# 基于ZEROMQ库的组播实现

1. 绪论
   1. ZEROMQ

ØMQ （也拼写作ZeroMQ，0MQ或ZMQ)是一个为可伸缩的分布式或并发应用程序设计的高性能异步消息库。它提供一个消息队列, 但是与面向消息的中间件不同，ZeroMQ的运行不需要专门的消息代理。该库设计成常见的套接字风格的API。

ZEROMQ支持发布/订阅的消息模式，即服务器为发布者，将消息发布到绑定的ip和端口，发布者只发送信息，不接受信息。用户为订阅者，连接到ip和端口，接收信息，不发送信息。

* 1. 组播

组播在发送者和每一接收者之间实现点对多点网络连接。如果一台发送者同时给多个接收者传输相同的数据，也只需复制一份相同的数据包。它提高了数据传送效率，减少了骨干网络出现拥塞的可能性。

* 1. PGM和EPGM

PGM是一种可靠多播协议，常见的PGM实现有开源的openpgm和微软的msPGM。ZEROMQ库中使用的是openpgm。但是pgm协议在windows系统中需要管理员权限，ZEROMQ基于UDP实现了PGM的可靠多播，称为EPGM，不需要管理员权限。

1. 配置环境
   1. OPENPGM

下载openpgm库<http://miru.hk/openpgm/>

安装openpgm

* 1. ZEROMQ

下载libzmq库<https://github.com/zeromq/libzmq>

使用cmake gui configure下载文件夹，选择with openpgm并选择openpgm安装文件夹

Generate

Open project

将openpgm的include文件夹添加到libzmq项目的附加目录

将openpgm的lib文件夹添加到libzmq项目的链接目录

将openpgm的lib文件夹内的.lib文件添加到libzmq项目的链接文件

Build libzmq项目

* 1. 可靠组播协议

在“网络和Internet”窗口中，选择左侧的“状态”，并在右侧中找到且点击“更改适配器选项”。打开网络连接界面，此时会有以太网和WLAN，如果是有线上网那么需要对以太网设置，无线的话就选择WLAN，右键点击属性。在弹出的“属性”窗口中，找到并勾选“Microsoft网络适配器多路传送协议”，点击下面的“安装”。在选择网络功能类型界面，选择协议，然后点击添加按钮。在弹出的窗口中，选择“可靠多播协议”并点击“确定”即可。

* 1. 在visual studio创建zmq项目

在build目录内的libzmq目录内可以找到.lib文件，放在zmq项目内的lib文件夹内，build目录内的bin文件夹内可以找到.dll文件，跟之后生成的exe文件放在一起。

将从github上下载的libzmq文件夹里的include文件夹复制到项目文件夹，并添加到项目的附加目录

将lib文件夹添加到链接目录，并把.lib文件添加到链接文件

1. ZEROMQ库常用函数
   1. void\* zmq\_ctx\_new()

创建一个新的上下文，主要用于生成zmq\_socket

* 1. void \*zmq\_socket (void \*context, int type);

创建一个新的socket, 其中的context即zmq\_ctx\_new生成的上下文，type我们只用了两种：ZMQ\_PUB和ZMQ\_SUB

* 1. int zmq\_setsockopt (void \*socket, int option\_name, const void \*option\_value, size\_t option\_len);

最重要的函数，用来设置socket的各种属性，比如缓存区，速率，高水位标记。必须在socket进行connect或者bind之前进行设置

Void\* socket即zmq\_socket生成的socket 剩下的参数请参考下表

|  |  |  |  |
| --- | --- | --- | --- |
| Option\_name | Option\_value type | Option\_value 默认值 | Option\_value单位 |
| ZMQ\_RATE | Int | 100 | kilobits per second |
| ZMQ\_RCVBUF | Int | The OS default value | bytes |
| ZMQ\_RCVHWM | Int | 1000 | Messages |
| ZMQ\_RECOVERY\_IVL | Int | 10000 | milliseconds |
| ZMQ\_SNDBUF | Int | The OS default value | bytes |
| ZMQ\_SNDHWM | Int | 1000 | messages |

Rate就是速率，但是这个速率也要取决于你发送信息的大小的频率，实际发送速率是min(信息大小\*频率，速率)。 但是当速率小于信息大小\*频率时，发送队列会开始堆积。所以建议设置一个较大的rate，通过控制频率来控制发送速率。

RCVBUF和SNDBUF就是接收区缓冲区和发送端缓冲区，经过测试，设置和不设置并不会影响消息延迟或者丢包。（测试为每个包512字节，每个包的发送延迟为100微秒）。但是建议设置一个相对大的值比如20mb。

RCVHWM和SNDHWM是zeromq中提出的新概念，分别是接收区高水位和发送区高水位，zeromq的发送是通过队列，也可以把这个队列看作是水管，队列中堆积的信息越多，水管水位越高。水位即是堆积信息的数量，如果水位高过高水位标记，pub端会将新发送的数据直接丢弃。正常情况下，队列中不会堆积数据，但是还是建议设置一个相对大的值来防止意外。

Recovery\_IVL是恢复时间，是多播组允许一个用户失联的最长时间，多播组会保存这段时间的数据在内存中，当用户恢复连接时重新发送。占用的内存为 发送速率\*时间，因为我们的发送速率相对较低，可以设置一个较大的恢复时间比如60000即一分钟。

SUB端一定要设置过滤器才能接收到数据，不然会将所有数据丢弃。例子如下

char filter[x]=”xxx”;

zmq\_setsockopt(subscriber, ZMQ\_SUBSCRIBE,filter,strlen(filter));若filter=“”则接收所有消息。

* 1. int zmq\_bind (void \*socket, const char \*endpoint);

这里的endpoint直接按照例子来，"epgm://239.192.1.23:5555"，可以选择epgm或者pgm，后面是组播的ip地址和端口。 Pub端使用bind绑定组播地址

* 1. int zmq\_connect (void \*socket, const char \*endpoint);

这里的endpoint跟bind一样，，"epgm://239.192.1.23:5555"，sub端用conncet

* 1. int zmq\_recv (void \*socket, void \*buf, size\_t len, int flags);

sub端的接收函数，这边的buf可以是字符串也可以是结构体，但一定要和发送端匹配，len就是buf的大小，flags设置为0即可

* 1. int zmq\_send (void \*socket, void \*buf, size\_t len, int flags);

pub端的发送函数，跟recv一样。

* 1. int zmq\_close (void \*socket);

使用完以后关闭socket

* 1. int zmq\_ctx\_destroy (void \*context);

使用完以后删除上下文

1. 其他函数
   1. uint64\_t xdk\_nanosecond\_timestamp()

用来获取纳秒级别的时间戳

* 1. inline void xdk\_microsecond\_delay(uint64\_t interval)

用来进行微秒级别的sleep

1. 具体实现