Subject: Adding network proxy support to U++
Posted by Oblivion on Sun, 12 Oct 2014 13:45:33 GMT
View Forum Message <> Reply to Message

Hello guys,

One thing I miss in the U++'s network/socket framework is a generic network proxy support.
I am aware that some of the net classes (HttpClient/Pop3, for example) have built-in proxy support, but then, they support only the HTTP_CONNECT tunneling.
While this might be sufficient for the above mentioned classes, other socket operations can and might require listening/binding, in which case SOCKS proxy protocol(s) seem(s) to be the only (or a better) way to go, given the robustness of SOCKS protocol(s).
Also implementing the same code for every different class doesn't seem to me a good idea.

So here's the issue:
What started as a simple proxy code path for my FTP class now turned into a full-blown network proxy class supporting HTTP_CONNECT, SOCKS v4\4a\5 protocols.
Writing a synchronous generic network proxy class is actually very simple, given the simplicity of these widely used protocols.
On the other hand, writing an asynchronous proxy class is a bit harder, because of the increased code complexity, so I used HttpClient as a model.
Currenty I have a working (a)synchronous network class prototype called "NetProxy", which can connect via HTTP or SOCKS servers, using HTTP_CONNECT,SOCK4/4a/5 protocols, which allows both connecting and binding (only on Socks protocol) sockets through the respective proxy servers.

I would like to ask you two questions and hear your opinions on the topic:

1) In its present form, NetProxy class is a simple delegate/wrapper class, which takes a reference to an already constructed TcpSocket instance in its constructor, or through the NetProxy::Client() method, and simply stores a pointer to the socket. The thing is, I am not sure if this is the right or

way to go, or would it be better to have a TcpSocket derived class, say, something like ProxiedTcpSocket? (One advantage of delegate/wrapper class over TcpSocket derived class could be that the existing app code can be modified to take advantage of the proxy connections without too much hassle.)

2) Is an asynchronous (non-blocking) mode which gives some complexity to both api and internal code is good, or would a simple synchronous (blocking) mode suffice?

To give you the picture, here is stripped down version of the actual NetProxy prototype example (in async mode, but with only one socket.):

```
CONSOLE_APP_MAIN
{
 TcpSocket  s;
```

```
// NetProxy proxy(s, "127.0.0.1", 1080, "anonymous", "anonymous", NetProxy::SOCKS5);

 NetProxy proxy;
 proxy.Client(s);
 proxy.Server("127.0.0.1", 1080);

 proxy.Auth("anonymous", "anonymous");
 proxy.Remote("ftp.uni-erlangen.de", 21);
 SocketWaitEvent we;
 we.Add(s);
  we.Wait(100);
 for(;;) {

  proxy.Do();
  if(!proxy.InProgress()) {
   if(proxy.IsFailure()) {
    Cout() << "Proxy failed.\r\n";
    break;

   }
   else
   if(proxy.IsSuccess()) {
    Cout() << s.GetLine() << "\r\n";
    Cout() << "Success!";
    break;
   }
  }
 }
 s.Close();
}
```

What do you think?
If everything goes well, and after I have reliable code, I will upload the NetProxy to the Bazaar section, in case someone might find it useful.

(I'm not sure if this is the right place to post this, so please feel free to move this topic to the appropriate forum).

Regards,
Oblivion

Subject: Re: Adding network proxy support to U++
Posted by Mindtraveller on Mon, 13 Oct 2014 13:49:10 GMT

View Forum Message <> Reply to Message

Hi Oblivion,

In my opinion better support for proxies is very good idea. I also suppose both async & sync versions are useful.
We just need to correct it's interface according to U++ network classes style.
Of course, better proxying must be easy to attach to classes like HttpRequest if one needs it.

Subject: Re: Adding network proxy support to U++
Posted by mirek on Mon, 20 Oct 2014 15:33:38 GMT
View Forum Message <> Reply to Message

Sounds good!

Mirek

Subject: Re: Adding network proxy support to U++
Posted by Oblivion on Sun, 09 Nov 2014 15:56:42 GMT
View Forum Message <> Reply to Message

Hello Mindtraveller, Mirek.
Thank you for your suggestions. And I apologize I couldn't reply earlier, I was away for a while.

Here is what happened meanwhile:

After some toying with the network proxy idea for U++, I concluded that writing a monolithic, all-in-one network proxy class wouldn't be very effective (also imho, it would not be so upp-ish) way to implement it.

So I ditched the somewhat bloated, monolithic NetProxy class in favour of a modular approach, and abstracted a common interface and code for sync/async network proxies. This abstract class I called NetworkProxy. It contains all the necessary elements so far to succesfully implement most popular types of network proxy protocols. It has some advantages: Permits adding new/other sync/async network proxy protocols easily, relatively faster than NetProxy (due to less control code). Also we now have polymorphism: With this approach we can use a single Array object to contain a collection of different types of proxy instances if needed.

NetworkProxy abstract class currently has three derivatives with almost-identical interface: HttpProxy, Socks4Proxy, Socks5Proxy.

Here is the current "public" interface of NetworkProxy class and its derivatives (Note that SSL support and binding methods (in SOCKS4/5 proxies) are not yet implemented):

```
class NetworkProxy
{
```

```
public:
   enum ProxyType      { HTTP = 1, SOCKS4, SOCKS5 };

   NetworkProxy&       Attach(TcpSocket& sock)
   NetworkProxy&       Host(const String& host)
   NetworkProxy&       Port(int port)
   NetworkProxy&       Timeout(int ms)

   TcpSocket*          GetSocket() const
   int                 GetType() const

   bool                Connect(const String& host, int port);
   void                StartConnect(const String& host, int port);
   virtual bool        Do() = 0;

   bool                InProgress() const
   bool                IsSuccess() const
   bool                IsFailure() const

   String              GetErrorDesc() const

   NetworkProxy();
   virtual ~NetworkProxy();
};

class HttpProxy : public NetworkProxy
{

public:
   HttpProxy&          Auth(const String& user, const String& pass)
   bool                Do();

   HttpProxy();
   HttpProxy(TcpSocket& sock, const String& host, int port, const String& user = Null, const
String& pass = Null);
   ~HttpProxy();
};

class Socks4Proxy : public NetworkProxy
{

public:
   Socks4Proxy&        Auth(const String& user)
   bool                Do();

   Socks4Proxy();
   Socks4Proxy(TcpSocket& sock, const String& host, int port, const String& user = Null);
```

```
    ~Socks4Proxy();
};


class Socks5Proxy : public NetworkProxy
{

public:
    Socks5Proxy&        Auth(const String& user, const String& pass)
    bool                Do();

    Socks5Proxy();
    Socks5Proxy(TcpSocket& sock, const String& host, int port, const String& user = Null, const
String& pass = Null);
    ~Socks5Proxy();
};
```

As Mindtraveller suggested, I corrected the interface, but any further corrections, suggestions or ideas are always welcome.

To give you further idea, here is the actual and complete working example code of async HTTP_CONNECT proxy (It simply connects to five different Ftp servers via a public http proxy and gets the server greeting):

```
#include <Core/Core.h>
#include <NetworkProxy/NetworkProxy.h>

using namespace Upp;

CONSOLE_APP_MAIN
{

 static const char *target_hosts[] = {
  "ftp.uni-erlangen.de", // Aminet ftp server.
  "ftp.mozilla.org", // Mozilla ftp server.
  "ftp.freebsd.org", // FreeBSD ftp server.
  "ftp.microsoft.com", // Microsoft ftp server.
  "ftp.kde.org",  // KDE ftp server.
 };

 Array<TcpSocket> sockets;
 Array<HttpProxy> proxies;
 SocketWaitEvent socketevent;

 for(int i = 0; i < 5; i++) {
  TcpSocket& socket = sockets.Add();
```

```
  socketevent.Add(socket);
  HttpProxy& proxy = proxies.Add();
  proxy.Attach(socket);
  proxy.Host("97.77.104.22");
  proxy.Port(80);
  proxy.StartConnect(target_hosts[i], 21);
 }

 while(proxies.GetCount()) {
  socketevent.Wait(10);
  for(int i = 0; i < proxies.GetCount(); i++) {
   HttpProxy& proxy = proxies[i];
   proxy.Do();
   if(!proxy.InProgress()) {
    if(proxy.IsFailure())
     Cout() << "Proxy connection failed. Reason: " << proxy.GetErrorDesc() << "\r\n";
    else
    if(proxy.IsSuccess()) {
     TcpSocket *socket = proxy.GetSocket();
     Cout() << "Succesfully connected via: " << socket->GetPeerAddr() << "\r\n";
     Cout() << socket->GetLine() << "\r\n";
    }
    proxies.Remove(i);
    break;
   }
  }
 }

 for(int i = 0; i < sockets.GetCount(); i++)
  sockets[i].Close();
}
```

There are some small quirks to iron out, but I am planning to upload the complete package next weekend.

Regards,
Oblivion.

Subject: Re: Adding network proxy support to U++
Posted by mirek on Wed, 12 Nov 2014 07:12:00 GMT

View Forum Message <> Reply to Message

Sounds pretty good, looking forward to the code...

Mirek

Subject: Re: Adding network proxy support to U++
Posted by Oblivion on Sat, 22 Nov 2014 13:56:00 GMT
View Forum Message <> Reply to Message

Hello Mirek,

Here I upload the code for review (I can move it to the Bazaar section, if you prefer). It is a complete package, contains API docs and two examples (examples are rudimentary though). This is the first public version, and contains the NetworkProxy, HttpProxy and SocksProxy classes.
As I mentioned before, I used HttpRequest class as a model (and borrowed some of its code too).

Comments, bug reports, fixes, constructive criticism are always welcome.

P.s. If you find it of enough quality, please feel free to add it to U++.

[Package Removed. You can find the final version below]
Regards,
Oblivion

Subject: Re: Adding network proxy support to U++
Posted by Oblivion on Mon, 24 Nov 2014 13:34:33 GMT
View Forum Message <> Reply to Message

Hello Mirek,

There was a bug with the Socks4Request() method. It is now fixed.
Please find the fixed package attached below.

[Package Removed. You can find the final version below]

Regards,
Oblivion.

Subject: Re: Adding network proxy support to U++
Posted by Oblivion on Fri, 19 Dec 2014 14:39:47 GMT
View Forum Message <> Reply to Message

Hello guys,

I've been away for a while.

Any comments on the current NetworkProxy package code?
Do you find it sufficient? Or Is it lacking any necessary features? If so, which are they?
Did you find any bugs?
Is it of enough quality?

I use the package and It works fine here, but any feedback and testing would be very helpful developing it further.
:)

Regards,
Oblivion.

Subject: Re: Adding network proxy support to U++
Posted by mirek on Wed, 24 Dec 2014 11:12:44 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Fri, 19 December 2014 15:39Hello guys,

I've been away for a while.

Any comments on the current NetworkProxy package code?
Do you find it sufficient? Or Is it lacking any necessary features? If so, which are they?
Did you find any bugs?
Is it of enough quality?

I use the package and It works fine here, but any feedback and testing would be very helpful developing it further.
:)

Regards,
Oblivion.

I am sorry for delays. I have in 'todo' list, among bazillion of other things.

One problem is that I have a little practical use for it right now (never had to use other than already supported proxies in my production environments)... hence the slow response.

Mirek

Subject: Re: Adding network proxy support to U++
Posted by Oblivion on Wed, 24 Dec 2014 11:53:07 GMT

View Forum Message <> Reply to Message

Hello Mirek,

No problem, and of course I understand that you are very busy. :)
Please feel free to review it anytime you find it ok to review.

By the way, I can move the package to Bazaar section of this forum, if you prefer.

Regarads,
Oblivion.

Subject: Re: Adding network proxy support to U++
Posted by mirek on Thu, 25 Dec 2014 20:03:18 GMT
View Forum Message <> Reply to Message

Well... I wanted it to go directly to something like Core/Proxy...

Mirek

Subject: Re: Adding network proxy support to U++
Posted by Oblivion on Thu, 25 Dec 2014 22:06:02 GMT
View Forum Message <> Reply to Message

Hello Mirek,

As always, I'd be happy to contribute to U++. :)
If you need my explicit permission, then permission is granted. You can freely add the NetworkProxy package to U++ if you find it of enough quality. (BSD licence is added to the package.)
I was going to upload it to bazaar, because people may look into that section for some new code/classes, and I understand that you are busy.
So here I upload the final code. Below package is, I believe, a complete package covering the basic HTTP and SOCKS4/4a/5 proxy protocols.
One more convenience function, ProxyAccept(), is added, which drastically simplifies SOCKS BIND requests.
From now on, and until you ask from me otherwise, the package is in maintenance mode. I am not going to add any other functions or a new class unless it's neccessary. Only bug fixes, etc. will happen.

Regards,
Oblivion.

File Attachments
1) NetworkProxy Class and Examples (12-29-14).zip, downloaded 378 times

Subject: Re: Adding network proxy support to U++
Posted by Klugier on Sat, 27 Dec 2014 13:37:10 GMT
View Forum Message <> Reply to Message

Hello Oblivion,

Little review for your code. Never use "using namespace Upp" in your header files (.h). More information you can find on StackOverflow: http://stackoverflow.com/questions/5849457/using-namespace-i n-c-headers. Instead of "using namespace Upp" use "NAMESPACE_UPP & END_UPP_NAMESPACE" construction:

NAMESPACE_UPP

```
class NetworkProxy
{
  ...
}
```

END_UPP_NAMESPACE

But in implementation (.cpp file) you will need to use "using namespace Upp;" directive or "NAMESPACE_UPP & END_UPP_NAMESPACE" construction. It dosen't metter in implementation file. Personally, I prefer first option, but for example core Upp code use second option.

Sincerely,
Klugier

Subject: Re: Adding network proxy support to U++
Posted by Oblivion on Mon, 29 Dec 2014 12:12:51 GMT
View Forum Message <> Reply to Message

Hello Klugier,

Thank you very much for reviewing the code, and, of course, for the informative article.
You are right, it is a bad habit, I overlooked the possible problems it can result in. It is now fixed.

Regards,
Oblivion.