doi:10.3969/j.issn.1000-436x.2012.z1.010

基于 RapidIO 的 GIOP 协议——RIO-IOP

陈文宇，曾茹，皮维，李文
（电子科技大学 计算机科学与工程学院，四川 成都 611731 ）

摘 要：在嵌入式系统应用中，所有硬件元件都是基于总线方式连接的，基于 TCP/IP 协议无法满足它们之间的通信，特别是在芯片间及板间的互联传输。因此在嵌入式环境中，基本采用 RapidIO 互联架构，而基于 RapidIO 的 GIOP 映射还处于空白阶段，借鉴 IIOP 思想，提出 GIOP 到 RapidIO 的映射：RIO-IOP。从网络层次模型、传输层组件模型等方面对 RIO-IOP 进行全面解析，并提出 RIO-IOP 对 CORBA 基本服务的支持。对 RIO-IOP 进行了实验，证明了在嵌入式领域和军事环境的硬件设备中，具有明显的优势和应用前景。

关键词：计算机系统结构；RIO-IOP；CORBA；IIOP；RapidIO
中图分类号：TP302 **文献标识码**：A **文章编号**：1000-436X(2012)Z1-0070-09

Protocol of GIOP onto RapidIO —— RIO-IOP

CHEN Wen-yu, ZENG Ru, PI Wei, LI Wen
(School of Computer Science and Engineering, University of Electronic Science & Technology of China, Chengdu 611731,China)

Abstract: In embedded system application, all hardware elements based on bus mode. TCP/IP can not satisfy their communications, especially the interconnection of chips and boards. Therefore, in embedded environment, RapidIO interconnection framework is employed. There is not the mapping of GIOP onto RapidIO. This paper taked example by IIOP, and proposed the mapping of GIOP onto RapidIO:RIO-IOP. It analyzed RIO-IOP from the network model and the transport component model aspects in an all-round. RIO-IOP also supported the basic service of CORBA. In embedded field and military environment in the hardware, there is obvious advantages and application prospect by RIO-IOP experimentation.

Key words: computer architecture; RIO-IOP; CORBA; IIOP; RapidIO

1 引言

RapidIO[1]是一种高速、分组交换、全双工的互连体系结构，它的提出是针对在芯片间及板间进行数据和控制信息的传输[2,3]，其传输效率高于以太网，在嵌入式系统领域具有明显的优势。

通用 ORB 间协议（GIOP, general Inter-ORB protocol）为 ORB 之间的交互详细规定了一套标准的传输语法（低层次的数据表达）和一系列消息格式。GIOP 是为 ORB 到 ORB 之间的交互而创建的，它直接工作在任何满足规定的面向连接的传输协议上[4]。它使 CORBA 可以在不同操作系统和编程语言的环境下实现客户和服务器对象的互操作[5]。

在 CORBA 规范中，对于 GIOP 自身而言，它不提供完整的交互功能，它必须被映射为具体的协议。GIOP 到 TCP/IP 存在映射 IIOP，但是 TCP/IP 应用在嵌入式系统中有明显的局限性，因此越来越多的嵌入系统采用 RapidIO 技术，GIOP 到 RapidIO 之间映射的研究还处于空白阶段。本文在研究 RapidIO、GIOP 和 IIOP 的基础上，从理论和实践上提出并实现了全新的 RIO-IOP 协议（它是 GIOP 到 RapidIO 的一种映射），它能完成经由 RIO-IOP 协议的 CORBA 基本调用。

收稿日期：2012-08-06

2　GIOP 协议和 IIOP 协议

CORBA 参照模型包括：ORB 内核(ORB core)[6]、对象适配、客户端存根（IDL stub）和服务端框架（IDL skeleton）、动态 DII/DSI、CORBA 客户端和服务端、ORB 之上的服务（包括名字服务，事件服务）、底层驱动等。

从 CORBA 的体系中可以看出，CORBA 客户端程序与服务端程序进行数据通信的基础是建立了统一的 GIOP 协议以及统一的数据通信架构。客户端对不同地址空间中的远端服务对象的使用过程，如同从本地的地址空间一样方便。

GIOP 协议内容包括：公共数据表示（CDR, common data representation）、GIOP 消息格式、GIOP 消息传输、IIOP 协议（internet ORB 间协议）以及双向 GIOP（Bi-Directional GIOP）[7]。

GIOP 定义了不同 ORB 间互操作的协议，它提供一个抽象协议规范，能被映射为常见的面向连接的传输协议。应用最广泛的因特网 ORB 间协议（IIOP, internet Inter-ORB protocol）就是 GIOP 消息传输到 TCP/IP 连接的映射。在基于网线传输的机制与 TCP/IP 协议基础上，IIOP[8]协议就是具体的 GIOP 实现。但是由于过多的编码/解码，数据复制以及高阶的功能调用，IIOP 协议在高速网络中表现出较低的性能[9]。

3　RIO-IOP 协议的分析

3.1　基于 RapidIO 的协议栈 RCS

基于 RapidIO 的协议栈（RCS, RapidIO communication system）是在分布式信号与信息处理机处理器间，利用 RapidIO 网络底层硬件通信机制和操作系统的资源调度策略，实现的为应用程序提供数据传输服务。

RCS 协议栈具有如下特性：面向连接、全双工、对称、可靠传输以及支持字节流。它是基于 RapidIO 总线驱动所做的一种协议栈，该协议栈对外暴露以下接口。

1) 连接的配置接口。

2) 基于配置好的连接发送数据和接收数据的接口。

3) 提供连接上的事件的捕获接口。

3.2　RIO-IOP 网络层次模型

在以 RapidIO 总线为基础的网络层次模型中，由于 RapidIO 网络终端节点的互联互通是采用 RapidIO 接口接入板卡上 RapidIO 交换芯片，RapidIO 交换芯片的物理链路再接入交换网络中，网络中的交换模块负责维护点对点通信的路由信息，因此 OSI 参考模型中网络层的功能是由 RapidIO 硬件实现的。RapidIO 是快速的数据传输的一种新型总线，在此基础上，RCS 协议栈是对 RapidIO 的第一次封装，运行在 RapidIO 的网络中的各个非交换节点上，通过 RCS 可以实现节点间的快速通信与数据交换。为了能够将 CORBA GIOP 协议运行在 RapidIO 总线连接的网络中的非交换节点上，需要对 RapidIO 进行二次封装，即把 RCS 封装成 RIO-IOP 协议。RIO-IOP 协议使得所有的 GIOP 消息可以通过 RCS 协议栈发送和接收。 RIO-IOP 协议与 IIOP 协议以及 ISO/OSI 层次模型的对比如图 1 所示。

图 1　网络层次模型对比

3.3　传输层组件模型

ORB 传输组件是负责将数据按照 GIOP 协议的格式打包，并负责数据的接收与发送，同时负责连接服务端的任务调度以及下层的协议栈和驱动。

ORB 传输层组件模型如图 2 所示。

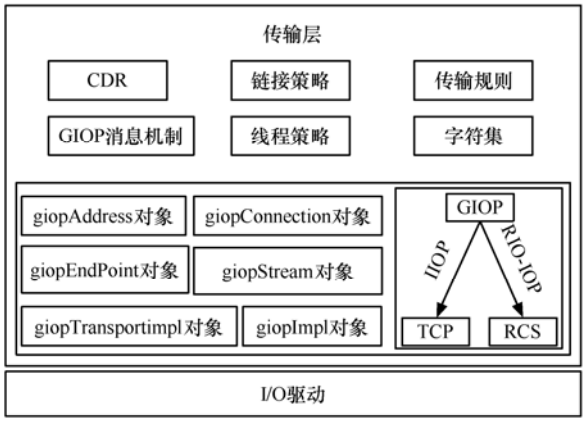


图 2　ORB 传输层组件模型

1) 服务端组件模型

服务端的传输层组件包含传输模块初始化组件、对同一目标地址空间的连接分组组件、遵循 GIOP 协议的 GIOP 消息机制组件、实现在 RapidIO 总线上的 GIOP 的映射 RIO-IOP 组件 、线程策略组件、协议包的封装组件 CDR 和传输规则组件。

2) 客户端组件模型

客户传输模块包括传输模块初始化组件，连接分组组件，GIOP 消息机制组件以及 RCS 协议的 GIOP 映射 RIO-IOP 组件。与服务端不同的是客户端不需要在连接侦听数据事件。在客户端呼叫服务对象的过程中，客户端对象将通过 CDR 对象将请求封装成 GIOP 协议格式，发送到服务端。

3) 传输层外部接口

传输层提供给的外部接口包括：connect（连接接口），acceptAndMonitor（监控连接事件接口），monitor（监控连接接口），notifyReadable（通知可读接口），sendRequest（发送请求接口），receiveReply（接收反馈接口），receiveReques（接收请求接口），sendReply（发送反馈接口），send（数据发送接口），recv（数据接收接口）。

3.4 互操作对象引用 IOR

1) IOR 的组成

客户端通过对象的引用与服务端建立通信联系，服务端在创建服务对象的时候，根据对象创建时候的上下文关系，将这些信息发布出来给客户端用于与伺服对象建立关联。

一个对象引用 IOR 中包含了多个 taggedProfile 对象，每一个 taggedProfile 对象是包含了用来联系远程对象的多种协议方式。比如，在客户端既可以通过 RIO-IOP 协议来与远程对象建立通信联系，也可以通过其他协议（如 IIOP）与远程对象建立通信关系。

每一种 taggedProfile 对象中包含了一个 ProfileId，如何对该对象的编解码进行解析，就依赖该标识。比如客户端按照 ProfileId 定义的一种方式将数据分组进行编码，服务端解码的过程就需要通过该 ProfileId 来解码。

在实际应用中，很可能一个服务器端的某个对象的互操作对象引用（IOR）会被其他不同 CORBA 厂商的客户端所使用。因此，IOR 采用 IDL 结构来定义的。

根据 RCS 网络节点间通信的机制，定义 RIO-IOP IOR Profile IDL 的结构为

```
Module IOP{
    typedef unsigned long ComponentId;
    const ProfileId TAG_RIO_IOP = 2;
    struct TaggedComponent{
        ComponenteId tag;
        sequence<octet> component_data;
    };
Module RIO_IOP{
struct Version{
    octet major;
    octet minor;
};
struct RIO_IOPProfileBode_1_2{
    Version             riop_version;
    string              riop_host;
    unsigned shot       riop_port;
    sequence<octet>     object_key;
    sequence<IOP::TaggedComponent>
components;
    };
```

使用 RCS 通信的对象，在对象 IOR 中提供了基于 RIO-IOP 的联系方式。riop_host 表示对象所在连接的目的地址，也就是 RapidIO 网络地址。riop_port 表示对象所在连接的目的端口。object_key 用来表示远程地址空间的对象。

2) CORBA 客户端通过 IOR 调用服务端服务

CORBA 客户端调用服务端对象服务，是基于 RIO-IOP 协议通过 IOR 实现的。假设某个服务对象 A 的 IOR 提供了多种调用方式。通过 RapidIO 总线，目标对象 RapidIO 网络地址为节点标识 HOSTA，端口为 PORT1，对象引用键值为 OBJECT_KEY。

通过 TCP/IP 协议，主机为 HOST B，端口为 PORT2，对象键值为 OBJECT_KEY。通过 ATM 协议，主机为 HOSTA_ATM，接入点为 SAP1，对象引用键值为 OBJECT_KEY。使用 RIO-IOP 连接服务端如图 3 所示。

图 3 中，对象 A 的 IOR 当前使用的为：RIO-IOP://HOSTA:PORT1/OBJECT_KEY，该对象中包含了联系服务端上的远程对象 A 的具体方式。通过 taggedProfile 的信息可以知道，该对象位于节点标识为 HOSTA，服务端口为 PORT1 的端口上，通过 RIO-IOP 协议与服务器联系，在服务端使用 OBJECT_KEY 索引对象的实现。

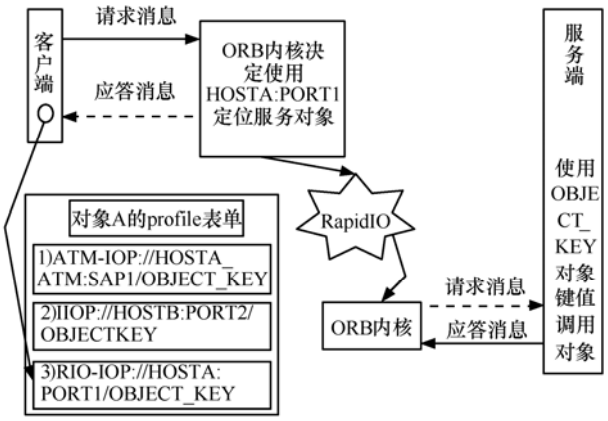


图 3　使用配置的 RIO-IOP 连接服务端

3.5　连接策略

IOR 中承载的目标地址信息是用来建立或者选择连接通道用的。服务端将创建好的服务对象通过 IOR 的方式发布出去。客户端可以实现在目标地址不可用的情况下，自动尝试配置好的下一个联系方式。

客户端与服务端以 IOR 形式通过网络连接进行数据的通信，在 ORB 中连接的方式，根据呼叫发起方的差异，以及连接复用的方式，将连接划分以下 3 种。

1) 单向连接上多重呼叫（MultiplecallPer Connection）

2) 单向连接上的单重呼叫（onecallPer Connection）

3) 双向连接（bidirectionConnection）

3.6　消息机制

在 RIO-IOP 消息机制中，客户端通过请求消息（request）定位到需要访问的对象，然后使用已经配置的连接。服务端在找到对应的实现后，通过应答消息（reply）将执行结果按照应答消息的包头以及消息体的格式，在外层加上 GIOP 协议头后传输给对应的客户端。请求消息与应答消息通过请求的标识 ID 进行关联。

客户端与服务端进行信息的交流逻辑如表 1 所示。

3.7　基于 RIO-IOP 的连接过程

为了能够通过 RIO-IOP 的方式访问服务对象，必须满足 2 点前置条件：识别 RIO-IOP 协议的 IOR 字符串和注册 RIO-IOP 传输到 ORB 内核中。对于第 1 点，需要在初始化以及 IOR 句柄解析的时候增加 RIO-IOP 对应的部分。对于第 2 点，需要新增 RIO-IOP 相关的协议支持接口以及关联部分的修改。RIO-IOP 协议是 GIOP 协议在 RCS 协议上的实现，需要继承原有 GIOP 中关于数据传输的所有类，使得 GIOP 可以在 RCS 协议栈上运行。因此需要新增的 9 个对象如表 2 所示。

表 1　　**客户端与服务端消息交流逻辑**

消息类型	消息发起者	说　明
请求消息(Request)	客户端	客户端调用服务器端的服务时，需要发送请求消息，该消息携带和操作相关的信息。请求消息消息头中包含了服务上下文、请求标识、应答模式、对象键值、操作名称等信息。操作名称用于识别操作属性和目标对象。请求消息体内包含了请求相关的输入输出参数，以及附加服务上下文
应答消息(Reply)	服务端	在需要应答时，服务器端会发送应答消息给客户端，消息内包含有和该请求相关执行结果、参数等信息。根据操作结果的不同，后续的应答报文格式和长度也会不同
取消请求消息(CancelRequest)	客户端	客户端对于远程调用可能使用一个指定的超时时间。当指定的时间到达时，如果客户端还没有收到应答消息，这时可以使用取消请求消息来取消之前的调用。在发出取消请求消息后，客户端不再处理被取消的消息应答
定位请求消息(LocateRequest)	客户端	GIOP 对于底层传输的假设是面向连接的。在进行大数据量的交互时，如果不确定服务对象是否可用，而直接进行服务请求的调用，可能造成大量无效数据在传输层的占用，影响传输效率。因此 GIOP 提供了一种类似 ping 的功能，通过发送定位请求消息，可以探测服务是否可用
定位应答消息(LocateReply)	服务端	与定位请求消息对应的反馈
关闭连接消息(CloseConnection)	服务端	允许关闭连接，比如关闭处于空闲状态的连接。例如一个客户到服务器的连接空闲一段时间后，服务器端决定关闭连接，但是客户端可能也会发出关闭连接消息。为了防止上述假象同时发生，服务器在关闭连接前，会先给客户端发一条关闭连接消息。当客户端收到该消息时，客户端认为服务器对于它之前发出的而未收到的应答的请求不再响应，使用新的连接来进行之前的请求，避免了连接限于死等的状态
消息错误(MessageError)	客户端/服务端	如果收到了非 GIOP 格式的消息时，会应答消息错误消息，以告知对端消息错误

表 2　新增对象

rioTransportImpl 对象	rioActiveCollection 对象
rioAddress 对象	rioConnection 对象
rioHolder 对象	rioActiveConnection 对象
rioCollection 对象	RIO_IOP 对象
rioEndPoint 对象	

另外，为了识别 RIO-IOP 协议的 IOR 字符串，需要在 RIO-IOP 设计时增加 RIO_IOP 对象，以及对一些 ORB 内核中原有的对象进行修改，这些修改对象包括：IOR、GIOP_S、Interceptor。

有了 2 点前置条件后，就可以通过具有 RIO-IOP 协议的 IOR 字符串访问服务对象。整个访问过程可以从客户端和服务端 2 个角度来看，通过 8 个阶段建立到服务端的联系，如图 4 所示。

8 个阶段分别为：客户端 ORB 初始化阶段；客户端创建对象引用阶段；客户准备连接阶段；客户端请求分发阶段；服务端 ORB 初始化阶段；服务端对象适配器 OA 初始化阶段；服务端任务调度阶段；服务端请求分发阶段。

3.8　基于 RIO-IOP 的任务调度

在 RIO-IOP 协议的网络中，由于连接都是预先分配，所以 RzTask 任务不可能会监控到新连接产生的事件，每个 RzTask 任务最多只需要监控一个连接句柄。

1) 专属线程策略模式

一个 EndPoint 固定对应着一个连接对象，没有需要监听的句柄，所以 EndPoint 中的句柄集合最多包含着一个连接句柄。在专职 workerTask 任务空闲的时候，监控连接的职责也将由专职 workerTask 任务完成，RzTask 任务将会无连接句柄可以监控。当专职任务进行请求处理时，就会把连接句柄放入到连接集合中，由 RzTask 任务对连接句柄进行监控。

2) 线程池策略模式

在此模式下连接集合不会发生变化，一个 RzTask 任务将只监控一个连接句柄上的数据到达情况。

4　RIO-IOP 对 CORBA 基本服务的支持

4.1　RIO-IOP 双命名服务

RIO-IOP 命名服务是指支持经由 RIO-IOP 协议的 CORBA 命名服务，并且为了保证在分布式嵌入式系统中命名服务的高可靠性，RIO-IOP 命名服务提供双命名服务。

RIO-IOP 双命名服务是指客户端程序通过网络与主备命名服务进行连接。在默认情况下，客户端程序仅与主命名服务之间进行交互通信，当主命名服务出现故障时，客户端可以自动向备命名服务请求，以保障系统的持续稳定运行。同时在主备命名服务之间需要进行名字配置数据的同步，以使在主命名服务故障的情况下，备命名服务可以对该服务进行无缝切换并接管。

4.2　RIO-IOP 事件服务

标准 CORBA 通信模式中，客户程序与伺服程序之间的通信是直接的，客户程序向伺服程序发出调用请求，伺服程序进行处理并返回结果，这种处理方式下，2 者之间完全是耦合在一起的一对一的关系[10]。

事件服务则是一种异步通信松耦合的模式，由事件的提供者产生事件而事件的消费者接收事件，它们之间不是一对一的紧耦合的关系也不需要关心对方的当前的情况，而是各自连接在事件通道上[11]。事件通道作为 2 者之间的纽带，负责注册提供者和消费者，传递事件，并进行错误处理。

RIO-IOP 事件服务是指：开发的 CORBA 应用程序能够使用 RIO-IOP 协议，作为事件的消费者，能够向事件服务注册自己以及感兴趣的事件，在事件产生后，将得到通知；作为事件的提供者，能够使用事件服务提供的推送接口，将产生的事件推送给注册的消费者应用程序。

RIO-IOP 事件服务的前置条件是可以通过 RapidIO 总线传输符合 CORBA 的 GIOP1.2 规范的协议数据分组，后置条件是作为消费者与提供者的应用程序能够通过事件服务通道建立关联。

4.3　RIO-IOP 动态属性剥离分析

CORBA 除了静态调用外，还支持 2 种用于动态调用的界面：动态调用界面(DII, dynamic invocation interface)和动态框架界面(DSI, dynamic skeleton interface)，分别用于服务端和客户端的动态调用[12]。限于 RapidIO 的应用场景的要求和制约，在基于 RapidIO 的 RIO-RIO 协议的 CORBA 应用中对动态特性没有需求，所以对动态属性进行剥离，以更好的适应嵌入式环境并节省程序的

空间消耗。

RIO-IOP 动态属性剥离的目标如下。

1) 去除 ORB 中动态调用 DII 的相关部分，不影响其他部分的正常工作。

2) 去除 ORB 中动态框架 DSI 相关的部分，不影响其他部分的正常工作。

3) 去除 ORB 的 POA 中动态激活相关部分，不影响其他部分的正常工作。

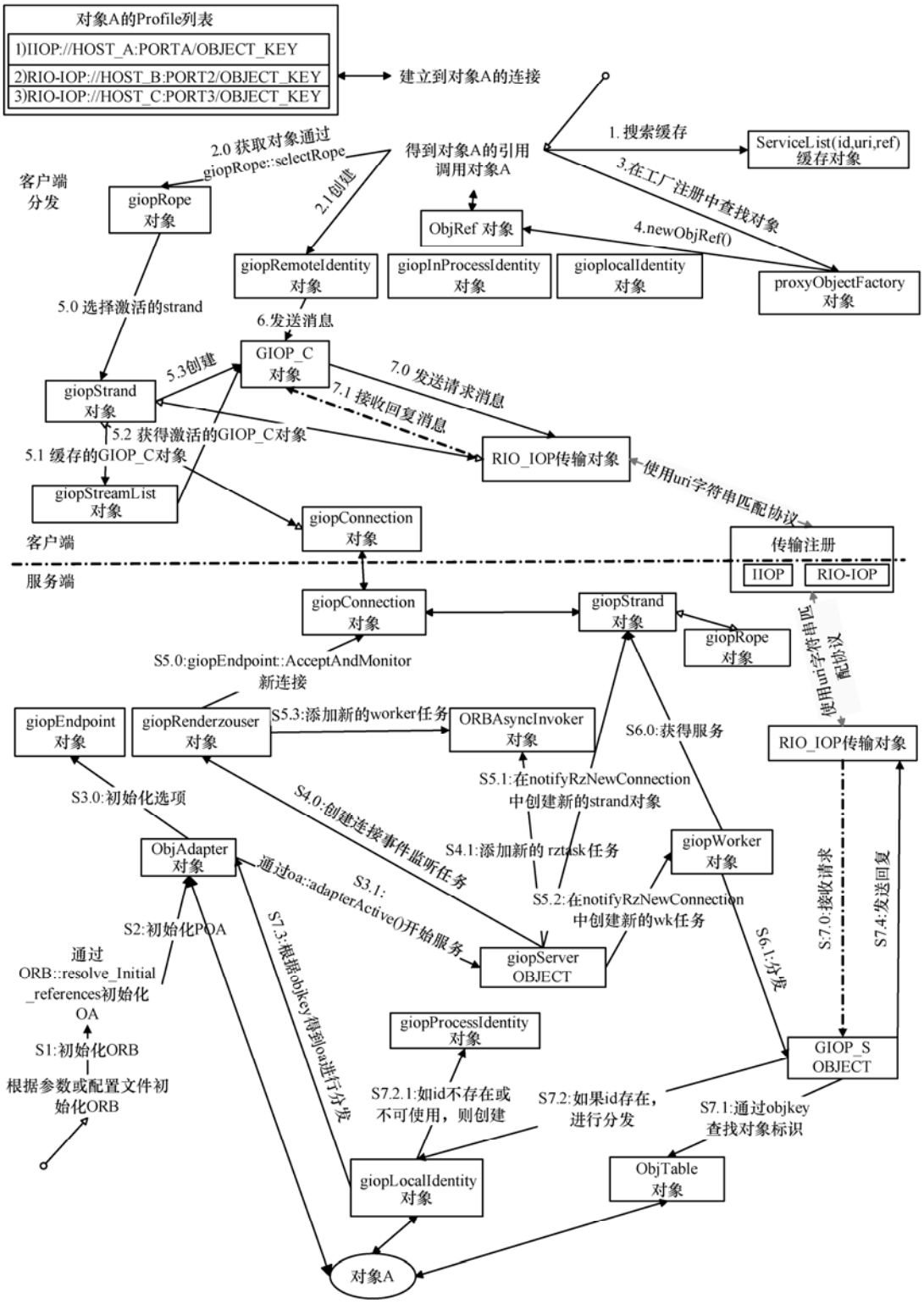


图 4　RIO-IOP 建立到服务对象的连接过程

5 协议 RIO-IOP 与 IIOP 的比较

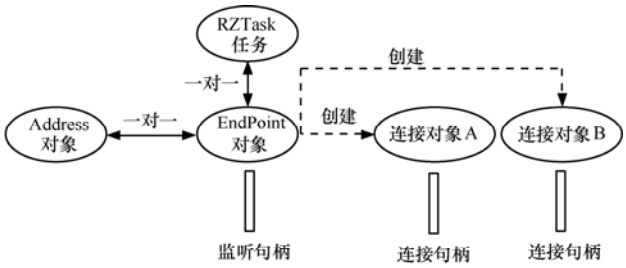
由于 RapidIO 的特性，构建在 RapidIO 上的协议栈 RCS 与 TCP/IP 存在差异，如表 3 所示。

表 3 RCS 连接与 TCP/IP 连接对比分析

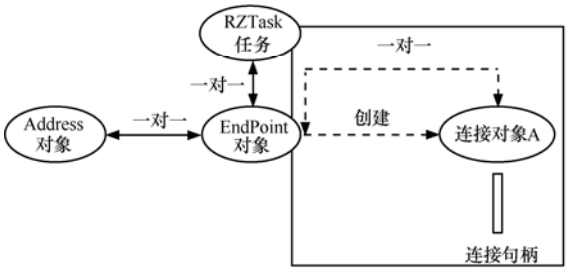
连接比较	TCP/IP	RCS
关系	不对等，服务端监听，客户端主动连接，服务端被动接受	对等
时间	临时建立	预先分配
方式	客户端连接到服务端的监听地址	网管系统配置
数量	对于服务端提供的每个监听地址，客户端都可以进行多个连接	固定

从表 3 中 TCP/IP 与 RCS 的连接差别中可以看出对于 TCP/IP 协议的服务地址，其地址是监听地址。而 RCS 协议中服务端发布的 IOR 中的服务地址必然也是连接在服务端侧的地址。

由于 TCP 协议与 RCS 协议在连接方面的差异，导致了服务端的服务端点对象（EndPoint）与连接对象之间关系在不同协议下的差别如图 5 所示。



(a)服务端点(EndPoint)与连接对象(connection)在IIOP中的关系



(b)服务端点(EndPoint)与连接对象(connection)在RIO-IOP中的关系

图 5 服务端点与连接对象在不同协议下的关系

服务端发布的服务端点对象（EndPoint）对应着客户端 Rope 对象中的一个服务地址对象（Address），EndPoint 对象与 Address 对象一一对应。每个服务端点都有一个 RzTask 任务为 EndPoint 对象服务。

在 IIOP 协议中，EndPoint 创建一个监听句柄，RzTask 任务负责监控。当有客户端连接上来的时候，RzTask 负责通知，并产生连接对象。

对于 RIO-IOP 协议中，网管系统需要预先对每个节点的连接进行配置，配置之后在每个节点上的 RCS 协议栈中都会保存着属于它的连接。每个 EndPoint 对应的是连接在服务端侧的地址，因此它只能对应着一个连接对象。

在客户端，每个 Rope 对象都包含了服务端发布的一系列服务端点地址，在 IIOP 协议中这些服务端点地址都是服务监听地址，客户端可以连接到此地址请求服务。而在 RIO-IOP 协议中，Rope 对象中的地址都是预先配置的那些连接在服务端侧的地址，Rope 对象中创建的 connection 对象必须循环那一系列地址与 RCS 的连接集合中的目标端地址进行匹配，匹配上的地址是服务端提供给此客户端的连接地址，对应的连接才是客户端与服务端的连接。

6 RIO-IOP 协议的实现与实验结果

6.1 RIO-IOP 协议的实现

基于现有 CORBA 产品的实现，在传输层增加一种新的通信方式，保证 CORBA 中的对象引用可以通过基于 RIO-IOP 协议进行请求/应答（包括异常）消息的传递，同时保证 CORBA 提供基本服务。

本文使用现有的 CORBA 产品 omniORB，新增一种通信方式 RIO-IOP，最终形成：基于 RIO-IOP 协议的 CORBA 中间件软件包；基于 RIO-IOP 的双命名服务包；基于 RIO-IOP 的事件服务包。它们都可以作为库文件（libOmniRIO_IOP.a，libOmniRIO_IOPNames.a ，libOminRIO_IOPEvent.a）被静态链接到 CORBA 应用中。

图 6 是包括 libOmniRIO_IOP.a，libOmniRIO_IOPNames.a，libOminRIO_IOPEvent.a3 个库文件的 CORBA 应用的镜像下载到某个嵌入式节点（此节点是在 RapidIO 网络中）上时，启动主命名服务后所有进程的运行情况，其中以 RCS 为前缀的进程是 RCS 协议栈相关的进程，omniORB 是基于 RIO-IOP 的 CORBA 中间件的相关进程，omni_ns 是 RIO_IOP 双命名服务的相关进程。

图 7 是某个嵌入式节点（此节点是在 RapidIO 网络中）加载了基于 RIO-IOP 的 CORBA 应用程序镜像后，运行 CORBA 客户端程序 omni_cli 的调试界面。图上显示了 omniORB 启动后各种调试信息的输出，这些信息包括该 ORB 使用的 GIOP 版本、字符集以及命名服务的地址信息等。

```
  NAME        ENTRY       TID    PRI   STATUS      PC       SP     ERRNO  DELAY
---------- ------------ -------- --- ---------- -------- -------- ------- -----
tExcTask   excTask      3f870500   0 PEND       20c294 3f870410       0     0
tLogTask   logTask      3f86d960   0 PEND       20c294 3f86d880       0     0
tShell     shell        3f65c540   1 READY      203194 3f65c170       0     0
tRlogind   rlogind      3f663910   2 PEND       1fe3cc 3f663530       0     0
tWdbTask   wdbTask      3f65e8a0   3 PEND       1fe3cc 3f65e640  3d0002     0
tNetTask   netTask      3f82f430  50 PEND       1fe3cc 3f82f360       0     0
tTelnetd   telnetd      3f661560  55 PEND       1fe3cc 3f661410       0     0
RCS_TRANS  792fc        3a0490f0  65 PEND       1fe3cc 3a049010       0     0
RCS_DEF_RCU8c8c0        3a026b40  90 PEND       1fe3cc 3a0269d0  3d0002     0
RCS_NAT    natMain      3a056260  95 PEND       1fe3cc 3a056180       0     0
omniORB    omni_corest  3ffff9b0 100 PEND       1feb74 3ffff610  3d0001     0
omni_ns    ns_st        3a012b60 128 PEND       1feb74 3a012150       0     0
omniORB    omni_thrst   39ffb190 128 PEND       1fe3cc 39ffa5c8  3d0002     0
```

图 6　RIO-IOP 应用进程运行状态截图

```
-> sp omni_cli
Task spawned: id = 0x3fffe590, name = t1
value = 1073735056 = 0x3fffe590
-> omniORB: Version: 4.1.3
omniORB: Distribution date: Tue Oct 17 10:43:28 BST 2011 dgrisby
omniORB: My addresses are:
omniORB: 6
omniORB: Maximum supported GIOP version is 1.2
omniORB: Native char code sets: ISO-8859-1 UTF-8.
omniORB:   InitRef = NameService=corbaname:rio:0:700
omniORB:   abortOnInternalError = 0
omniORB:   abortOnNativeException = 0
omniORB:   acceptBiDirectionalGIOP = 0
omniORB:   acceptMisalignedTcIndirections = 0
omniORB:   bootstrapAgentHostname =
omniORB:   bootstrapAgentPort = 900
omniORB:   clientCallTimeOutPeriod = 0
omniORB:   clientConnectTimeOutPeriod = 0
omniORB:   clientTransportRule = * bidir,rio,tcp
```

图 7　RIO-IOP 应用调试界面截图

6.2　RIO-IOP 协议的性能测试

实验条件：包括 3 个 PPC 数据处理模块和一个交换模块, PowerPC 处理器模块包括 5 片 PowerPC 处理器(MSU-ppc 和 ppc_1 至 ppc_4)和 1 片 FPGA 处理单元，采用 RapidIO 接口接入板卡上的 RapidIO 交换芯片，RapidIO 交换芯片的 2 个物理链路对称地接入 RapidIO 交换网络中。RapidIO 交换模块包括 1 片 PowerPC 处理器和一个 RapidIO 交换网络。

对基于 RIO-IOP 协议的客户端与服务端的通信时延进行测试，采用 2 个节点之间多次回弹数据分组的方式进行测试。其中客户端与服务端之间发送不同大小数据分组，RapidIO：3.125GHz，TCP：100M，PowerPC 与 PowerPC 互通。

图 8 是使用基于 RIO-IOP 协议的 omniORB 中间件软件包与基于 IIOP 协议的 omniORB 中间件软件包。

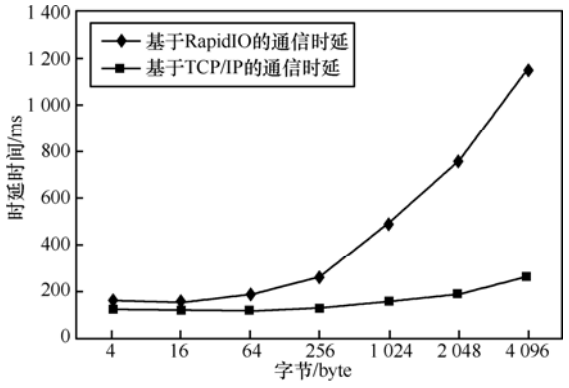


图 8　omni over RapidIO 与 omni over TCP 通信时延对比

从上述通信时延曲线对比分析可以得出在分布式嵌入式系统中，特别是芯片间及板间互连，对于不同 CORBA 产品，ORB 时延与传输速率在 RIO-IOP 协议下，都要高于基于 TCP/IP 的 IIOP 协议。

7　结束语

本文在对 CORBA 规范研究的基础上，将基于高速总线 RapidIO 的协议栈 RCS 无缝衔接到 CORBA 的传输层，把抽象协议 GIOP 映射为具体的 RapidIO，从理论和实践上提出并实现了全新的 RIO-IOP 协议，完成经由 RIO-IOP 协议的 CORBA 基本调用。经过对 RIO-IOP 协议性能的论证和测试，证明 RIO-IOP 协议在嵌入式领域和军事环境的硬件设备中，具有明显的优势和应用前景。但是本文实现上只使用了有限的硬件平台，对硬件平台的多样性未作出深入研究分析，下一步将在不同的硬件平台进行实验论证。

参考文献：

[1] RapidIO trade association. RapidIO specification 1.3[EB/OL]. www.rapidio.org/specs/current. 2001.

[2] ADMS J, KATSINIS C, ROSEN W, *et al*. Simulation experiments of a high-performance RapidIO-based processing architecture[A]. Network Computing and applications,2001[C]. Cambridge, MA,USA, 336-339.

[3] WU C R, CEN F, CAI H Z. A high-performance heterogeneous embedded signal peocessing system based on serial RapidIO interconnection[A]. IEEE Computer Science and Information Technology[C]. Cherydu, China, 2010. 611-614.

[4] OMG. Common Object Request Broker Architecture(CORBA) Specification, Part 2:CORBA Interoperability[S]. 2010.

[5] 李芳, 张虹. GIOP 协议和 CORBA 的性能优化[J]. 微计算机信息, 2006, 22(21):7-10.
LI F, ZHANG H. GIOP protocol and the performance optimization of CORBA[J]. Microcomputer Information, 2006, 22(21):7-10.

[6] GAILIARD G, BALP H, SARLOTTE M, *et al*. Mapping semantics of CORBA IDL and GIOP to open core protocol for portability and interoperability of SDR waveform components[A]. Design,Automation and Test in Europe[C]. Munich, Germany, 2008. 330-335.

[7] GOKHALE A S, SCHMIDT D C. Optimizing a CORBA internet inter-ORB protocol (IIOP) engine for minimal footprint embedded multimedia systems[J]. IEEE Communication, 2007, 17. 1673-1706.

[8] ZHANG S X, DEWEY J C F. An IIOP architecture for web-enabled physiological models[J]. IEEE Engineering in Medicine and Biology Society, 2001, 4:4048-4051.

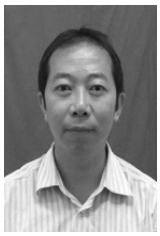
[9] GOKHALE A. Principles for optimizing CORBA internet Inter-ORB protocol performance system sciences[A]. Proceedings of the Thirty-First Hawaii International Conference[C]. Hawai, USA, 1998. 376-385.

[10] SENIVONGSE T, SURIYENTRAKORN P. A CORBA-based architecture for service change notification[A]. IEEE Enterprise Distributed Object Computing Conference[C]. Seattle, WA, USA, 2001. 22-33.

[11] MA C, BACON J. CORBA:a CORBA-based event architecture[A]. Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems[C]. Santa Fe, New Mexico, USA, 1998. 21-24

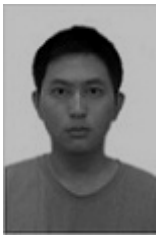
[12] 李祎.J2EE 平台下消息中间件及其安全性的研究[D]. 武汉: 武汉理工大学，2007
LI Y. The Study on Message Middleware and Its Security in J2EE platform[D]. Wuhan: Wuhan University of Technology,2007.

作者简介：



陈文宇（1968-），男，浙江兰溪人，博士，电子科技大学教授，主要研究方向为软件理论、模式识别、编译技术。

曾茹（1986-），女，四川宜宾人，电子科技大学硕士生，主要研究方向为软件理论、计算理论、编译技术。



皮维（1987-），男，重庆人，电子科技大学硕士生，主要研究方向为软件理论、模式识别、编译技术。

李文（1987-），男，四川资阳人，电子科技大学硕士生，主要研究方向为软件理论、模式识别、编译技术。

（上接第 69 页）

[8] ATENIESE G, HOHENBERGER S. Proxy re-signatures:new definitions algorithms, and applications[A]. ACM CCS 2005[C]. New York, NY, USA, 2005.310-319.

[9] SHAO J, CAO Z F, WANG L C, *et al.* Proxy re-signature schemes without random Oracles[A]. INDOCRYPT 2007[C]. Chennai, India, 2007.197-209.

作者简介：

冯涛（1970-），男，甘肃临洮人，博士，兰州理工大学计算机与通信学院副院长、研究员，主要研究方向为可证明安全协议理论、无线和移动网络安全。

梁一鑫（1980-），男，江苏扬州人，硕士，兰州理工大学计算机与通信学院讲师，主要研究方向网络安全、代理重签名。