Boost asio vs. Qt Network

Qt Network

HTTP, FTP

TCP/IP, UDP

Resolve host names

Proxy

Roaming (Bearer management) (Qt 4.7)

SSL (OpenSSL)

Qt Network Higl Level API

HTTP Request → QNetworkRequest

HTTP Reply → QNetworkReply

send & receive → QNetworkAccessManager

roaming, session management

→ NetworkSession

enum QNetworkConfiguration::BearerType

QNetworkConfiguration::BearerUnknown
QNetworkConfiguration::BearerEthernet
QNetworkConfiguration::BearerWLAN
QNetworkConfiguration::Bearer2G
QNetworkConfiguration::BearerCDMA2000
QNetworkConfiguration::BearerWCDMA
QNetworkConfiguration::BearerHSPA
QNetworkConfiguration::BearerBluetooth

QNetworkAccessManager - async

```
QNetworkAccessManager *manager = new QNetworkAccessManager(this);

connect(manager, SIGNAL(finished(QNetworkReply*)),
        this, SLOT(replyFinished(QNetworkReply*)));

manager->get(QNetworkRequest(QUrl("http://qt-project.org")));
```

Another example

```
QNetworkRequest request;
request.setUrl(QUrl("http://qt-project.org"));
request.setRawHeader("User-Agent", "MyOwnBrowser 1.0");

QNetworkReply *reply = manager->get(request);


connect(reply, SIGNAL(readyRead()), this, SLOT(slotReadyRead()));
connect(reply, SIGNAL(error(QNetworkReply::NetworkError)),
        this, SLOT(slotError(QNetworkReply::NetworkError)));
connect(reply, SIGNAL(sslErrors(QList<QSslError>)),
        this, SLOT(slotSslErrors(QList<QSslError>)));
```

Session management

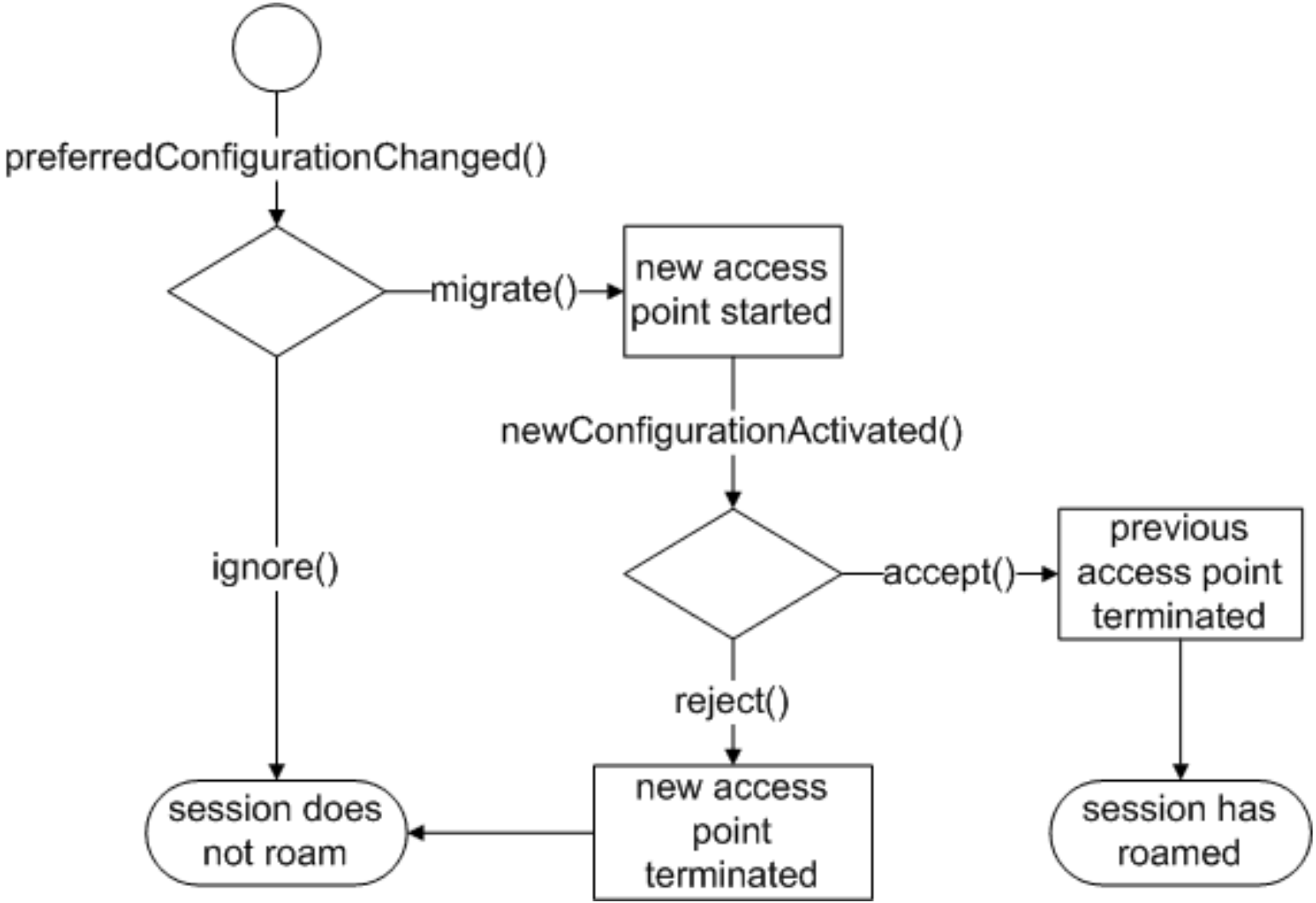
```
QNetworkConfigurationManager mgr;
QNetworkConfiguration ap = mgr.defaultConfiguration();
QNetworkSession *session = new QNetworkSession(ap);

session->open();
```

Roaming → QNetworkSession

void preferredConfigurationChanged(...) [signal]

void ignore() [slot]

void migrate() [slot]


preferredConfigurationChanged()
migrate()
new access point started
newConfigurationActivated()
ignore()
accept()
previous access point terminated
reject()
session does not roam
new access point terminated
session has roamed

TCP/IP

QTcpSocket

QTcpServer

QNetworkProxy

Async - QTcpServer

```
1.  //main.cpp
2.  int main(int argc, char* argv[])
3.  {
4.      QCoreApplication app;
5.      MyServer server;
6.      app.exec();
7.  }
8.
```

```
1.  //MyServer.cpp
2.  MyServer::MyServer(QObject *parent) : QObject(parent)
3.  {
4.      this->server = new QTcpServer(this);
5.      connect(server, SIGNAL(newConnection()),
                this, SLOT(on_newConnection()));
6.      if (!server->listen(QHostAddress::Any, 1234))
7.          //do something in case of error
8.  }
9.
10. void MyServer::on_newConnection()
11. {
12.     QTcpSocket* socket = server->nextPendingConnection();
13.     //do some communication...
14. }
```

Blocking - QTcpServer

```
server->waitForNewConnection(/*msec*/0, /*timedOut*/ 0)

QTcpSocket* socket = server->nextPendingConnection();
```

Blocking - QTcpSocket

```
int numRead = 0, numReadTotal = 0;
   char buffer[50];
   forever {
       numRead  = socket.read(buffer, 50);
       // do whatever with array
       numReadTotal += numRead;
       if (numRead == 0 && !socket.waitForReadyRead())
           break;
   }
```

Proxy

```
QNetworkProxy proxy;
proxy.setType(QNetworkProxy::Socks5Proxy);
proxy.setHostName("proxy.example.com");
proxy.setPort(1080);
proxy.setUser("username");
proxy.setPassword("password");
```

```
QNetworkProxy::setApplicationProxy(proxy);
// and/or
server->setProxy(QNetworkProxy::NoProxy);

//or
server->setProxy(proxy);
```

Boost.asio

boost::asio

Networking - Low level API

Timers

SSL

Serial Ports

Signal handling - (OS signals!)

Blocking & async

connect(...), async_connect(...)
read(...), async_read(...)
write(...), async_write(...)

etc...

Blocking

Networking

ip::tcp::resolver

ip::tcp::acceptor

ip::tcp::socket

boost::asio::io_service

ip:tcp::resolver

```
ip::tcp::resolver resolver(my_io_service);
ip::tcp::resolver::query query("www.boost.org", "http");
ip::tcp::resolver::iterator iter = resolver.resolve(query);
ip::tcp::resolver::iterator end;
while (iter != end)
{
  ip::tcp::endpoint endpoint = *iter++;
  std::cout << endpoint << std::endl;
}
```

```
ip::tcp::socket socket(my_io_service);

ip::tcp::resolver::query query("www.boost.org", "http");

boost::asio::connect(socket, resolver.resolve(query));
```

```
ip::tcp::acceptor acceptor(my_io_service, my_endpoint);

ip::tcp::socket socket(my_io_service);

acceptor.accept(socket);
```