基于 Protobuf 的数据传输协议①

聂晓旭[1], 于凤芹[1], 钦道理[2]

[1](江南大学 物联网工程学院, 无锡 214122)
[2](锐泰节能系统科学有限公司, 无锡 214135)

摘 要: 针对 XML、JSON 等数据结构在数据通信方面的不足, 利用 Protobuf 轻便高效的数据结构和二进制数据流传输方式的优点, 设计了一种网关与数据平台通信的数据传输协议. 该协议将数据以 Protobuf 格式封装到传输报文中, 通过定义应用层协议和设计数据通信确认机制建立数据传输通道, 并引入数据的序列化方法实现报文的传输. 测试结果表明, 该数据传输协议实现了网关和应用数据平台之间数据的高效和可靠传输.

关键词: 数据结构; Protobuf; 应用层协议; 通信确认机制; 序列化

Data Transmission Protocol Based on Protobuf

NIE Xiao-Xu[1], YU Feng-Qin[1], QIN Dao-Li[2]

[1](School of Internet of Things Engineering, Jiangnan University, Wuxi 214122, China)
[2](Reatgreen Energy-Saving System Science Co. Ltd, Wuxi 214135, China)

Abstract: Aiming at the shortage of XML and JSON data structure in the aspect of data communication, a data transmission protocol is designed to ues the data communication between gateway and data platform. It takes full advantage of the Protobuf that is portable and uses the binary data stream to transmit data. The protocol encapsulates data transmitted in Protobuf format in a message and builds a data transmission channel using the application layer protocol defined and the confirmation mechanism designed. It also introduces the serialization method of data to achieve the transmission of message. The test results show that the data transmission protocol achieve data transmission efficiently and reliably between the gateway and data platform.

Key words: data structure; Protobuf; application layer protocol; communication confirmation mechanism; serialization

随着物联网技术的快速发展, 异构网络中设备的种类和数量越来越多, 通信日益复杂. 尤其是感知层网关与应用层管理平台之间的数据传输量日益庞大, 对系统的性能要求也越来越高. 数据转发是网关运行的核心任务之一, 传统的数据表示语言, 如 XML、JSON 等, 可读性高, 广泛应用于 Web 开发的数据交换中[1-4], XML 是结构化的文档, JSON 是轻量级的文本数据交换格式,在数据传输过程中, 需要传输与数据本身无关的对象名和边界符等信息[5-7], 增加网络负担和硬件的开销. Google Protobuf buffers(简称 Protobuf)是一种轻便高效的结构化数据存储格式, 它用于结构化数据的串行化. 通过序列化速度和内存占用等分析, Protobuf 在数据交换中更加有优势[8,9]. Protobuf 在通信过程中, 传输的数据只包含数据本身和数据标签, 不传输属性名和边界符, 传输同样的数据, 占用更小的内存空间. 为了实现物联网感知层网关与应用层数据平台的通信, 通过对数据结构和协议通信确认机制的设计, 实现了数据的可靠传输.

1 协议架构的设计

该协议用于感知层网关与应用层平台间的通信, 根据对数据帧元素、数据安全性传输和数据流向的分析, 协议分为数据链路层、数据分包层、数据加密层和 Protobuf 应用层, 如图 1 所示. 其中数据链路层主要

① 基金项目:2012 年度教育部—中国移动科研基金(MCM20122013)
收稿时间:2014-11-23;收到修改稿时间:2015-01-05

用于通讯链路参数的配置及连接状态的维护, 数据分包层用于数据包的封装和拆分解析, 数据加密层实现了对数据包的加密和解密, 其中数据加密层可以根据配置进行关闭与打开, Protobuf 应用层则是对数据传输中所使用的 Protobuf 数据结构的定义. 网关与应用平台之间定义的通信协议为信息上报协议, 即网关向平台推送感知层终端的数据, 该协议承载在 TCP 传输协议之上.

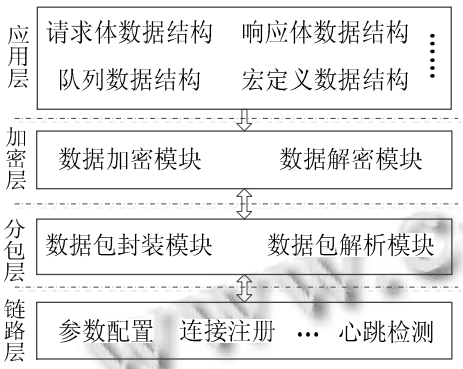

图 1 协议框架

2 数据传输协议的设计

2.1 应用层协议语法的定义

在数据通信中应用层协议要实现的功能是如何在 TCP 协议基础上通过自定义命令名实现数据传输、数据传输中的应答确认以及通信中异常事件的处理. 根据数据通信中流控制的需要, 定义了数据通信协议中使用的命令名. 如表 1 列出了部分正常通信所使用的命令名、实现的功能和对应的宏定义.

应用层协议所要实现的功能包括登陆平台、登陆成功确认, 检测连接状态的心跳、心跳确认, 发起数据透明传输和确认数据透明传输等, 以及异常处理等功能. 在表 1 中, 网关命令符表示由网关发送至平台的请求或应答命令名, 功能表示该命令符在平台内所要执行的操作, 如登陆平台请求、发起数据透明传输请求等. 宏定义是该命令符所对应的具体 16 进制的值. 平台命令符是由平台发送至网关的请求或应答命令名, 功能是该命令名在网关内所要执行的操作, 宏定义是该命令符所对应的 16 进制的值.

表 1 应用层协议定义

网关命令符	功能	宏定义	平台命令符	功能	宏定义
CMD_DU_LOGIN	登陆平台	0x0502	CMD_SVR_LOGIN_ACK	登录确认	0x1502
CMD_DU_LOGOUT	注销登录	0x0503	CMD_SVR_LOGOUT_ACK	注销确认	0x1503
CMD_DU_HEARTBEAT	发送心跳检测	0x0504	CMD_SVR_HEARTBEAT_ACK	心跳确认	0x1504
CMD_DU_UPDATE_NO_ACK	编号写入确认	0x1505	CMD_SVR_UPDATE_NO	设置网关编号	0x0505
CMD_DU_RESPONSE_PARAMS	网关返回参数	0x0508	CMD_SVR_REQUEST_PARAMS	获取网关参数	0x0507
CMD_DU_DIRECT	发起透明传输	0x0601	CMD_SVR_DIRECT_ACK	透明传输确认	0x1601
CMD_DU_DIRECT_ACK	透明传输确认	0x1602	CMD_SVR_DIRECT	发起透明传输	0x0602

2.2 通信报文格式的设计

通信过程中, 平台和网关都按照约定的固定格式对报文进行封装打包, 当接收到对方发送的数据报文时, 解析器按照打包的逆过程解析数据包. 本协议所设计的通信报文格式如图 2 所示, 主要由命令头和数据区组成.

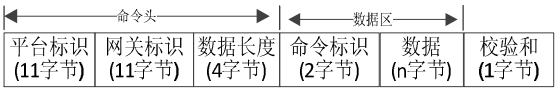

图 2 报文格式

命令头由平台标识、网关标识和数据长度构成, 平台标识是网关发送注册请求的目标平台, 网关标识是请求接入平台的网关, 数据长度是数据包中封装的数据的字节数; 数据区包括命令标识和数据域, 命令标识是网关或者平台作为报文接收者将要执行的操作, 数据是将要传输的数据信息; 校验和是命令头和数据区的校验和. 校验和是用于检验报文在传输过程中是否产生错误, 如果产生错误, 则将该数据包简单丢弃.

2.3 Protobuf 数据结构的定义

Protobuf 数据结构不同于 XML 和 JSON, JSON 数据结构在定义和传输时, 是以名称/值对的形式定义, 传输的数据包括名称、值和边界符. 如{“router”: “123”}, 传输的数据除了值 123, 还包括名称 router 和 “ ” : “ ”5 个边界符. protobuf 应用层在.proto 文件中定义了所有数据的 message 格式的数据结构, 是数据包

的原型文件. 在如下.proto 文件中定义了协议的请求体和响应体的数据结构, 在数据结构中定义通信数据参数和采集数据等信息.

Protobuf 数据结构拥有一个或多个特定的字段, 每个字段拥有一个修饰符、值类型、值名和字段标签. 修饰符用于指定该字段是必选字段(required)、可选字段(optional)或者是重复字段(repeated), 值类型指定了消息内容的数据类型, 如整形 int32、string 等, 值名是用于标识消息内容所属的对象, 字段标签用于标识该字段序列化时在二进制流中的位置, 反序列化后该字段的标签值必须与序列化前相同. Protobuf 数据结构在数据传输过程中是以键/值对的形式定义的, 如.proto 中 router 的数据的传输包括“值”123 和“键”所对应的标签 1, 数据表示形式非常精简, 由此可以看出 Protobuf 传输数据时所占用的内存较小, 效率相对较高.

.proto 文件的请求和响应数据结构文档:

```
message RouterRequest {// 请求数据结构
  required  int64   seq = 1; // 包序列号
  required  string  router = 2; // 网关编号
  required  Command  cmd = 3; // 命令体
  optional  bytes   serialized_data = 4; // 序列化数据
}
message RouterResponse {// 应答数据结构
  required  int64   seq = 1; // 确认包序列号
  required  string  router = 2; // 网关编号
  required  Result  result = 3; // 应答确认标识
  required  Command  cmd = 4; // 应答命令体
  optional  bytes   serialized_data = 5; // 序列化数据
}
```

网关和平台可以同时发送多个请求或应答, 如下文档中定义了请求队列和响应队列的数据结构, 当网关或平台有多个请求或响应时, 将请求或响应封装在请求队列或者响应队列中, 通过一个数据包将队列发送到接收端.

.proto 文件的队列数据结构文档:

```
message RouterMessage {// 队列数据结构
 repeated  RouterRequest  request = 1; // 请求队列
 repeated  RouterResponse  response = 2; // 响应队列
}
```

3 数据传输原理

3.1 数据传输通道的建立

数据传输通道的是由协议的链路层建立和维护的, TCP 协议是实现数据传输的传输层协议, 应用层协议用于实现数据的具体交互. 网关与平台的通信过程描述如图 3 所示. 完整的数据通信从建立 TCP 连接开始, 当网关向平台上报数据时, 网关首先向平台发送注册认证请求, 如果在超时时间内没有收到应答, 网关重新发送注册请求; 如果注册成功, 平台返回注册成功应答. 然后网关向平台定时发送心跳检测命令, 检测链路连接状态, 如果网关收到平台的心跳应答命令, 则链路连接正常, 如果在超时时间内未收到心跳应答, 心跳检测失败变量值加 1, 平台再次发送心跳检测, 如果连续 10 次未收到心跳应答, 则表示连接断开, 网关再次向平台发送链接请求; 如果心跳检测正常, 则网关向平台发送数据传输请求, 平台收到请求后进入数据收发状态, 同时向网关返回传输应答命令, 网关也进入数据收发状态. 此时, 平台与网关之间的数据传输通道建立成功. 当平台和网关之间数据传输结束时, 为了节省 CPU 资源, 平台调用网关提供的注销方法, 断开连接.

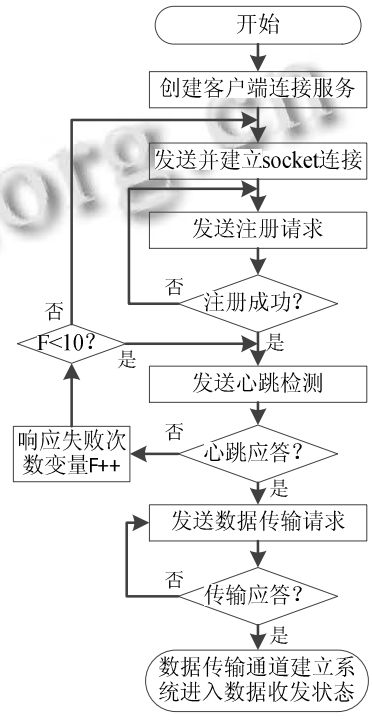

图 3　数据传输通道建立流程

3.2 数据传输的具体实现

平台作为资源整合节点，支持多个网关子节点的接入，接入数据量由系统硬件和带宽等资源决定. 在数据传输过程中，平台作为 TCP 协议的服务器，网关作为 TCP 协议的客户端，如下客户端程序中给出了客户端在数据传输中的实现.

```
client.on('data', function (data) {
  var rv = ParseRecvData(data); //响应数据解析
  if (rv > 0) {
   if (rv === Command.CMD_SVR_LOGIN_ACK) {
       server_state.login = 1; //登陆成功
       comm.HeatServer(client); //心跳检测
}else if
(rv===Command.CMD_SVR_HEARTBEAT_ACK)
{   if (server_state.connect != 1) {
       server_state.connect = 1;//连接成功
      comm.SendDataDirect(); //数据透传
                                  }
} else if (rv === Command.CMD_SVR_DIRECT_ACK)
{ console.log('recv .CMD_SVR_DIRECT_ACK:','0x'+rv.
toString(16)); //输出传输状态
   }
 }
})
```

当网关向平台发起通信时，若连接和通信正常，网关接收平台的响应数据包，并通过 ParseRecvData() 封装函数解析，同时判断响应体数据结构中 cmd 字段的命令符，客户端做出响应. 如果平台响应命令符是登陆确认字段，则网关启动 HeatServer()函数发送心跳检测命令，当网关收到平台的心跳确认命，即表示连接正常，平台进入数据收发状态，同时网关启动数据透传模块 SendDataDirect()，读取网关下感知层终端的数据，然后通过数据分包层将数据打包并发送至平台. 当平台接收到网关上报的数据时，通过数据解析模块解析并验证数据，同时向网关返回接收确认报文，网关接收并输出传输状态信息，实现单次数据上报工作.

3.3 传输数据的封装和拆分流程

网关向平台推送数据时，首先由 Builder(构建器)格式化.proto 文件，它解决了所有类型的引用，并返回执行所有必要的检查准备使用的类. 网关中上报数据的封装过程如图 4 所示. 网关首先初始化 seq、router、cmd 和 serialized_data 字段，即将要传输的数据与通信参数初始化到响应数据结构中，seq 表示该数据包在响应队列中的位置，router 是上报该数据的网关的标识，cmd 表示网关与平台通信的功能码的宏定义，serialized_data 是将要传输的数据的二进制值. 然后调用 Protobuf 提供的 encode()方法，将数据结构序列化为二进制数据流，存入缓存中，并将数据写入队列数据结构的请求队列中. 最后将队列数据结构发送给数据分包层，数据分包层调用数据包封装模块，向数据包添加平台标识、网关标识和计算校验和，并封装成通信报文. 网关通过数据发送函数将通信报文发送目标平台，同时开启响应计时，等待接收报文的平台返回接收确认. 如果网关在设置的超时时间内未接收到数据接收方的确认包，则重新发送报文.

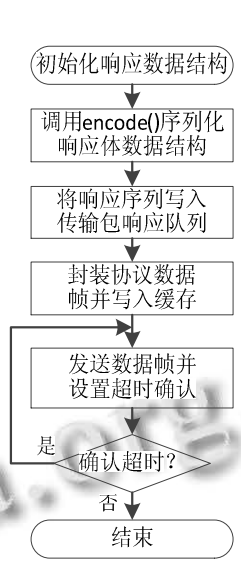

图 4 报文封装流程

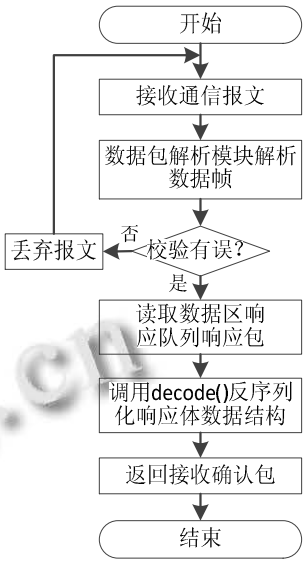

图 5 报文解析流程

当平台接收到网关推送的报文时，平台将报文发送到数据分包层，由分包层对报文进行解析，如图 5 所示. 首先提取目标平台标识，判断目标平台是否与自身标识匹配，然后提取数据域并计算校验码与报文中校验码对比，校验数据在传输中是否发生错误，如果数据有误则将数据包丢弃，不返回接收确认响应，平台继续等待接收网关推送的数据. 如果数据传输中未发生错误，则调用 Protobuf 中的 decode()方法，反序列化请求包，同时平台向网关返回数据接收成功确认包，数据传输完成.

4 实验测试

该协议的技术实现采用 javascript 语言，并以 Linux 下的 Node.js 作为开发运行平台，实验中将感知层电表的数据信息通过网关的客户端发送至平台的服务器端，并打印显示. 图 6 是客户端在整个数据传输中的数据流. 客户端首先想服务端发送 TCP 连接请求，连接成功后根据协议通信确认机制发送登陆注册请求、心跳检测和数据透传等命令. 客户端依次收到登陆应答 (CMD_SVR_LOGIN_ACK) 和心跳应答 (CMD_SVR_HEARTBEAT)，表示数据传输通道建立成功. 然后客户端想服务器端发送数据传输请求，并启动数据透传模块，该模块包括数据采集和数据封装模块，它将采集数据通过报文发送至平台的服务器端. 服务器端接收到数据并解析，同时向客户端返回数据传输成功应答(CMD_SVR_DIRECT_ACK)，单次数据传输结束.

```
[root@aman Tunner]# node lwclient.js
client connected:8001
recv CMD_SVR_LOGIN_ACK: 0x1502
recv CMD_SVR_HEARTBEAT_ACK: 0x1504
send data: {"timestamp":1415015567602,"nodeid":4136886848,"datas":[
{"name":"EP","vale":5242.91,"unit":"w"}]}
recv .CMD_SVR_DIRECT_ACK: 0x1601
recv CMD_SVR_HEARTBEAT_ACK: 0x1504
```

图 6 客户端数据流

服务器端在接收到客户端的数据时，通过数据解析模块对数据解析，同时根据协议命令符的宏定义做出响应操作. 如图 7 中，服务器端依次接收到客户端的登陆注册(CMD_DU_LOGIN)、心跳(CMD_DU_HEARTBEAT)、透传(CMD_DU_DIRECT)请求，当连接正常时开始接收数据并解析，同时返回数据接收确认. 通过本协议中对数据流通机制的控制，实现了数据的可靠传输. 通过登录注册、心跳检测和数据传输前的传输请求等通信控制方法，实现了数据身份的认证、连接通道的异常检测等.

```
[root@aman Tunner]# node lwserver.js
server start:8001
server connected
recv CMD_DU_LOGIN: 0x0502
recv CMD_DU_HEARTBEAT: 0x0504
recv CMD_DU_DIRECT: 0x0601
recive data: {"timestamp":1415015567602,"nodeid":4136886848,"datas"
:[{"name":"EP","vale":5242.91,"unit":"w"}]}
recv CMD_DU_HEARTBEAT: 0x0504
```

图 7 服务端数据流

5 结语

通过引入 Protobuf 数据结构和序列化方法，并在协议架构的基础上，对通信协议、传输报文格式和数据结构的详细设计和介绍，设计了完整的数据传输协议，实现了感知层网关和应用层数据平台间的数据传输. Protobuf 精简的数据结构提高了数据传输的效率，通过将数据结构封装到传输报文，并按照应用层数据通道的通信流程，实现了数据在网关和平台间的可靠传输. 同时采用封装队列发送多个请求或响应，节省网络资源. 该协议已成功的实现现场应用，且不仅仅局限于网关和平台的通信，通过对数据接收和数据发送方的应用环境的配置，以及数据传输通道的建立，可以实现两个资源整合节点间的数据通信.

参考文献

1 黄国言,郭徽.基于 XML 的协同设计中数据交换方法的研究.计算机工程与设计,2007,28(24):6000–6002,6047.

2 范炜,徐洪泽.基于 TCP/IP 协议的嵌入式多串口网关的设计.计算机工程与设计,2008,29(1):80–82.

3 赵冲冲,王鑫,胡长军.XML 格式领域数据传输的优化技术研究.计算机科学,2009,36(08):185–188.

4 胡文发,白中英.基于 J2ME/J2EE 的 JSON 数据交换的探讨.电子设计工程,2009,17(12):102–103.

5 高静,段会川.基于移动设备的 JSON 数据传输效率研究.信息技术与信息化,2011,(1):13–16.

6 高静,段会川.JSON 数据传输效率研究.计算机工程与设计,2011,32(7):2267–2270.

7 张涛,黄强,毛磊雅,高兴.一个基于 JSON 的对象序列化算法.计算机工程与应用,2007,43(15):98–100,133.

8 Gligoric N, Dejanovic I, Krco S. Performance evaluation of compact binary XML representation for constrained devices. 2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS). 2011. 1–5.

9 Wendt A, Faschang M, Leber T, Pollhammer K, Deutsch T. Software architecture for a smart grids test facility. Industrial Electronics Society, 39th Annual Conference of the IEEE (IECON 2013). 2013. 7062–7067.