冲区可以分为两部分。首先，内容部分(Content Part)存储资源数据，并遵循生产者-消费者模式进行操作，即一个操作符总是将数据写入缓冲区，而另一个操作符总是读取数据。通过这种方式，两个操作方始终保持同步，避免了写入时的争用。其次，控制部分(Control Part)存储操作内容部分所需的控制信息，例如读写指针。

采用写更新和写失效协议来实现共享内存的同步。写更新协议中，每次本地写入后都会立即发送同步消息，以始终保持本地和远程内存之间的一致性。在写失效协议中，远程内存将保留旧值并带有失效标志（图中红色表示失效），该标志由本地写入前发送的失效消息设置，新值在本地读取发生之前不会同步。本文采用写更新协议来同步频繁读取的内存，并节省失效消息的开销。采用写失效协议来同步不频繁读取的内存，以最大化内存同步的效率。

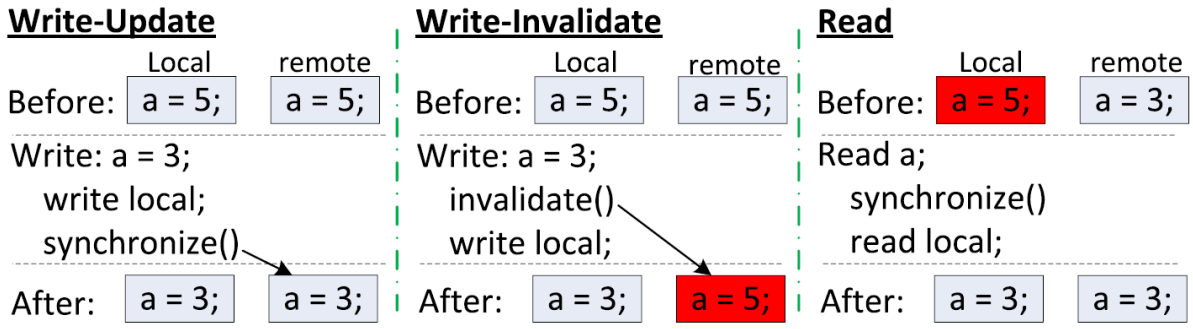


Fig. 10. Memory synchronization protocols.

5.1.1内容部分

为了高效地操作内容部分，与传统的分布式共享内存方法不同，这些方法通常以固定大小的单元共享内存，但可能导致移动系统之间大量冗余的数据同步。本文根据内存访问的实际应用模式，灵活地以任意大小同步共享内存。

本文在客户端和服务器之间建立内存映射，并基于该映射同步内存内容。每当一个端点分配一块共享内存时，另一端点也会相应地分配相同大小的内存块，并在两端点维护的映射表中添加它们地址之间的映射条目。每当一个端点完成其写操作时，我们使用映射条目和写入偏移量来计算目标写入地址并同步被写入的数据。例如，在图11中，当客户端分配12字节的内存时，服务器也会相应地分配相同数量的内存，并且客户端和服务器内存的地址都存储在两端点维护的映射表中。之后，当客户端向缓冲区写入5字节时，资源共享框架根据接收到的目标地址和偏移量，同步并更新适当大小的内存。

在写操作期间，被写入的共享内存地址集合被标记为无效，并添加到远程系统维护的列表中。

当另一端点读取内存时，我们的框架根据维护的列表同步内存。

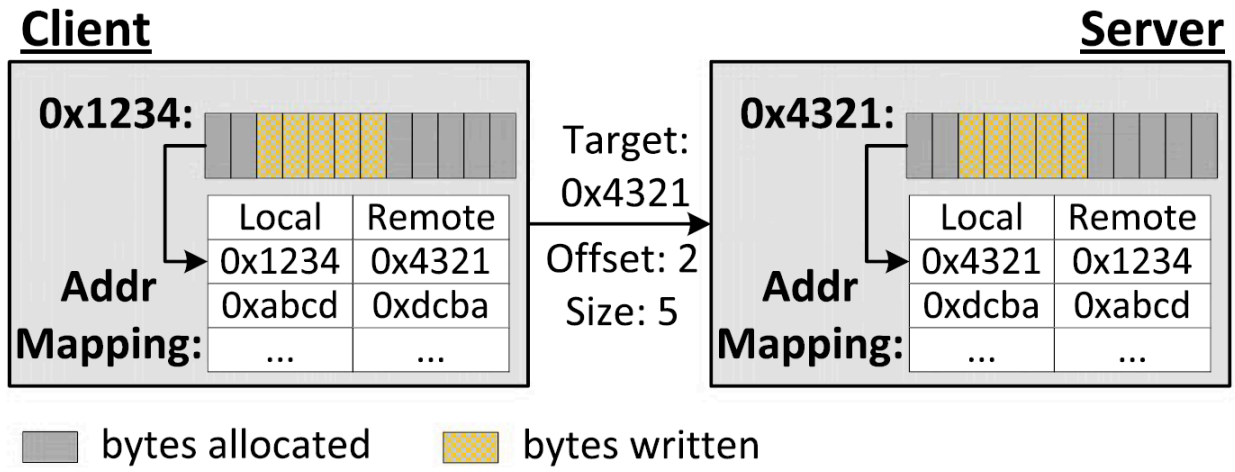


Fig. 11. Memory synchronization for content part.

5.1.2控制部分

为了确保在写入竞争情况下的数据一致性，本文通过所有权标志在两个移动设备之间建立了一个“先发生”关系，并且只允许内存字段的所有者对该字段进行写入（owner=1表示所有者）。当另一端尝试写入该字段时，所有权会发生转移。

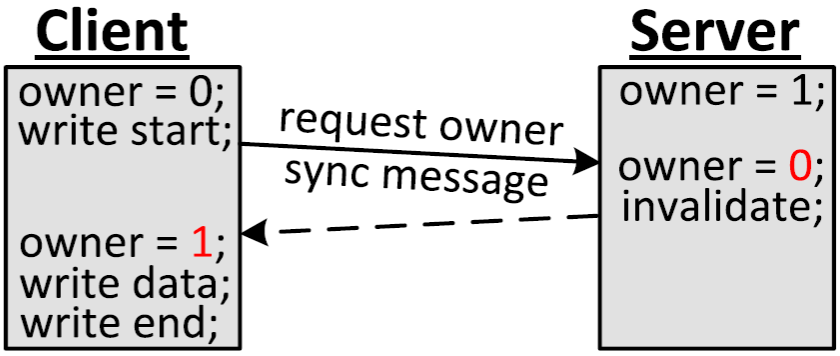


Fig. 12. Memory synchronization for control part.

5.1.3优化

在实践中，观察到对内容部分的写入通常紧随其后的是对控制部分的更新。例如，应用程序将数据写入内容缓冲区后，会更新控制部分中的下一个写入位置。系统服务可以延迟内容部分的同步，直到相应的控制部分开始同步。然后，内容数据可以与控制部分的同步消息一起发送，从而减少了通过无线链接进行多次往返的额外开销。

5.2图形缓冲区

图形缓冲区由供应商特定的HAL（硬件抽象层）分配和操作。因此，我们无法直接从资源共享框架中拦截缓冲区操作，并进一步为远程系统之间的同步建立内存映射。我们的方法是将与操作系统提供的用于用户应用程序管理和操作图形缓冲区的API集成到我们的资源共享框架中。因此，我们的资源共享框架充当BufferQueue的消费者，提取图形缓冲区的内容，然后将这些内容推送到远程设备。

在图13中，服务器中的摄像头驱动程序将预览图像缓冲区发布到BufferQueue后，我们的框架作为消费者收集图形缓冲区内容，并将其发送到客户端。然后，客户端的框架检索一个空缓冲区，用接收到的图像内容填充缓冲区，并将缓冲区重新发布到BufferQueue。之后，客户端的Surface Flinger在摄像头应用程序中渲染预览图像。

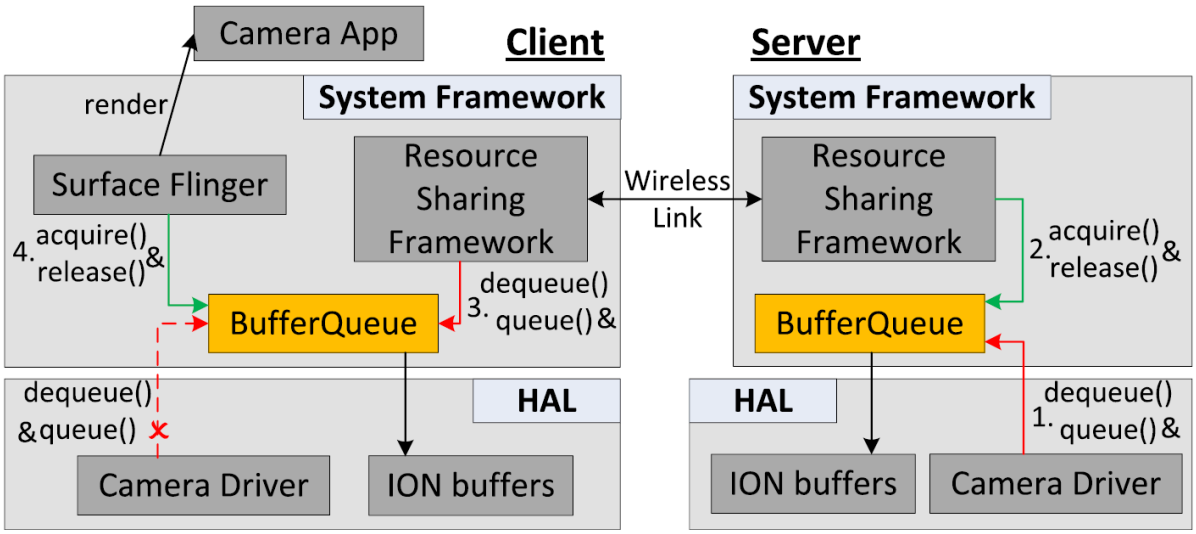


Fig. 13. Operating the graphic buffer for remote camera access.

6实现

在Android5.1.1通过CyanogenMod12.1实现，CyanogenMod是一个基于开源Android系统，供某些手机使用的二级市场固件。它提供一些在官方Android系统或手机厂商没有提供的功能 。动态编制通过dexmaker实现，Dexmaker 是一个针对Dalvik虚拟机的Java语言API，支持在编译时或运行时生成代码。

下表为支持不同服务所需代码行数（LoC）。

TABLE 2
List of Supported Services

Service	Type of code	LoC	Hardware
Location	Java	<10	GPS, WiFi network
Sensor	C++	283	All onboard sensors
Audio	C++	647	Speaker
Camera	C++	594	Camera

同时为了解决具有不同网络接口的设备相互连接复杂的问题，本文部署了网络管理模块在高层提供统一的接口。

6.1基于服务的优化

传感器：在套接通道上附加专门的控制模块，无论硬件或操作系统产生数据的速率有多快，允许移动设备之间以指定速率接受互相的数据。服务器会接收传感数据速率，并仅在必要时将收据传送给客户端。
音频数据：安卓系统中只有音频缓冲区填满后音频才会播放，而且应用程序的一次写入操作可能不足以填满缓冲区，这样每次写操作之后立即同步移动用户会遇到很长的延迟，为了减少延迟框架会积累缓冲区内容，消除初始缓冲区的多次往返。
通知服务：框架还支持移动系统之间共享消息服务，允许用户在远程设备上显示通知。移动用户点击图标时，通过远程回调将关联操作在本地移动设备执行。

6.2跨移动平台部署

在不同制造商制造的的硬件及不同版本的驱动程序之上实现框架，传统dsm方案需要手动将驱动程序从一个设备移植到另一个设备，本文框架利用操作系统接口实现移动平台自动迁移，无需手动修改。
对于手机和平板电脑来说只需要更新最新的系统安装包就行。但智能手表的开源操作系统尚未完成，所以需要替换智能手表中的现有库文件将框架部署上去。
下图传感器服务中Java库时用户应用程序和传感器交互服务的入口，应被修改为获得远程服务

句柄并给应用程序。

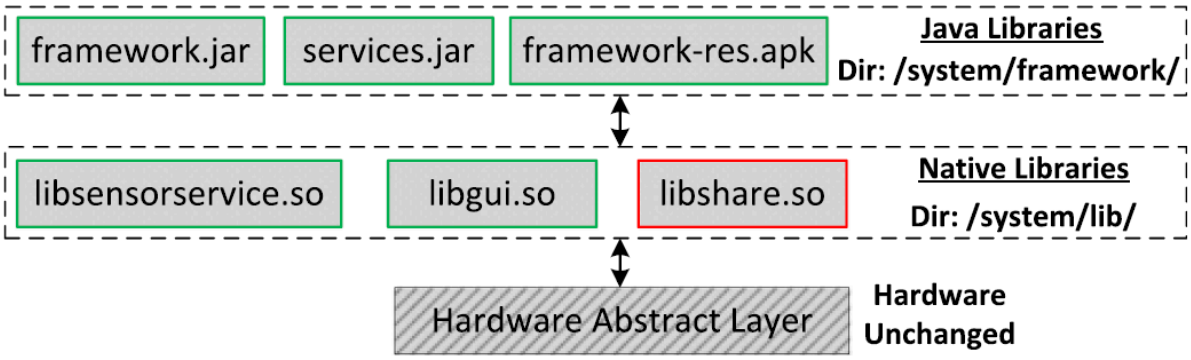


Fig. 14. Porting sensor service libraries in Android wear.

7性能评估

7.1实验设置

不同移动平台：三星Galaxy S4，LG Nexus4，三星Nexus 4平板和LG Watch Urbane，系统为Android 5.1.1。
网络：40Mbps校园网，延迟为3.5ms
功率监视器：Monsoon power monitor

7.2资源共享性能

不同类型资源的访问延迟，包括网络传输时间、系统服务方法以及共享框架的运行耗时。对四种资源进行试验：间隔1s的GPS数据，间隔20ms的加速计数据、44.1KHz的音频及176*144的图片，运行多次取平均。

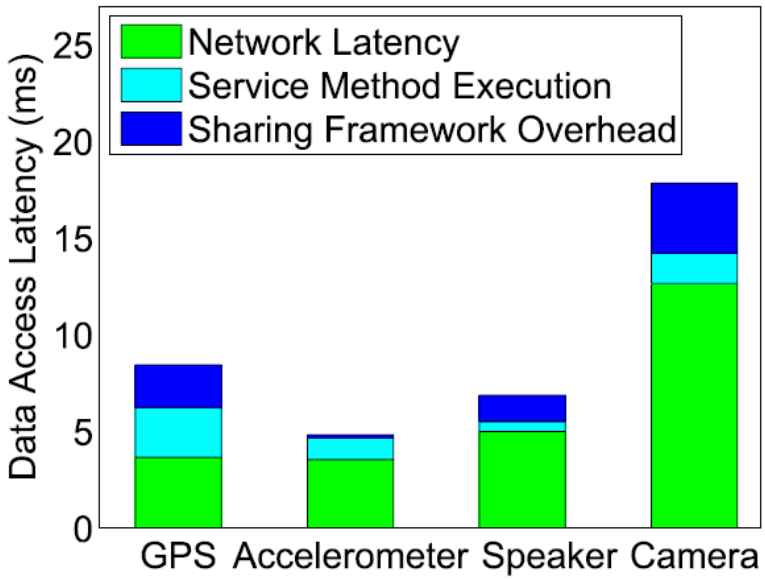


Fig. 15. Latency of accessing shared resources.

如上图服务框架耗时较少，延迟大多数情况由网络延迟决定，数据量越小延迟越低。

在不同缓冲区大小下对音频实验：

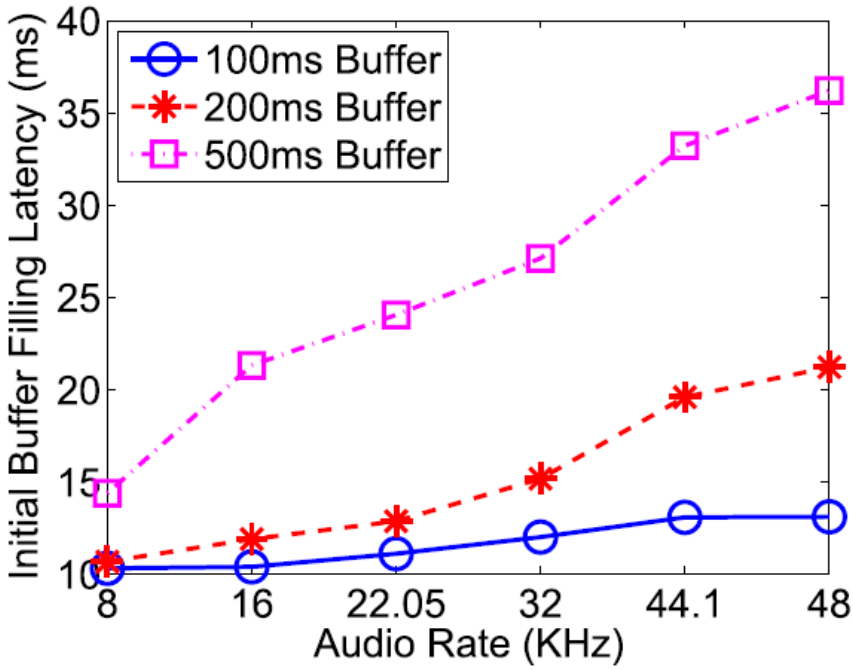


Fig. 16. Latency of initial audio buffer filling.

可以发现初始延迟随缓冲区的大小增大而增大。

不同像素时相机的FPS：

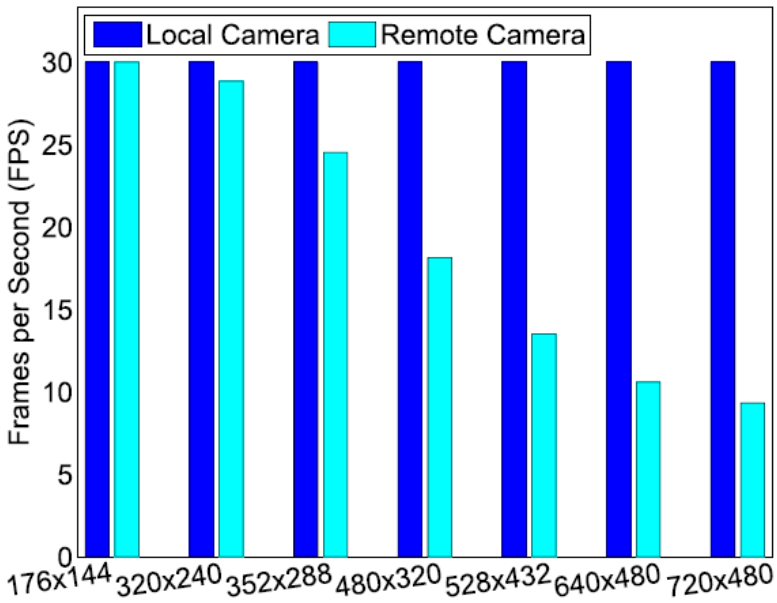
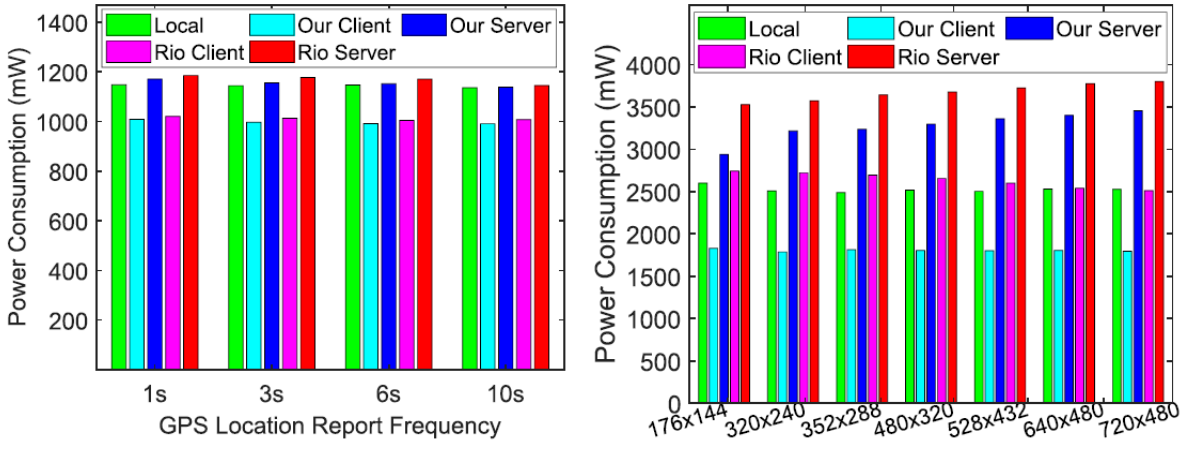


Fig. 17. Client FPS for real-time camera preview.

最低分辨率时框架可以和本地表现相同，当分辨率不断上升时由于无线链路传输的数据量增加，FPS会下降。

7.3功耗

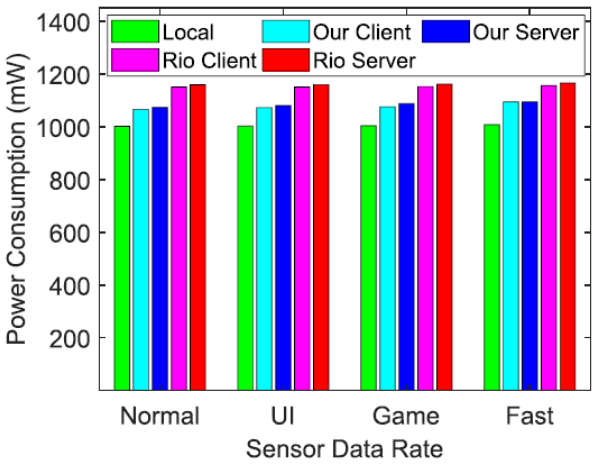
将使用本地资源时的功耗、运行客户机和服务器的功耗以及**RIO共享系统**的功耗比较。如下图：



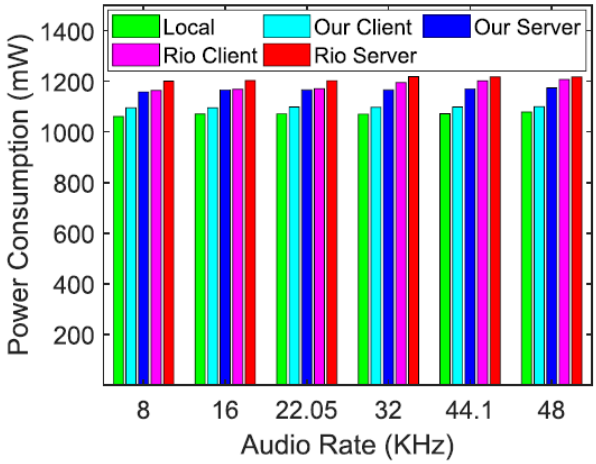
(a) GPS sharing (b) Camera sharing

相比RIO，本文框架耗能更少。框架在访问远程而不是本地资源时减少了较多功耗，服务器需要消耗额外的能量向客户端发送消息，但服务器一般是功能更强的设备，这些额外成本是可接受的。

下图为**加速计**（传感器的一种）和扬声器的实验比较：



(c) Sensor sharing

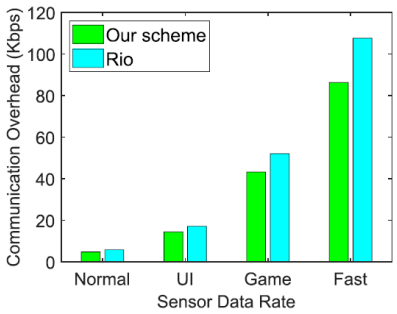


(d) Audio sharing

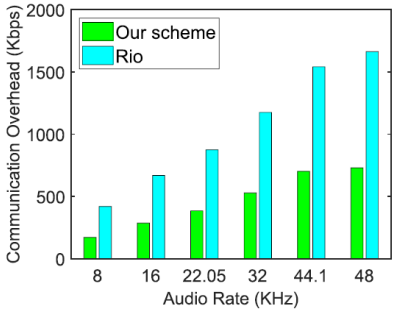
客户端相对本地来说消耗了更多的功率，是因为这两类数据需要频繁的同步资源数据，导致无线传输消耗额外的能量。

7.4无线通信开销

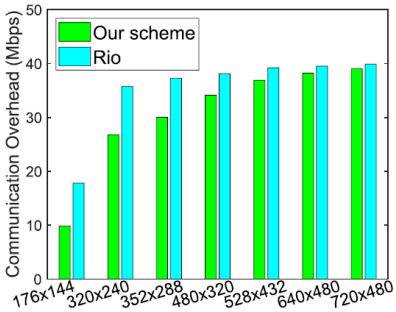
本节评估远程资源共享带来的无线通信开销，资源包括加速计、音频、相机。如下图：



(a) Sensor sharing



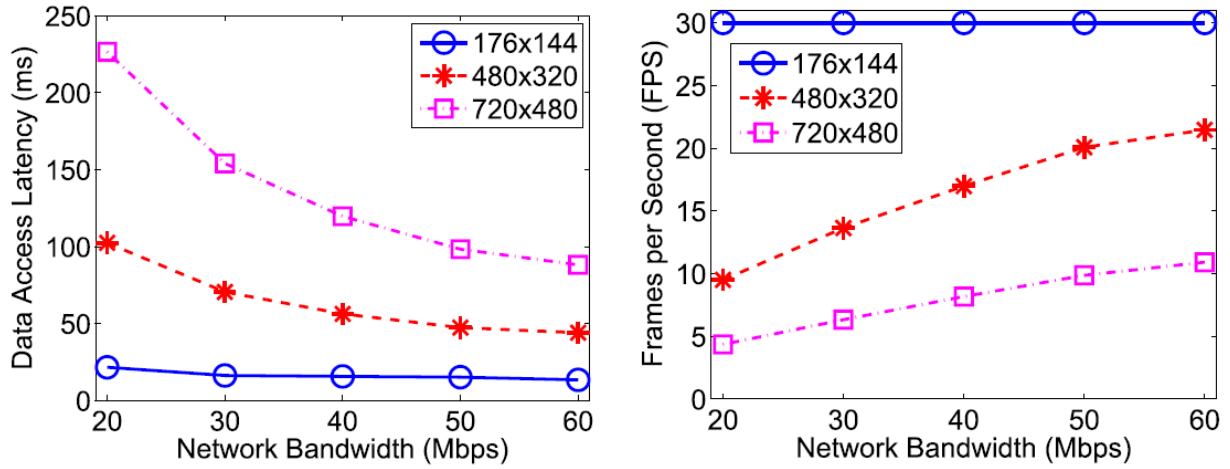
(b) Audio sharing



(c) Camera sharing

三者因为数据的增大开销也增大，前两者只需要1Mbps就可以支持远程访问，但相机需要更高的无线带宽，需要高效的网络协议解决。

测试带宽与摄像机性能相关性：



(a) Latency of previewing images (b) FPS for camera preview

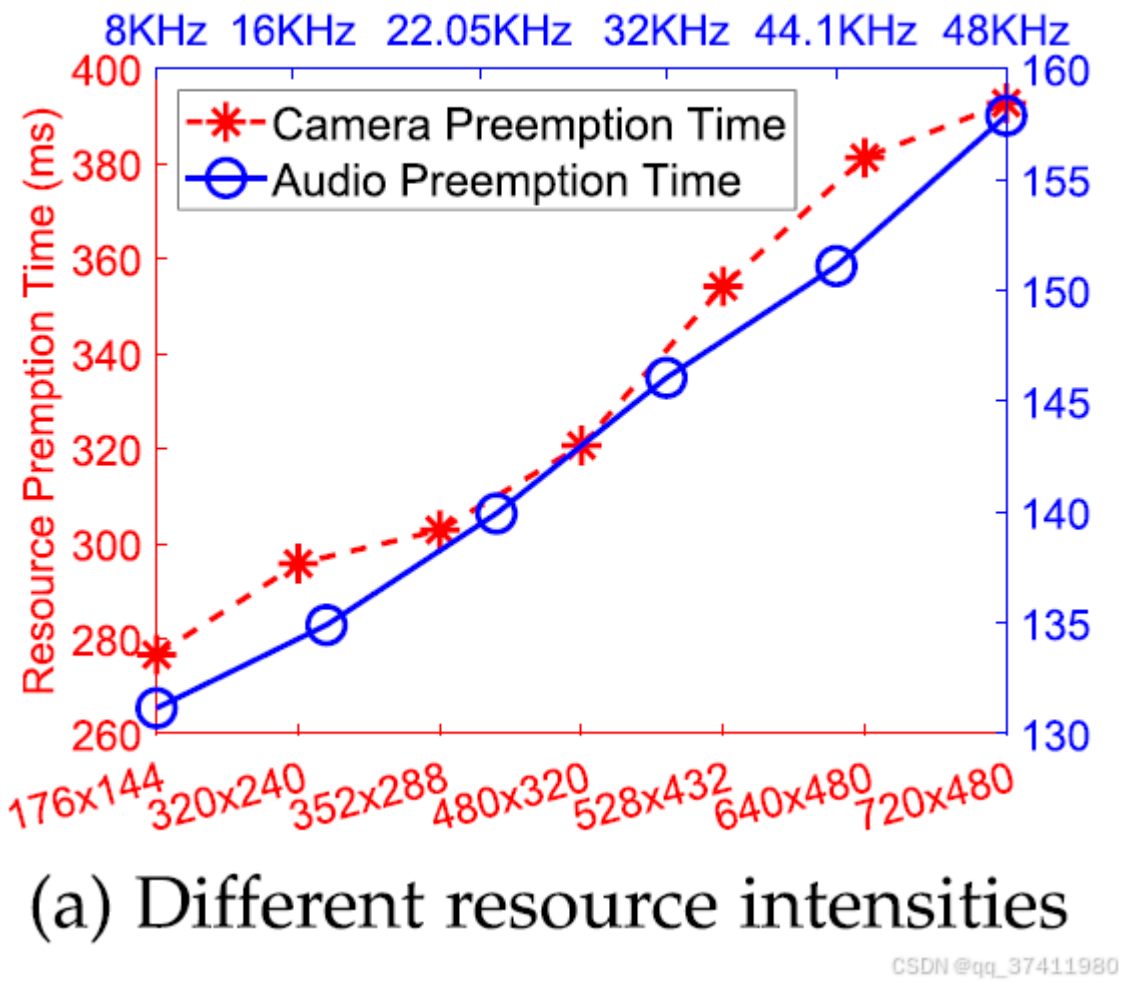
Fig. 20. Performance of remote camera access with different network bandwidth.

图a随带宽增强相机延迟降低，因此可以传输更多的预览帧，以实现更高的FPS。图中最低分辨率图像的瓶颈是硬件相机的图像捕获率。

7.5并发资源共享

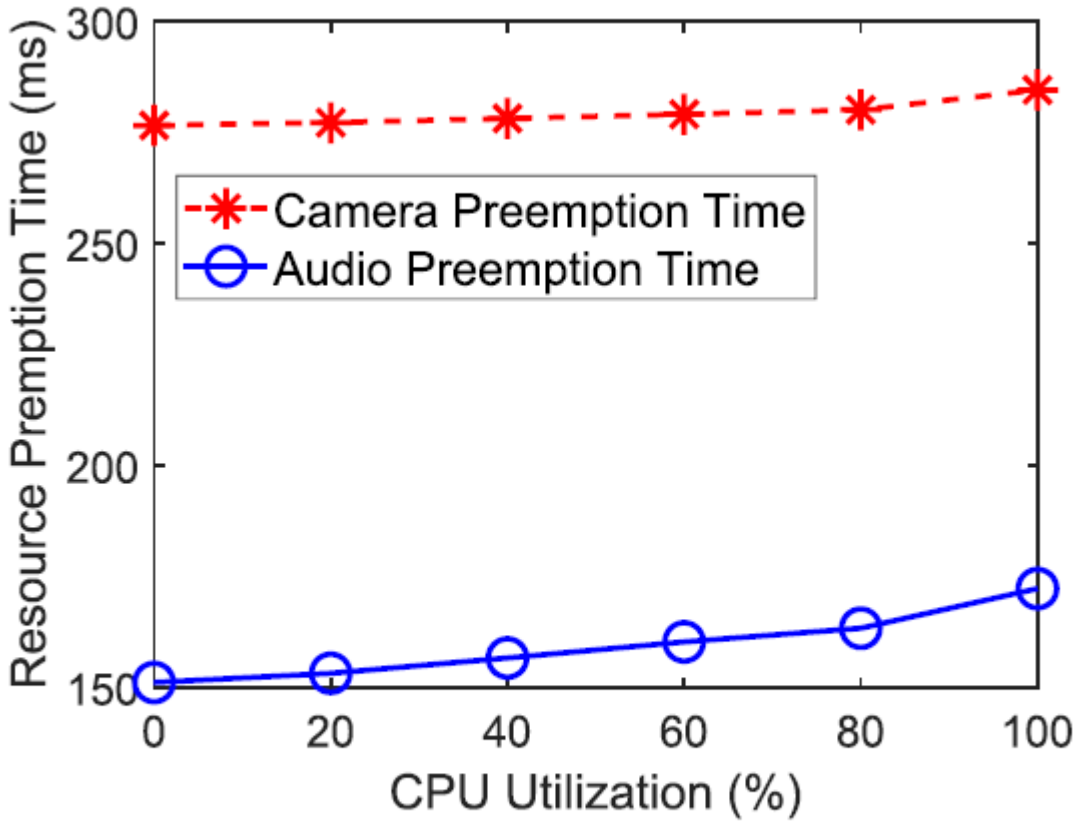
7.5.1多客户端访问

多媒体资源：扬声器和相机应该是被独占的，一个程序使用时若有新的请求则被新请求抢占。



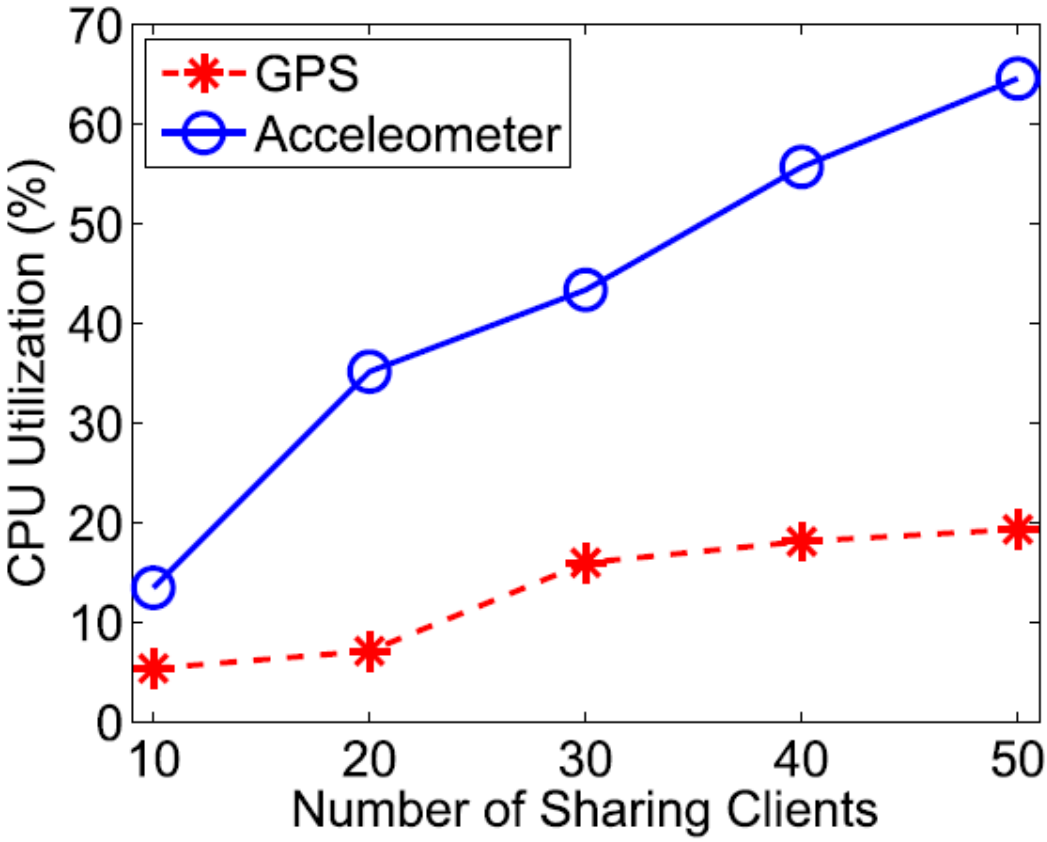
(a) Different resource intensities

上图显示抢占一个资源的延迟主要来源于重新初始换资源硬件及查询设置硬件参数，故服务器工作负载情况对抢占影响很小，如下图：



(b) Different server workloads

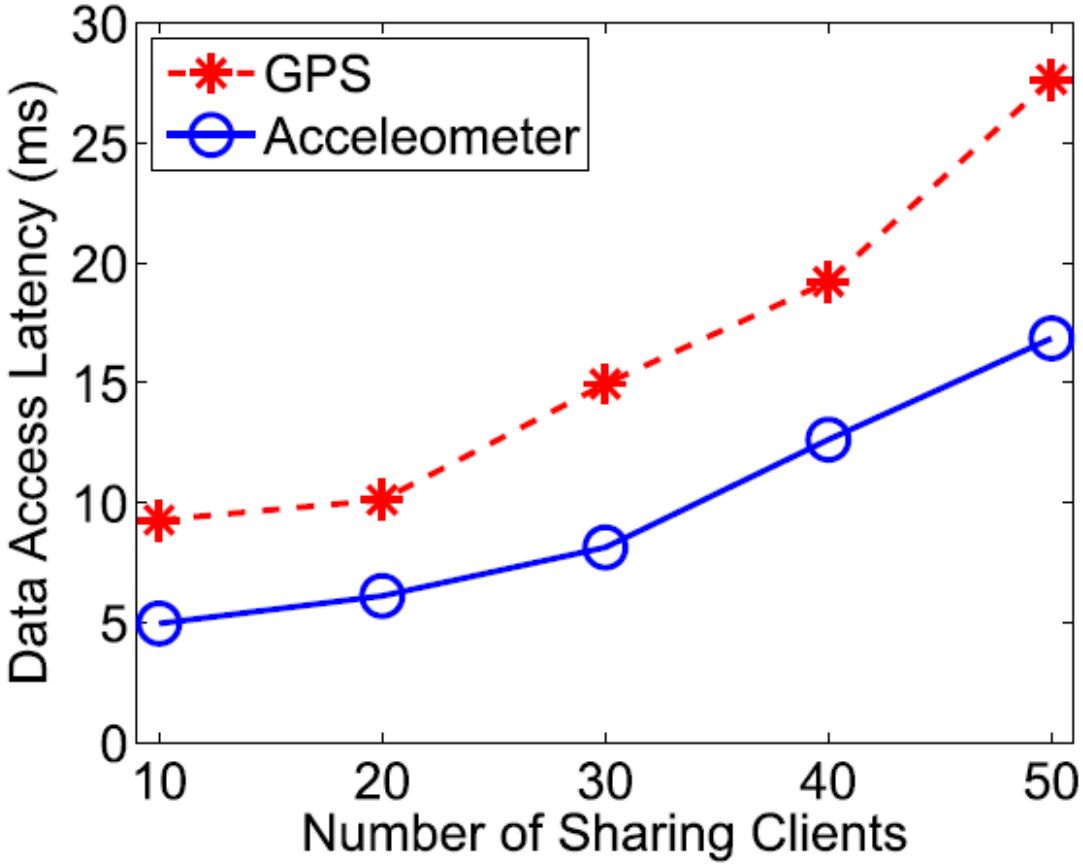
传感器，GPS资源：资源可同时被多个应用使用。实验时同时访问两类资源，首先测试服务器CPU利用率受客户机数量的影响：



(a) CPU utilization of sharing server

GPS的情况不是线性的，因为位置服务是单线程的，请求必须按顺序完成。但传感器数据会频繁传输，操作系统需要处理小而频繁的数据包。

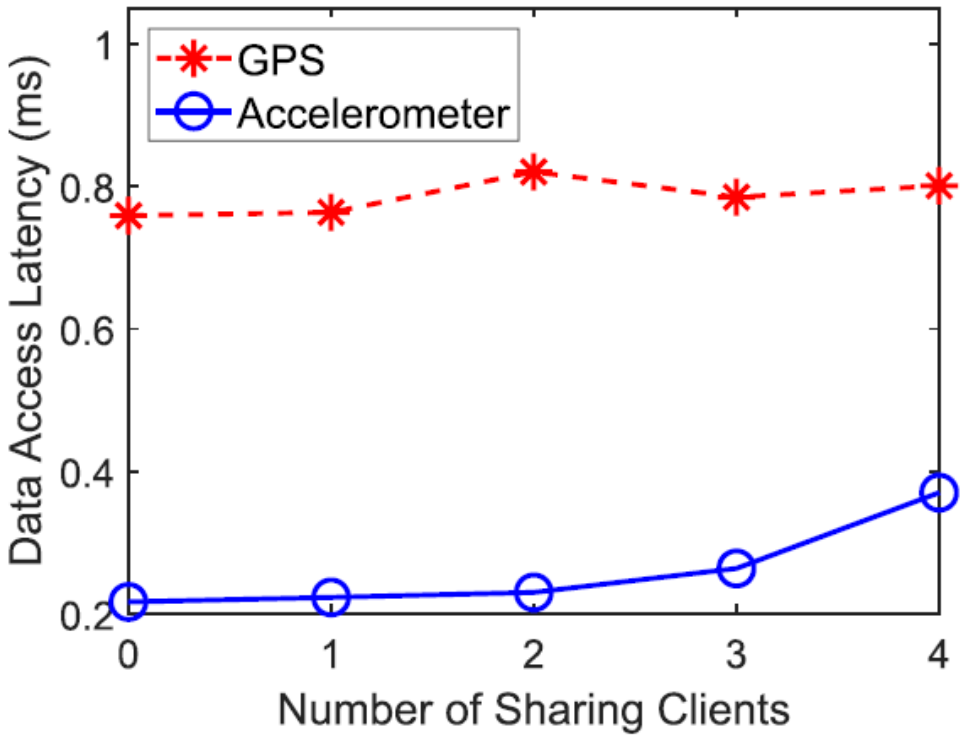
并发请求对服务器的响应性进行评估：



(b) Resource data delivery time to client

数量低时影响小，高时影响大。

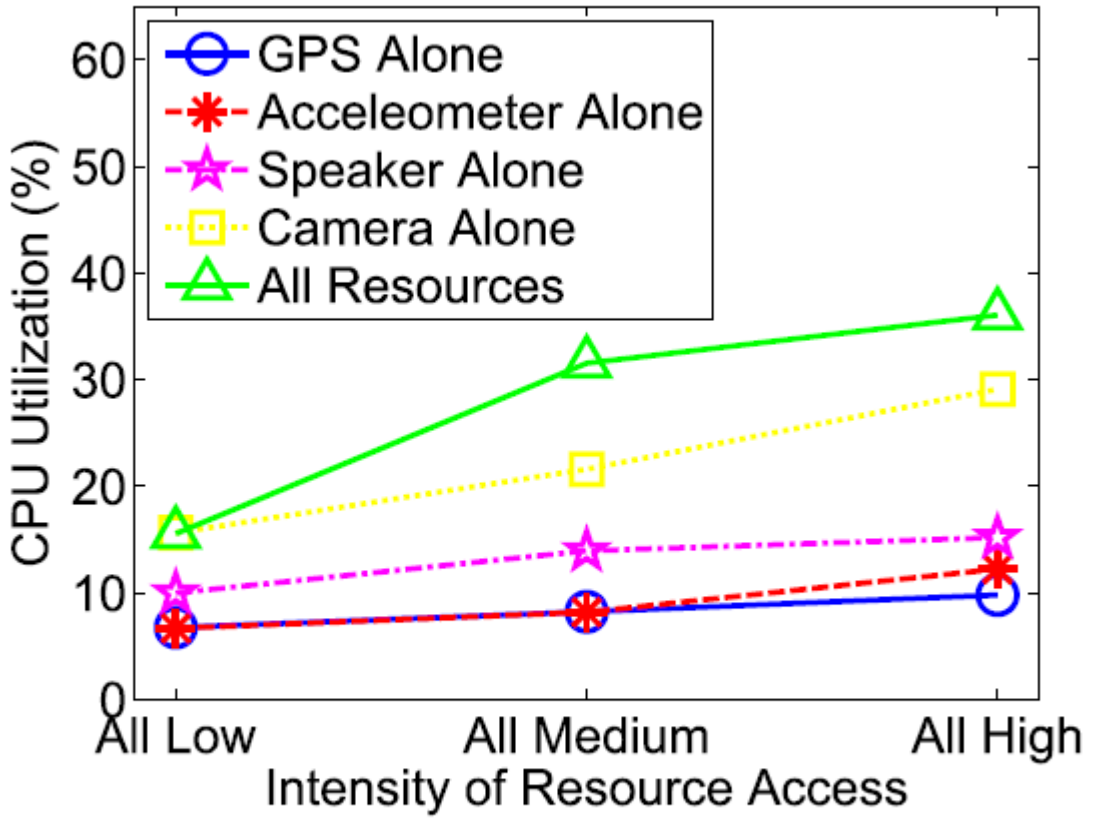
并发请求对服务器请求本地资源的影响：



(c) Local resource data delivery time at server

其影响很小，因为频率很低。

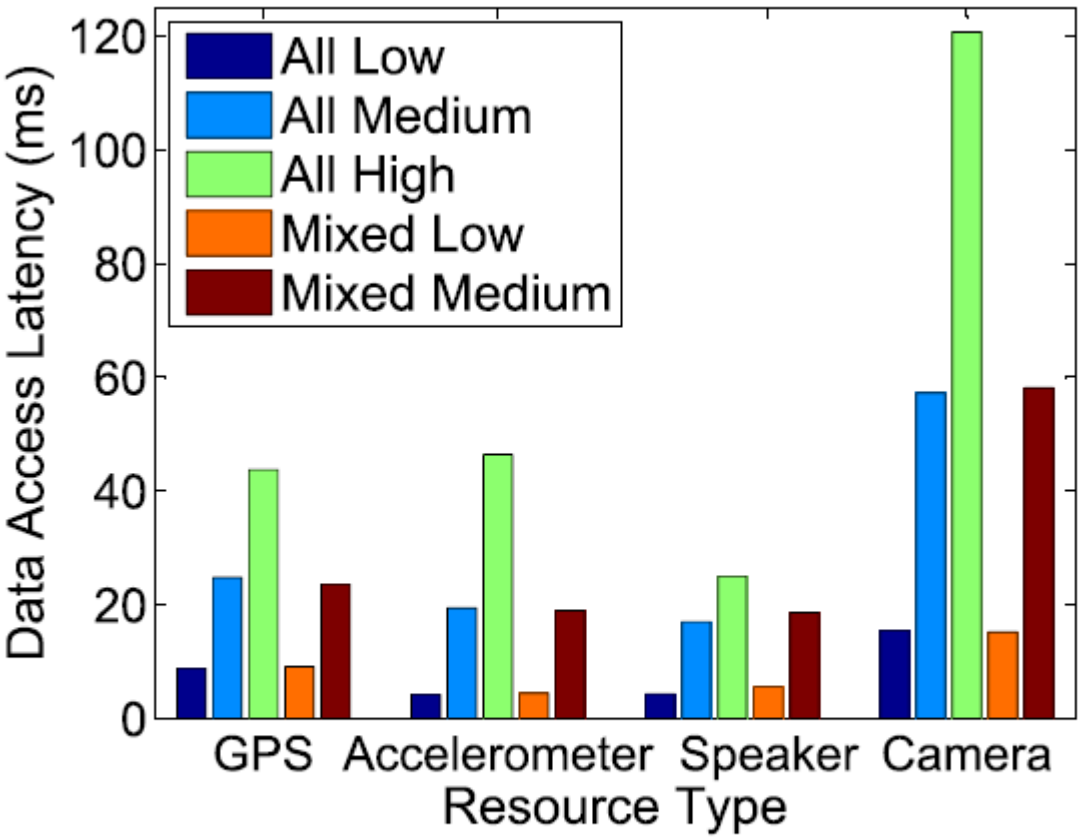
7.5.2单客户端同时访问多资源



(a) CPU utilization

上图显示随着资源使用的增加cpu利用率不断增加，但在高强度的访问下CPU利用率只有36%左右，说明框架可以在低开销情况下支持资源并发访问。

评估并发访问对资源传输速率影响：

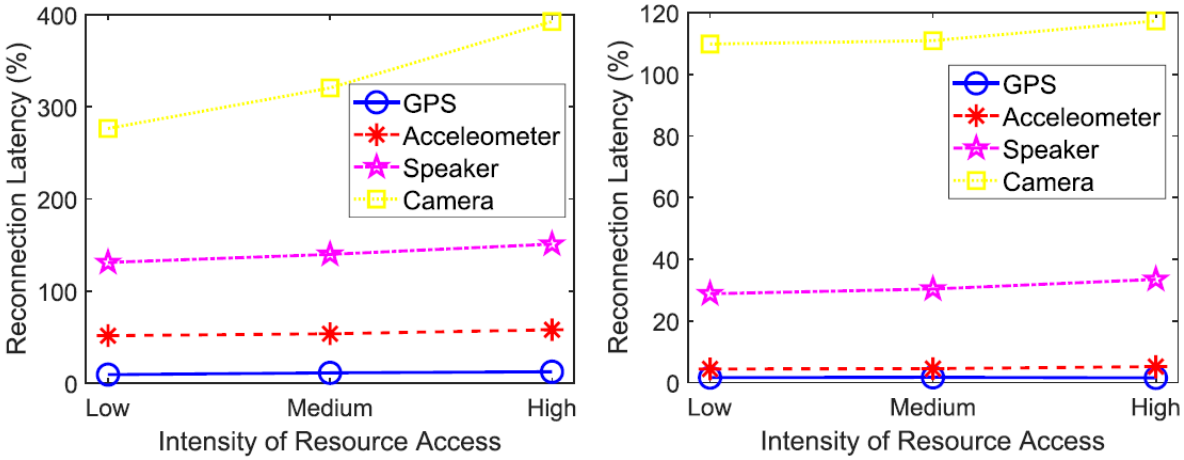


(b) Resource data delivery time

随着强度增加，资源传输到客户端时间会变长从而增加传输延迟。

7.6错误处理

通过重置网络连接和访问远程资源时抛出异常评估框架处理错误的效率：



(a) Remote reconnection (b) Local reconnection

Fig. 24. Latency to reconnect the system resources when error occurs.

其中重新连接到远程资源需要更长时间，因为需要重新构建数据交换通道和配置硬件参数。多媒体资源延迟更高是因为需要设置共享缓冲区及更复杂的资源初始化。

8相关工作

移动代码分流：支持在云上执行应用程序来减少本地资源消耗。主要是聚合一组相互协作的移动设备，将他们的计算资源集中用于服务。

应用程序级资源共享：系统间资源共享的研究主要在瘦客户机（指的是在客户端-服务器网络体系中的一个基本无需应用程序的计算哑终端。它通过一些协议和服务器通信，进而接入局域网。瘦客户端将其鼠标、键盘等输入传送到服务器处理，服务器再把处理结果回传至瘦客户端显示），但只能共享开发人员指定的特定类型的设备。本文研究的框架可以共享所有类型的系统资源。

操作系统级资源共享：
在OS层支持分布式客户机之间的资源共享。移动网格计算使移动设备可以加入网格并共享资源，但这些资源只能通过特定框架的API访问，不能由程序直接使用。
RIO是第一个在移动系统之间共享多种IO设备而不用修改客户应用程序的方案。但它的实现与硬件驱动程序绑定，对不同类型的硬件必须重新编程。文章的方案在OS的更高层实现，不一致性由OS内核解决。

远程方法调用：
是远程系统之间的基本IPC方式，常用于分布式系统中，未用于共享系统硬件和软件。

9讨论

9.1远程相机共享的像素格式

共享图形设备时移动设备间像素格式可能不一致，因为不同设备供应商可能有不同的像素格式，解决方法是应用一种与所有类型的移动设备兼容的图形格式。未来探索如H.264标准的图形数据编码。

9.2访问控制

需要一个身份验证或控制方案防止恶意方的攻击，可以在序列化之前添加新的身份验证层实现。

9.3远程资源访问限制

有一个限制是若没在相应系统服务的api中预先定义对远程系统资源的控制操作，便无法操作。如一些硬件配置由操作系统库设置系统服务无法访问，但这种限制对访问远程服务无影响。

10结论

本文的移动系统框架可以有效地将异构移动设备连接到移动云，实现设备间资源共享。基本思路是允许移动设备通过远程调用操作系统的服务来访问另一个移动设备上的远程系统资源，并进行实验证明了有效性。