GHF automation GmbH

# Boost ASIO

## Broadcasts and Multiple Interfaces

# Key Components of ASIO

GHF automation GmbH

**io_service**

- interacts with operating system
- calls handlers
- non-copyable -> use shared_ptr

I/O objects, e.g. tcp::**socket**, udp::socket

- hold a reference to **io_service**
- provide synchonous and asynchronous methods
- pass request and callback functions to io_service

**timer**, e.g. deadline_timer

- hold a reference to **io_service**
- used to define timeouts on pending asynchronous operations

**strands**

- ensure sequential execution of operations

# Sequential Execution with 'strand'

GHF automation GmbH

- Requests to same 'strand' are executed sequentially
- use strand::**post()** for maintaining sequential order or strand::**wrap()**, if order is not relevant

```
for( auto l_socket : l_aSockets ){
  _fillRequestQueue(&l_socket->m_aReqQ);

  for( auto l_tlgr : l_socket->m_aReqQ){
    l_socket->m_strand.post(
      boost::bind(&CLocalSocket::SendReq
            , l_socket, l_tlgr));
  }
  l_socket->m_strand.post(
      boost::bind(&CLocalSocket::RecvRsp
            , l_socket));
}
```

# Steps in Scan Routine

GHF automation GmbH

1. get network interfaces
2. create a socket for each interface
3. post each telegram to send to each socket
4. post one receive operation for each socket
5. create deadline_timer, which will close sockets after timeout has expired
6. call io_service::run()
7. run() will continue, until receive operations terminate, because their socket has been closed

# Get Local Network Interfaces I

GHF automation GmbH

- via resolve() of local host name -> only IP address

```
ba::io_service io_service;

ba::ip::tcp::resolver resolver(io_service);
ba::ip::tcp::resolver::query query(ba::ip::host_name(), "");
ba::ip::tcp::resolver::iterator it = resolver.resolve(query);

while (it != ba::ip::tcp::resolver::iterator())
{
  ba::ip::address addr = (it++)->endpoint().address();
  if (addr.is_v4())
    // store
```

# Get Local Network Interfaces II

GHF automation GmbH

- via GetAdaptersInfo() -> IP address and subnet mask
  - Windows specific

```
IP_ADAPTER_INFO *pAdapterInfo;
...
l_dwRet = ::GetAdaptersInfo( pAdapterInfo, &ulOutBufLen);

if( l_dwRet == NO_ERROR ){
 const IP_ADAPTER_INFO *pAdapter = pAdapterInfo;
 while (pAdapter != nullptr) {
   if (pAdapter->Type == MIB_IF_TYPE_ETHERNET
       || pAdapter->Type == IF_TYPE_IEEE80211) {
     const IP_ADDR_STRING *pAddr = &(pAdapter->IpAddressList);
     while (pAddr != nullptr)
     {
      ba::ip::address_v4 l_addr =
        ba::ip::address_v4::from_string(pAddr->IpAddress.String);
```

# Create and Open Socket

GHF automation GmbH

```
if( m_socket.open(ba::ip::udp::v4(), l_error) == 0 ){
 try{
  std::string l_strAddrV4 = p_addrLocal.to_string();
  m_socket.set_option(
      ba::ip::udp::socket::reuse_address(true));
  m_socket.set_option(
      ba::socket_base::broadcast(true));

  // setting the outbound_interface is for
  // multicasts only, not for broadcasts
  // m_socket.set_option(ba::ip::multicast::
  //   outbound_interface(m_addrLocal));

  // 'p_addrLocal' specifies interface to use
  m_socket.bind(ba::ip::udp::endpoint(p_addrLocal, 0));
```

# async_send_to()

GHF automation GmbH

```
m_socket.async_send_to(
  ba::buffer(l_tlgr.data(),l_tlgr.size())
   ,ba::ip::udp::endpoint(ba::ip::address_v4:
   :broadcast(), l_usPortNo)
    ,boost::bind(&CLocalSocket::OnSendTo
      , this, ba::placeholders::error
      , ba::placeholders::bytes_transferred));

void OnSendTo ( const sys::error_code& error,
   size_t bytes_transferred )
{
```

# async_receive_from()

GHF automation GmbH

```
m_socket.async_receive_from(
   ba::null_buffers(), m_senderEndpoint
   , boost::bind(&CLocalSocket::OnRecvRsp
      , this, ba::placeholders::error
      , ba::placeholders::bytes_transferred));

 void OnRecvRsp(const sys::error_code& error
   , size_t bytes_transferred) {
  if (error == 0 || error == ba::error::message_size)
  {
    ba::ip::udp::endpoint l_senderEndpoint;
    std::vector<uint8_t> buf(m_socket.available());
    m_socket.receive_from( ba::buffer(buf)
      , l_senderEndpoint );
```

# Execute the Scan

GHF automation GmbH

```
l_deadline_timer.async_wait(boost::bind(&OnScanTimerEx
   pired, boost::ref(l_deadline_timer),
   boost::ref(l_aSockets)));
```

**`io_service->run();`**

- io_service::run() processes events and calls the send and receive handlers
- io_service schedules the tasks on the threads, which call io_service::run()
- handlers are executed in one of this threads -> user determines the possible thread contexts, but not the thread for a specific request
- run() will exit, when there is nothing to do (but we always have an async_receive pending, until socket is closed by timer)

# deadline_timer closes sockets

GHF automation GmbH

```
static void OnScanTimerExpired(ba::deadline_timer&
   p_timer, t_LocalSockets& p_aSockets ){

 if (p_timer.expires_at() <=
   ba::deadline_timer::traits_type::now()) {
  for (t_LocalSockets::value_type& l_entry:p_aSockets)
    l_entry.get()->PostClose();

  // clear timer by setting timeout to infinite
  p_timer.expires_at(boost::posix_time::pos_infin);
}

void CLocalSocket::PostClose (){
 this->m_strand.post(boost::bind(&CLocalSocket::Close
                                  , this));
}
```

# Caveats

GHF automation GmbH

1. avoid conflicting access to same socket

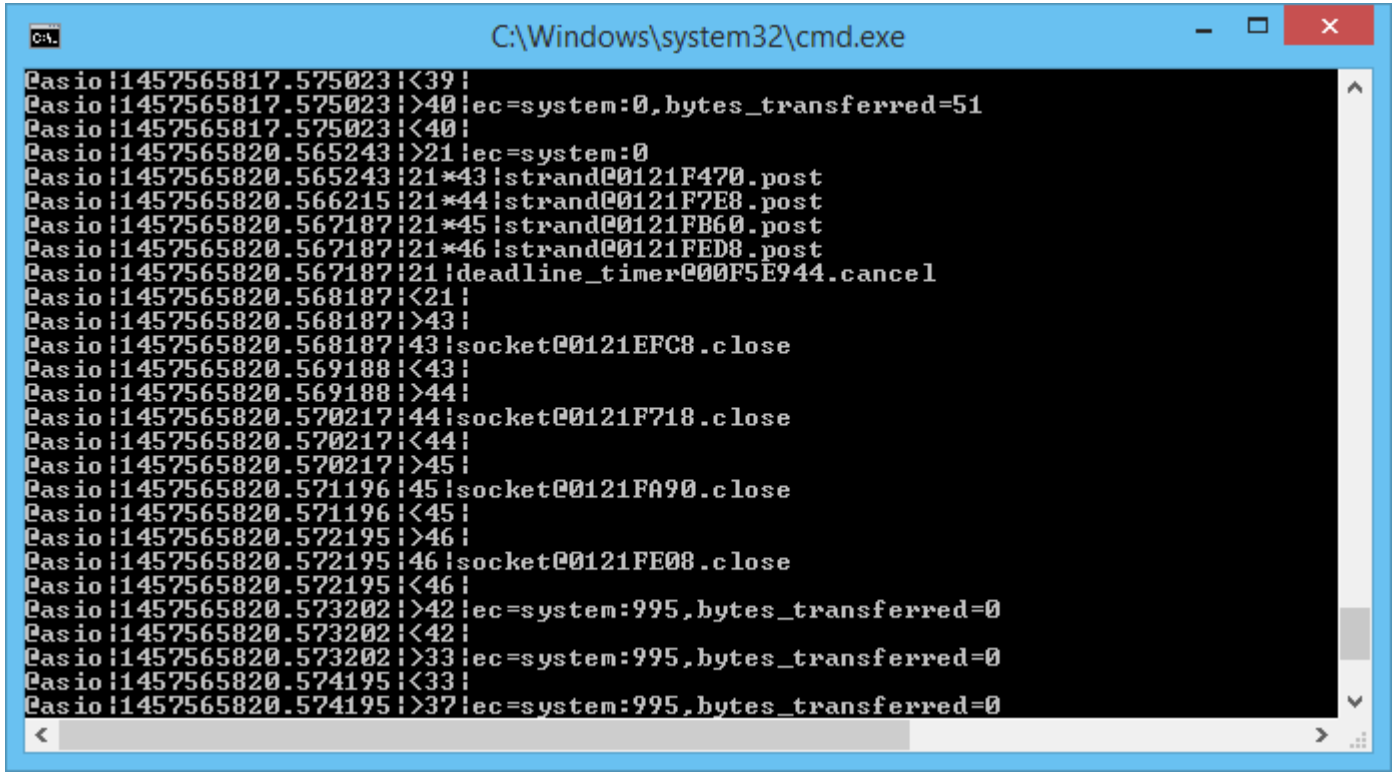
   -> use strands
2. ensure that the handler method's class instance still exists, when the handler is called

   -> pass 'shared_from_this' to handler object
3. if io_service::run() exists and you want to reuse the io_service, call io_service::reset() first
4. observe the difference between strand::post() and strand::wrap()
5. socket option 'outbound_interface' does not affect the interface used for broadcasts

# Debugging Support I

GHF automation GmbH

define BOOST_ASIO_ENABLE_HANDLER_TRACKING
enables tracking output to standard error

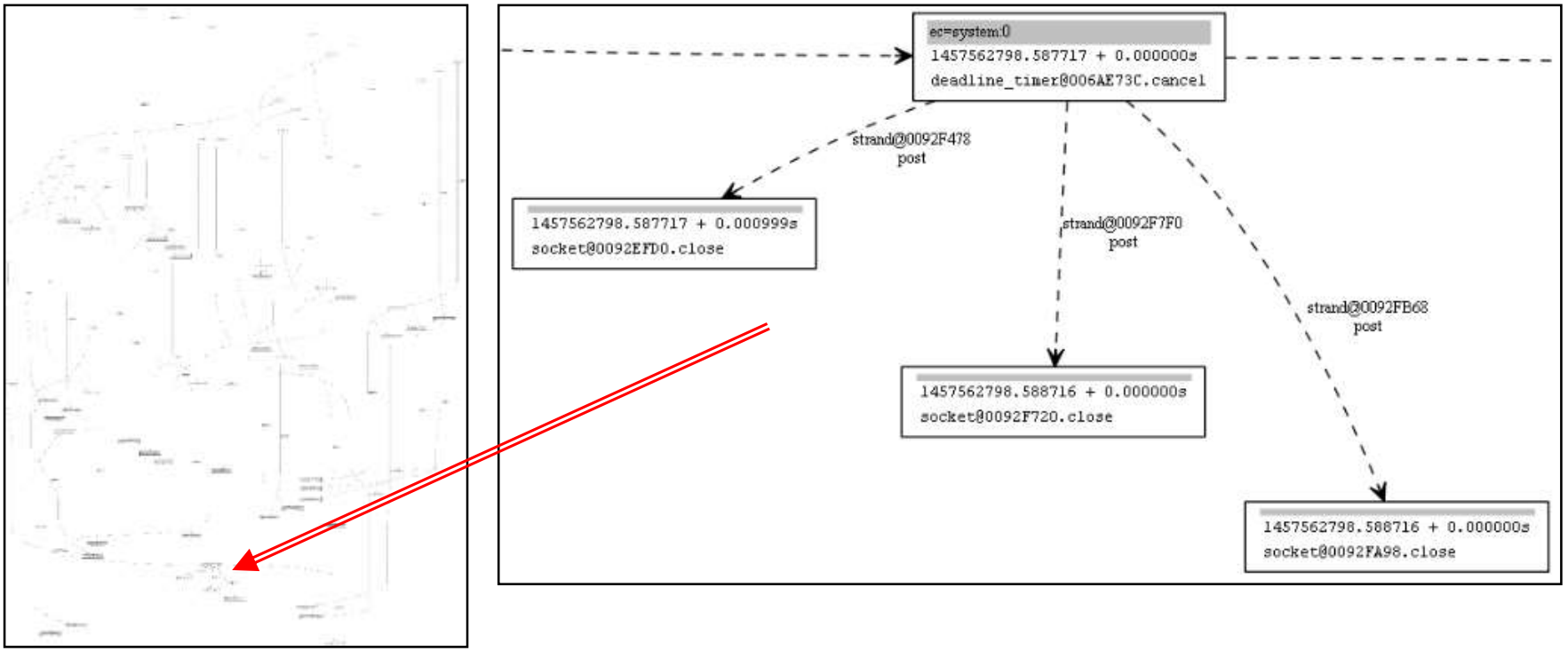


# Debugging Support II

GHF automation GmbH

Boost Asio includes Perl script to create drawing from tracking messages

```
perl handlerviz.pl <output.txt | dot -Tpng >output.png
```



# The End

GHF automation GmbH

Thank you for your attention!