面向智能空间的异构网络同构化通信框架

杨 溢 王志良 王 鲁 张富深
(北京科技大学计算机与通信工程学院 北京 100083)

摘 要 从智能空间的一般通讯需求出发,提出了一种面向智能空间的异构网络同构化通信框架,并对其构建方法进行了详细描述。这种通信框架同时具有动态性、松耦合性、鲁棒性、通用性、灵活性等特点。使用实际智能家居环境(LookeyHome)进行了一系列多角度、多线程并发测试,结果表明,即便是采用吞吐量最低的基于wsDualHttpBinding的网关中转通信,该通信框架也能至少承受每秒大约70次的并发访问,这个性能足够胜任绝大多数智能空间的通讯需求。

关键词 异构网络,智能空间,Agent
中图法分类号 TP311 **文献标识码** A

Heterogeneous Network Communication Framework in Smart Space Environment

YANG Yi WANG Zhi-liang WANG Lu ZHANG Fu-shen
(School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)

Abstract A heterogeneous network communication framework that meets the general communication needs in smart space was proposed and described in detail. In addition to dynamic adaptability, the framework also has other characteristics such as loosely coupled, robustness, versatility, flexibility. Serials currency tests in a real smart home environment (LookeyHome) indicate that even when communicating in gateway wsDualHttpBinding transmit mode which has been proved with minimum throughput, the framework can withstand concurrent access at least about 70 times per second. This performance is good enough in most smart space communication occasions.

Keywords Heterogeneous network, Smart space, Agent

1 引言

智能空间是一个嵌入了计算机、信息设备和多模态传感器的工作空间,其目的是使用户能够方便地访问信息和获得计算机的服务,进而高效地实现个人目标和与他人协同工作。智能空间的标志性特征是信息服务的透明性和随时随地性,使人们可以从繁琐的信息交互中解脱,从而全心关注要完成的任务[1]。正是因为智能空间内设备及其通讯介质的复杂性,使其成为了一个典型的异构网络。为了使这个异构网络的各个终结点能够进行无障碍的信息交互,就必然要将这个异构网络同构化。目前国内外许多典型的智能空间研究项目和协议组织都朝着这个方向努力,例如德州大学阿林顿分校的Mav-Home[2,3];MIT的Home of Future[4];科罗拉多大学ACHE[5];佛罗里达大学移动和普适计算实验室的GatorTech Home[6];中国的闪联[7]等。虽然这些科研项目和协议组织在各自领域都取得了一些成果,但大多都停留在科研阶段,很少存在一个具体的、通用的采用公认协议的通信框架。本文从智能空间的实际通讯需求出发,提出了一种通用的面向智能空间的异构网络同构化通讯框架。

2 智能空间的通讯需求

智能空间是一个宽泛的概念,无法提出一个通用的具体需求。但所有的智能空间的通讯框架都应具有以下几个特性。

动态性:智能空间内的所有设备构成了一个复杂系统,每个设备的行为对整个系统而言是不可预知的,比如故障、重启、新增设备、移除设备等。因此动态性是整个智能空间通讯框架的首要特性。

松耦合性:松耦合性体现在2个方面,即不同设备之间应该是松耦合的;设备和上层系统之间应该是松耦合的。

鲁棒性:智能空间中的某个设备的故障不应影响到其它设备的正常运行。

通用性:智能空间通讯框架应该能够应用在大部分不同类型的智能空间中。

灵活性:智能空间的通讯框架应该在保证统一性的情况下给予设备尽可能多的定义自由度。

在本文描述的智能空间通讯框架中,使用动态上下线机制和动态服务发现机制来构建动态性;使用MAS(Multi-

到稿日期:2011-05-08 返修日期:2011-07-17 本文受国家自然基金项目(60903067),国家高科技研究计划(2007AA04Z218)资助。
杨 溢(1984—),男,博士生,主要研究方向为人工智能、智能决策、信息集成,E-mail:yangyi0252087@163.com;王志良(1956—),男,教授,博士生导师,主要研究方向为人工心理及情感计算、智能机器人学、和谐人机交互、3C融合;王 鲁(1986—),男,博士生,主要研究方向为人工智能、智能决策、信息集成;张富深(1987—),硕士生,主要研究方向为智能决策、信息集成。

Agent System)中间件来保证松耦合性和鲁棒性;使用 SOA 来保持通用性;使用动态代码生成、动态编译以及反射机制来实现灵活性。

3 面向智能空间的异构网络同构化通信框架

3.1 通信框架的整体结构

本文提出的面向智能空间的异构网络同构化通信框架从逻辑上一共分为 5 个层次,自下而上分别为设备层、物理层、代理层、服务层、表现层,如图 1 所示。

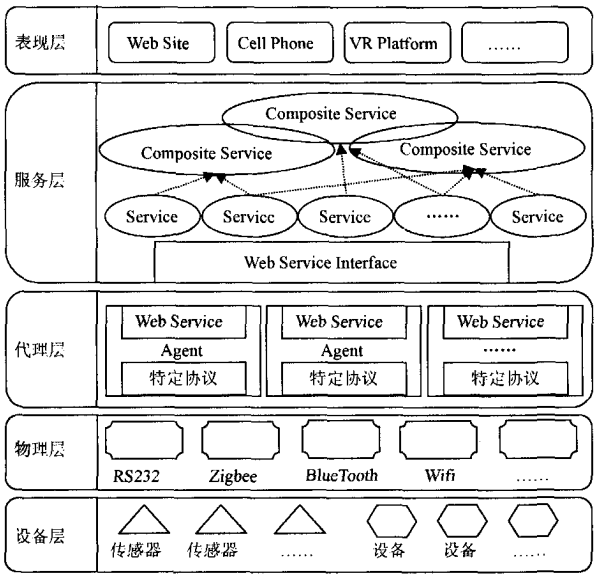

图 1 通信框架的整体结构

设备层由智能空间中所有能与系统进行信息交互的设备组成,是整个系统逻辑结构的最底层。系统的原始信息采集以及系统指令的最终动作都在这一层完成。

物理层由设备层的设备及其之上层次进行通讯的所有物理通讯通道和附加在物理通讯通道上的通讯协议组成。智能空间中各种设备的多样性决定了物理层所包含的通讯介质和协议异常丰富,其基本包含了所有常见的通讯介质和协议,比如 ZigBee、红外、RS232、Bluetooth、Wi-Fi 等。也就是设备层和物理层构成了一个智能空间中的异构网络。

代理层其实可以看作是一个物理通讯通道和通讯协议的转换层。任何一类设备所使用的物理通讯通道在这个层次中被转换成具有承载 TCP/IP 协议能力的物理通道;任何一类设备所使用的通讯协议在这个层次中被转换成 Web Service 协议。如此,一个复杂的异构网络就被转换成相对简单的同构网络,而对设备的功能调用也都统一到 Web Service 接口上,这样就极大地简化了逻辑上层的实现难度,使智能服务成为可能。代理层的转换工作是由协议转换 Agent 来完成的,整个代理层也可以看成是由针对不同类型设备的协议转换 Agent 组成。在实际应用中,需要依据每种类型设备来分别开发一种协议转换 Agent。智能空间中所有的 Agent 组成了该空间的异构网络同构化中间件。总之,代理层将异构网络变成了同构网络。但由于每种 Agent 提供的服务可以是自定义的,因此其服务还是异构的。

服务层将完全不相同的协议转换成了具有统一接口的 Web Service,每一个设备都存在一个与之对应的 Web Service。因此从服务层往下看去,没有设备的概念,只有一个一个的 Web Service。服务层集合了所有的单个服务,在特定的情形下服务层还可以将类似的、有关联的单个服务组成功能更强的组合服务(Composite Service)[8],便于集成调用。服务层的逻辑功能实际上由一个 Agent 网关来完成。总之,服务层将面向一组 Agent 的异构服务调用变成了面向一个 Agent 网关的同构服务调用。

3.2 WCF

各大软件厂商都推出了自己的 SOA 开发工具,例如 Microsoft 的 WCF、IBM 的 WebSphere 系列、BEA 的 AquaLogic 系列、Oracle 的 Oracle SOA Suite、SAP 的 NetWeaver 等[9]。这些开发工具都各有特点和优势。本文提出了基于 WCF 的 Agent 通讯架构。基于 WCF 的 SOA 在智能空间通讯框架的开发上具有如下优势:

操作系统友好性:大部分智能空间(例如智能家居)都不会专门配备专用的服务器,而是直接采用 PC 机。而针对 PC 机而言,Windows 操作系统的应用更为广泛,这就意味着 WCF 的各项功能能够得到更好的支持。

统一性:WCF 统一了 Microsoft 之前的各项通信技术,而且已很好地集成到了 Visual Studio 中,能够快速方便地开发一个面向服务的应用程序。

互操作性:对于某些绑定而言,WCF 采用的是标准的 Web Service 协议,这就意味着使用 WCF 编写的应用程序可以被跨平台调用。

安全性:WCF 很好地支持了 WS-Security,WS-Trust 和 WS-SecureConversation 等安全策略。

兼容性:WCF 能很好地兼容早期的版本。

3.3 基于软件 Agent 的异构网络同构化中间件

本框架中描述的异构网络同构化中间件由所有独立的软件 Agent 组成。软件 Agent 是一类在能特定环境下感知环境并能自治地运行以代表其设计者或使用者实现一系列目标的计算实体或程序[10]。正是因为软件 Agent 具有的自治性和独立性,使其能够很好地满足本框架的强鲁棒性和松耦合性的需求。和一般的 Agent 行为分类类似[11],符合本框架定义的软件 Agent 如果只具有应答其余 Agent 请求的能力,那么称之为反应型 Agent;如果只具有请求其它 Agent 的能力,那么称之为主动型 Agent;如果兼有上述两种的能力,那么称之为混合型 Agent。本文建立的 Agent 的一般组成如图 2 所示。

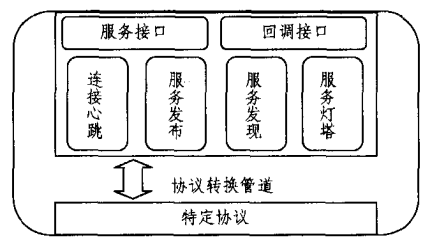

图 2 Agent 的一般组成

3.3.1 协议转换管道

协议转换管道提供了一种从特定协议到 Web Service 接口的协议映射方式。协议的映射方式根据具体的特定协议的不同而不同。

3.3.2 连接心跳、服务发布、服务发现及服务灯塔

连接心跳：为测试和维护链路通常所设置的心跳测试，类似于传输层协议中的 KeepAlive。本框架定义的 Agent 的心跳间隔设置为 30s。

服务发布：在特定的地址将 WSDL 和元数据发布给可能的调用客户端。本框架定义的 Agent 同时采用 WCF 中的两种服务发布机制：HttpGet 和 MexMetadata[12]，并同时使用服务灯塔在 UDP 端口 8800 组播服务发布地址信息。

服务发现：如果 Agent 为主动型和混合型 Agent，那么就需要能够动态发现其它 Agent 的服务信息，能够主动下载所发现的服务信息，并将其动态编译和加载到当前的引用程序集中。

服务灯塔：本框架的服务发现机制依赖于服务灯塔。Agent 开启服务后，服务灯塔每隔 15s 在 UDP 端口 8800 组播 Agent 在线宣告。在线宣告为一个 XML 数据包，其中包含了该 Agent 上线时间、WSDL 地址等信息。如果一个已经上线的 Agent 在 30s 内没有发出其在线宣告数据包，那么其他 Agent 可以认为该 Agent 已经下线。

3.3.3 Agent 的服务接口

表 1 显示了 Agent 的服务契约。某个特定 Agent 的服务契约由两部分组成：IAgentService 是所有符合本框架定义的 Agent 所必须满足的接口定义；IAgentSelfService 是每个特定 Agent 的自定义服务接口，Agent 开发者可以根据所开发的 Agent 的具体需求来定义该接口。Agent 的两种服务契约接口设计既维护了系统的统一性，又保持了系统的灵活性。

表 1 IAgentService 接口定义

操作	作用
HeartBeat	测试和维护连接状态心跳
SubscribeStatus	订阅该 Agent 状态改变消息
UnSubscribeStatus	取消订阅该 Agent 状态改变消息
GetAllStatus	获得该 Agent 的所有操作状态列表
GetStatus	获得该 Agent 的指定操作状态
GetDeviceInfo	获得该 Agent 的基本信息
IAgentSelfService	定义可选实现的自定义 Agent 服务接口

3.3.4 Agent 的回调接口

Agent 除了一问一答的传统 SOA 服务响应方式以外，还具有类似订阅信息的信息推送能力，这种能力由 Agent 的回调契约来实现。Agent 回调接口提供了一个回调方法：StatusChanged。该方法在该 Agent 的某个状态被更改的时候调用，用以通知已经订阅了该回调服务的其它 Agent。回调机制极大地降低了 Agent 之间由于频繁的轮询状态而造成的系统开销，同时提高了状态改变事件的实时性。

3.3.5 Agent 的服务发布方式

Agent 的服务结构如表 2 所列。由于 Agent 的调用和被调用的对象比较单一，而且回调机制需要双向的通信支持，同时只能在智能空间内部使用，因此本框架所定义的 Agent 提供了一个使用 wsDualHttpBinding 的服务终结点。

表 2 Agent 的服务结构

服务	终结点	绑定	服务契约
AgentService	HttpPortal	wsDualHttpBinding	IAgentService

3.4 Agent 网关

从功能上来说，Agent 已经能够很好地完成从异构网络到同构网络的转换工作，但同时存在一些问题。

（1）MAS 固有的问题：基于 Agent 的异构网络同构化中间件实质上是一种 MAS（多智能体系统），这在带来统一性、灵活性、松耦合性以及鲁棒性的同时，也带来了 MAS 固有的挑战，例如多 Agent 之间的协作问题、行动同步问题、决策冲突问题等[13]。

（2）服务的不确定性问题：本框架定义的 Agent 具有极大的灵活性，对于上层系统而言，这种灵活性就将变成对提供服务的不确定性，例如服务的类型不确定、服务的地址不确定、服务的内容不确定等。而 SOA 要求服务在调用前必须是确定的[14]。

（3）安全性问题：每个 Agent 都向外发布了一个特定的 Web Service，而这需要占用一个端口，由此将会有很多的服务发布从很多的端口发布出来。如果需要从外部网络访问这些服务，就必须让防火墙打开这些端口，这样有可能带来很高的安全风险。

为了解决以上 3 个问题，本框架在代理层（Agent 层）上设计了服务层（Agent 网关层）。

3.4.1 Agent 网关的组成

图 3 显示了 Agent 网关的逻辑结构。Agent 网关中的服务灯塔（ServiceBeacon）内部具有一个采用 LRU 算法[15]的上线 Agent 缓存。服务灯塔在执行的组播端口监听在线宣告、下线宣告以及判断超时下线，并且将结果告知 Agent 工厂（AgentFactory）。Agent 工厂负责生成新上线的 Agent 代理实例，即调用相应 Agent 服务的客户端。生成 Agent 代理实例需要经过下载 WSDL（WSDLImporter）、导入 WSDL（WSDLImporter）、生成代理代码（CodeDomProvider、AgentCodeModifier）、动态编译代码并加载程序集（AgentCodeComplier）、实例化 Agent 代理并注册回调（AgentActivator）等主要过程。新生成的 Agent 代理实例随后被添加到了一个实例仓库（AgentStore）中，实例仓库采用一个 Key-Value 类型的哈希表缓存来保存新生成的实例，并且把这个缓存的引用交给 Agent 代理实例桥接对象（AgentBridge）。Agent 代理实例桥接对象负责处理来自 Agent 网关服务接口的请求和 Agent 代理实例之间的交互。

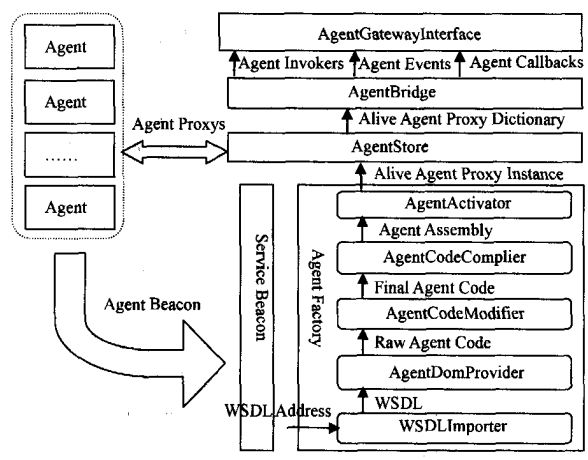

图 3 Agent 网关的组成

3.4.2 Agent 网关的服务接口

由于 Agent 网关可能会被某些不支持双工通信的客户端调用，因此 Agent 网关将服务契约接口定义拆分为两部分：IBasicGatewayService，其包含了最基本的功能实现，如表 3 所

列；IGatewayService(IGatewayService 继承于 IBasicGatewayService)，其包含了高级的回调实现，如订阅与取消订阅 Agent 在线情况、订阅与取消订阅制定设备 ID 的 Agent 的状态情况。IGatewayService 除 IBasicGatewayService 接口外，其它接口定义如表 4 所列。

表 3 IBasicGatewayService 接口定义

操作	作用
GetAgentGatewayInfo	获得当前 AgentGateway 的基本信息
GetAliveAgentCount	获得当前所有在线 Agent 数量
GetAgentInfo	获得指定 Agent 的设备信息
GetAliveAgentInfos	获得当前所有在线 Agent 的设备信息
GetAgentOperations	获得指定 Agent 的方法列表
GetAgentOperation	获得指定 Agent 的指定方法名称的方法信息
InvokeAgentOperation	调用指定 Agent 的指定操作

表 4 IGatewayService 除 IBasicGatewayService 接口外，其它接口定义

操作	作用
SubscribeAgentAlive	订阅 Agent 在线情况
UnSubscribeAgentAlive	取消订阅 Agent 在线情况
SubscribeAgentStatus	订阅指定设备 Agent 的状态情况
UnSubscribeAgentStatus	取消订阅指定设备 Agent 的状态情况

3.4.3 Agent 网关的回调接口

与 Agent 不同，Agent 网关的调用者不但关心 Agent 的状态变化，而且还关心 Agent 的在线情况。因此除了有 Agent 上线或下线触发的 AgentAliveChanged 回调方法外，Agent 网关还增加了一个 AgentStatusChanged 回调方法，在 Agent 状态发生变化时触发此回调方法。

3.4.4 Agent 网关的发布方式

由于 Agent 网关大部分面向外部网络的调用者，因此必须考虑各种内外网客户端的调用能力差异和需求差异，具体表现在：

(1)调用客户端的调用可能产生跨域访问的问题。

(2)调用客户端可能仅支持 Soap1.1 而不支持 Soap1。

(3)调用客户端可能不支持某些高级的安全机制。

(4)调用客户端可能不支持包括回调在内的双工通信机制。

(5)基于上述原因，Agent 网关提供了丰富的服务类型和绑定类型，来承载 3 种不同类型服务的 4 种终结点，并分别使用 4 种不同的绑定方式，如表 5 所列。

表 5 Agent 网关中提供的服务、终结点及绑定方式

服务	终结点	绑定	服务契约
DomainService	DsPortal	webHttpBinding	IDomainService
BasicGateyService	BasicPortal1	basicHttpBinding	IBasicGatewayService
	BasicPortal2	wsHttpBinding	IBasicGatewayService
GatewayService	GatewayPortal	wsDualHttpBinding	IGatewayService

4 系统性能研究

4.1 相关工作

我们使用本文所描述的框架构建了一个智能家庭的实际环境(LookeyHome，有关本项目的介绍和视频可以直接从百度搜索)，并在这个环境中进行了相关的性能试验。

4.2 测试环境

测试环境：使用一台 PC 主机和该主机上的一个虚拟机进行了以下试验。

主机：Windows XP sp3；Intel core 2Duo E7300 2.66 GHz；3G Ram。

虚拟机：Windows XP sp3；Intel core 2Duo E7300 2.66 GHz；1G Ram。

主机上运行 Agent 和 Agent 网关，虚拟机上运行调用测试程序，调用测试程序使用 Visual Studio 2010 的 Debug 模式。

所有程序均关闭 WCF 本身的限流策略。

为了排除网络数据传输带来的延时和数据量大小不同带来的干扰，在性能测试部分均调用服务操作 SetTemperature(设定温度)，该操作具有一个 float 类型的参数，返回值类型为 float。性能测试部分均使用 1、10、20、50 线程，每线程分别执行 5、10、100、500、1000 次的测试方式。

4.3 Agent 和 Agent 网关通信管道建立耗时分析

Agent 网关在收到第一次上线的 Agent 的在线宣告后，会通过 Agent 工厂采取一系列的过程来生成一个可用的 Agent 代理实例。这个过程包括：下载 Wsdl->导入 Wsdl->生成代理代码->编译代理代码->实例化代理->打开连接通道->注册状态回调。图 4 显示了这个过程的耗时曲线。从图 4 中可以看出，第一次产生 Agent 代理实例的过程比之后的过程要耗时得多，特别是下载 Wsdl 部分。这是因为在第一次尝试下载 Wsdl 时，Agent 会根据自身的契约生成一个 Wsdl 和元数据文档，这就需要 WCF 的底层程序扫描整个 Agent 程序集中的服务契约、操作契约、数据契约和回调契约，这是一个十分耗时的过程。而当 Wsdl 和元数据文档一旦生成，WCF 将自动缓存这些数据，因此在之后的请求中不必重新生成这些数据，从而加快了访问速度。图 5 展示了整个过程的平均用时分布(排除了第一次过程的数据)。从图中可知，导入 Wsdl 和编译代码两项一共占用了大约 85%的时间。和第一次下载 Wsdl 非常耗时类似，这两项的时间损耗在实际中也是可以避免的，例如在 Agent 网关端采用缓存策略就能很好地解决这个问题。

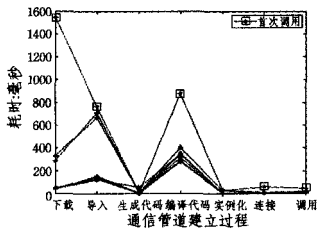

图 4 网关与 Agent 建立通信管道耗时分析(a)

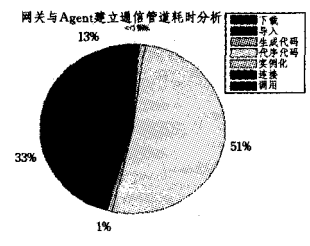

图 5 网关与 Agent 建立通信管道耗时分析(b)

4.4 网关 3 种绑定方式建立连接的平均耗时对比

在网络应用程序的实践中，存在 2 种基本的链路维护模式：长连接和短连接。长连接是指客户端一旦连接上服务器，直到调用过程结束前都不断开通信链路。这样做的优点是减

少了反复建立通信链路的系统开销，其缺点是服务器端将承受更大的压力，因为维护通信链路需要花费额外的 CPU 和 RAM；短连接是指每次请求都重新建立一个连接，通信完成后立即销毁该连接。短连接的优缺点正好和长连接相反。因此在实际的系统中采用哪种链路维护模式，需要根据系统的实际来取舍。图 6 显示了 Agent 网关的 3 种绑定下 1000 次单线程同步连接调用的平均耗时。很明显，wsDualHttpBinding由于需要建立 2 条 Http 通信链路(模拟双工)，因此比另外两种绑定方式耗费更长的时间，所以在采用 wsDualHttpBinding 连接方式的客户端最好采用长连接的方式(实际上是必须采用长连接的方式，因为回调上下文需要相同的链路)。wsHttpBinding 的连接平均耗时仅是 dual 方式的 1/3，所以采用这种方式连接时，如果调用很频繁，建议使用长连接；反之使用短连接。basicHttpBinding 的一次平均连接耗时不到 0.5ms，应该采用短连接。

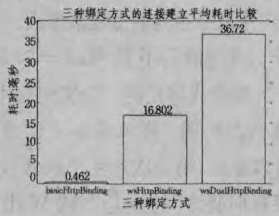

图 6　3 种绑定方式的连接建立平均耗时比较

4.5　Agent 服务的并发性能

Agent 多线程并发耗时测试如图 7 所示，Agent 多线程并发吞吐量测试如图 8 所示。从图 7 和图 8 中可以得到以下结论：

(1)低线程数时，随着执行次数的增加，吞吐量逐渐增加，最终将稳定在某个吞吐量极限。

(2)在高线程数、低执行次数时，吞吐量较小，但其在一定范围内随着执行次数的增加而增加很快，超过该范围后增加变缓，最终将稳定在某个吞吐量极限。

(3)存在最佳线程数量区间，线程数量在这个区间内将拥有最大极限吞吐量。从实验数据来看，这个区间大致在 10～20。

(4)最佳线程数量区间虽然拥有最大极限吞吐量，但在低执行次数时吞吐量反而很低。

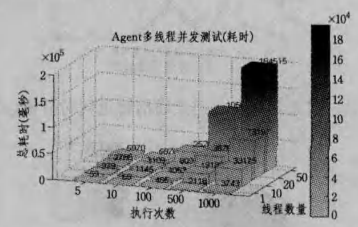

图 7　Agent 多线程并发测试(耗时)

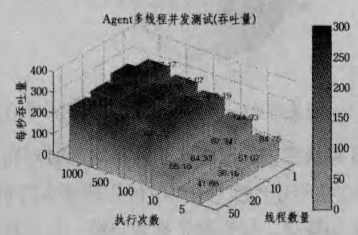

图 8　Agent 多线程并发测试(吞吐量)

4.6　通过网关中转的服务并发性能

网关中转多线程并发 basicHttpBinding 耗时测试如图 9 所示，网关中转多线程并发 basicHttpBinding 吞吐量测试如图 10 所示。从图 9 和图 10 中可以得到以下结论：

(1)低线程时，随着执行次数的增加，吞吐量变化不大。

(2)存在最佳线程数量区间(10～50)和最佳执行次数区间(10～100)。

(3)在高执行次数时(>100)的吞吐量都小于低执行数时的吞吐量。

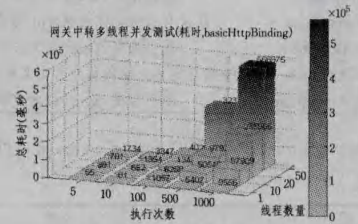

图 9　网关中转多线程并发 basicHttpBinding 耗时测试

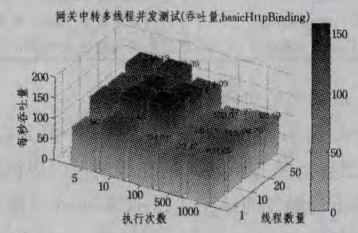

图 10　网关中转多线程并发 basicHttpBinding 吞吐量测试

网关中转多线程并发 wsHttpBinding 耗时测试如图 11 所示，网关中转多线程并发 wsHttpBinding 吞吐量测试如图 12 所示。从图 11 和图 12 中可以得到以下结论：

(1)吞吐量随着线程数量的增加有缓慢上升趋势。

(2)在执行次数为 100 左右时，在每个线程数量档次上都拥有最大吞吐量。

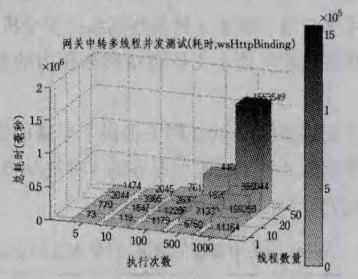

图 11　网关中转多线程并发 wsHttpBinding 耗时测试

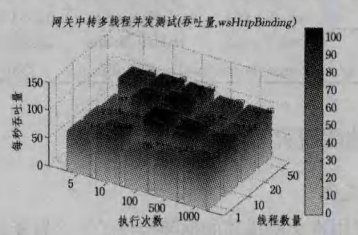

图 12　网关中转多线程并发 wsHttpBinding 吞吐量测试

网关中转多线程并发 wsHttpBinding 耗时测试如图 13 所示，网关中转多线程并发 wsHttpBinding 吞吐量测试如图 14 所示。从图 13 和图 14 中可以得到以下结论：

(1)每个执行次数档次的吞吐量随着线程数的增加而降

低。

(2)单线程时,随着执行次数的增加,吞吐量很快稳定在一个最大极限。

(3)每线程执行 100 次左右时的吞吐量在每个线程数量档次中都具有最大值。

(4)过高的并发线程数会导致 Agent 网关抛出"ServerTooBusyException"(50 线程时无测试数据,实际测试中当并发线程数超过 40 时,就会发生这个问题)。

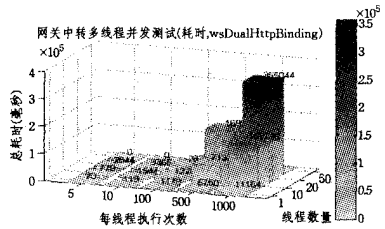

图 13 网关中转多线程并发 wsDualHttpBinding 耗时测试

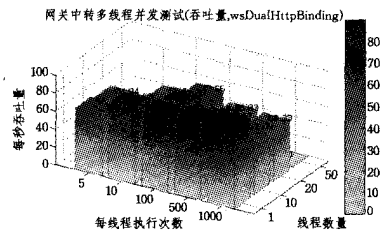

图 14 网关中转多线程并发 wsDualHttpBinding 吞吐量测试

4.7 网关中转通信时的吞吐量性能消耗

在经过 Agent 网关中转时,数据包将多经过一次数据包拆包、装包、网络通信的过程。因此通过 Agent 网关中转时将不可避免地带来性能损失。表 6 表明,从 basicHttpBinding 到 wsHttpBinding 再到 wsDualHttpBinding 的变化过程中,包装的信息量不断增多,吞吐量下降明显,吞吐量性能从直接调用 Agent 的 66%递减到 38%。可以说,越复杂的绑定方式在带来了诸如回调、安全性等好处的同时,性能损耗也值得注意。不过,从另一方面说,越复杂的绑定方式带来了一定程度上系统吞吐量的相对均衡,这体现在标准差指标从直接调用 Agent 时的 93.44 递减到 Dual 的 9.59,降低了大约 90%。均衡的吞吐量能够保证系统不会频繁地遭受到瞬间的冲击,从而降低了运行时服务阻塞的风险。

表 6 网关中转通信时的性能消耗(吞吐量)

线程数	Agent		basicHttpBinding		wsHttpBinding		wsDualHttpBinding	
	均值/标准差	性能	均值/标准差	性能	均值/标准差	性能	均值/标准差	性能
1	187.8/73.2	1.0	101.2/13.5	0.54	75.5/8.1	0.40	82.8/8.3	0.44
10	189.5/112.3	1.0	131.4/25.8	0.69	86.8/11.7	0.46	68.8/7.6	0.36
20	178.5/118.0	1.0	122.3/28.3	0.68	85.3/14.4	0.48	56.1/12.9	0.31
50	163.7/95.1	1.0	116.6/32.6	0.71	88.5/5.1	0.54	null	null
合计	179.9/93.44	1.0	117.9/26.4	0.66	84.0/10.9	0.47	69.2/9.6	0.38

虽然在通过网关中转时会产生一定的性能损耗,但是就大部分智能空间系统而言,几乎不会遇到短时间高并发的访问冲击,每秒至少 70 次的吞吐量对于智能空间系统而言应该是足够冗余了。

结束语 本文详细介绍了一种面向智能空间的异构网络同构化通信框架的构建方法,这种通信框架同时具有动态性、松耦合性、鲁棒性、通用性、灵活性等特点。使用 Lookey-Home 的实际环境进行了一系列多角度的多线程并发测试,并给出了测试数据和性能结果。测试的结果表明,即便是采用吞吐量最低的基于 wsDualHttpBinding 的网关中转通信,系统也能至少承受每秒大约 70 次的并发访问,这个性能足已胜任大多数智能空间的通讯需求。下一步将在框架中应用安全策略,并研究如何将通信框架与本体语义相结合。

参 考 文 献

[1] 况晓辉,黄敏桓,金旗,等. 基于 SOA 的可信智能空间系统软件研究[J]. 计算机科学,2010,37(1):34-38

[2] Karthik G, Diane J C. Online Sequential Prediction via Incremental Parsing: The Active LeZi Algorithm [J]. IEEE Intelligent Systems, 2007, 22(1): 52-58

[3] Abhishek R, Sajal K D, Kalyan B. A Predictive Framework for Location-aware Resource Management in Smart Homes [J]. IEEE Trans on Mobile Computing, 2007, 6(11): 1270-1283

[4] Stephen S I. Designing a Home of the Future[J]. IEEE Pervasive Computing, 2002, 1(2): 76-82

[5] Mozer M. The Neural Network House: An Environment that Adapts to Its Inhabitants[C]// Proc. of the AAAI Spring Symposium on Intelligent Environments. 1998: 110-114

[6] Helal A, Mann W, El-Zabadani H, et al. The gator tech smart house: A programmable pervasive space[J]. IEEE Computer, 2003, 38(3): 50-60

[7] 谢侃. 基于 IGRS 闪联协议的智能组网方法研究及应用[D]. 广州:华南理工大学,2010

[8] Chen W, Xu Y, Peng B, et al. Dynamic monitor based service recovery for composite service in MANETs[C]// 11th IEEE International Conference on Communication Technology. 2008: 557-560

[9] 邢少敏,周伯生. SOA 研究进展[J]. 计算机科学,2008,35(9):13-20

[10] 刘大有,杨鲲,陈建中. Agent 研究现状与发展趋势[J]. 软件学报,2000,11(3):315-321

[11] 何炎祥,陈莘萌. Agent 和多 Agent 系统的设计与应用[M]. 武汉:武汉大学出版社,2001

[12] Pablo C, Kurt C, Fabio C, et al. Professional WCF 4: Windows Communication Foundation with. NET 4 [M]. WROX PR/PEER Information INC, 2010

[13] Katia P S. Multiagent Systems [J]. American Association for Artificial Intelligence, 1998, 19(2): 79-93

[14] Erl T. Service-oriented Architecture: Concepts, Technology, and Design [M]. London: Prentice Hall PTR, 2005

[15] 张震波,马振华. 基于 LRU 算法的 Web 系统缓存机制[J]. 计算机工程,2006(19):68-70