Network Programming for the Rest of Us

Glyph Lefkowitz
Twisted Matrix Labs
glyph@twistedmatrix.com, http://www.twistedmatrix.com/users/glyph

Itamar Shtull-Trauring
Zoteca
itamar@zoteca.com, http://www.itamarst.org/

Abstract

Twisted is a high-level networking framework that is built around event-driven asynchronous I/O. It supports TCP, SSL, UDP and other network transports. Twisted supports a wide variety of network protocols (including IMAP, SSH, HTTP, DNS). It is designed in a way that makes it easy to use with other event-driven toolkits such as GTK+, Qt, Tk, wxPython and Win32. Implemented mostly in Python, Twisted makes it possible to create network applications without having to worry about low-level platform specific details. However, unlike many other network toolkits, Twisted still allows developers to access platform specific features if necessary. Twisted has been used to develop a wide variety of applications, including messaging clients, distributed hash tables, web applications and both open source projects and commercial applications.

1 Introduction

Networked infrastructure development currently exhibits a curious asymmetry. Very high-level infrastructure, such as web servers, and very low-level infrastructure, such as OS-level I/O, have both been actively developed. However, there is no popular middle layer between the two; each high-level abstraction implements all the infrastructure from the base OS level all the way up to its application needs in a specific way.

Development in the area of high performance multiplexing has continued on many platforms, yielding ever-increasing performance. These mechanisms are many and varied: /dev/poll, epoll, select, poll, kqueue[1], completion ports, and POSIX AIO, to name a few. On the other end of the spectrum, many frameworks provide high-level constructs for specific application domains. For example, web application servers provide infrastructure for developing HTTP-based applications. There are few frameworks that provide access to both the low-level and high-level infrastructure required in real networking applications. Web application servers don't provide access to their low-level networking event loop for extension with new protocols, while low-level libraries require far too much work to be usable out of the box.

Twisted is a networking framework suitable for building a wide range of networked servers and clients. Twisted provides portability by using high-level abstractions of protocols, various transports (such as TCP and UDP) and an event loop, allowing deployment of the same code across multiple platforms, primarily Unix and Windows NT.

However, access to platform-specific functionality is a real requirement for many applications. For example, without the ability to access file descriptors directly, it isn't possible to write programs that integrate access to the serial port into the event loop.

Twisted provides low-level access to operating system and event loop specific functionality. On UNIX, for example, it is possible to register file descriptors with select and poll, even though this functionality is not available on Windows. At the same time, one can use the Win32 API to access a serial port through different mechanisms when using Windows. Twisted makes it possible for a user to abstract such a feature themselves if the framework does not provide it already.

Twisted also provides many high-level facilities commonly used by networking applications. A mail server shares a large number of requirements with a web server, such as I/O, protocol parsing, logging and daemonization. In addition, many standardized protocols are shared between applications, e.g.: web mail requires both HTTP and SMTP. Including basic implementations of such protocols saves the developers the need of developing them from scratch.

By providing the full spectrum of functionality for networking applications, both low-level and high-level, developing networking applications is a far simpler task, allowing the developer to concentrate on developing their application rather than reinventing the wheel. Twisted also provides a middle layer so that high-level networking applications may take advantage of low-level advances in functionality and scalability.

This approach contrasts to the two main approaches taken by most networking frameworks. One approach is to use a low-level framework and language (C or C++). The low-level approach gives access to all of the capabilities and APIs provided by the operating system, and if done correctly can result in a very fast program. On the other hand, pervasively using platform-specific functionality results in a platform-specific program, so portability is hindered. Unless carefully audited, C and C++ code is more prone to buffer overflows and system-crashing bugs than their high-level counterparts. This results either in a much longer development process as testing locates all the problems, or in a more fragile system.

Another approach is to provide functionality which only provides access to the lowest common denominator between all supported platforms. This approach is taken by most high-level frameworks. While the "lowest common denominator" approach makes the framework very portable, it means one can't do many basic tasks that only work on specific platforms. For example, the Java platform, which takes this approach, does not support running a server as a daemon on UNIX or an NT Service on Windows, since neither of these features are available on other platforms. This decision pleases no-one by trying to please everyone.

High performance is not a major goal of the Twisted framework. While efficiency and scalability are taken into account during development, flexibility and clean design are more important. This focus stems from the belief that in the real world, as long as performance meets the user's requirements, other factors are more important when choosing a platform. For example, there are a large number of open source and commercial web servers and academic papers describing architectures all of which are significantly faster than the Apache web server. Nevertheless, as of February 2003 Apache runs more than 60% of the sites on the web[3].

To achieve scalable performance that will meet most user's requirements, Twisted uses multiplexing for all I/O operations, and a single thread for almost all computation. As the progression of the Java language has shown[2], blocking, threaded I/O libraries simply do not scale to meet more than the most basic demand. In addition, performance costs associated with context-switching and synchronization, which are exacerbated by Python's global interpreter lock, are eliminated. As others have shown[11], threading is useful only in certain circumstances and should be regarded as a low-level tool. As with all such tools, Twisted has a high-level wrapper that provides a portable and convenient way to integrate threaded code with the event loop when necessary.

Twisted is implemented mostly in Python, with a few parts also available as C extension modules for performance reasons. Python is a very high-level programming language with a great deal of run-time flexibility that allows rapid development of dynamic systems. Despite its high-level nature, it offers access to many system calls necessary for networking, such as select(), socket() and poll(). It provides an excellent base for porting Twisted's functionality to new operating systems.

Thanks to the facilities provided by Python, Twisted is rather small: at the time of this writing Twisted has only 89,000 lines of code in Python, including all of its protocol implementations. The C portion, which only duplicates certain performance-critical python functionality, is even smaller, only 4,000 lines of code in C.

Twisted is currently at version 1.0.3, and is being used by a variety of commercial and open source applications. It is a mature project with a large and active community.

2 The Python Programming Language

Python is a high-level object-oriented programming language, with runtimes written in C and Java (and an experimental .NET CLR runtime). Python is highly portable, and runs on a large number of operating systems — including UNIX and UNIX-like systems, Windows and others. However, unlike Java, Python does not go down the path of the the lowest common denominator. Instead, Python supports platform specific features in addition to common functionality. On Windows, Python can be used with COM and Win32 APIs. On UNIX, Python has access to a large range of the POSIX functionality, from fork() to signals. At the same time, where necessary Python provides common wrappers for low-level functionality — threads in Python use a common API that provides the same functionality on different platforms.

Because of its high-level orientation, Python alleviates the need to deal with memory allocation, array bounds checking and pointers, speeding development and preventing common security issues. Moreover, Python scales upwards when designing complex systems, allowing well designed libraries to provide powerful functionality with simple interfaces.

Twisted builds on Python's flexibility, power and clean design. Python wraps GTK+, Qt, Tk, wxPython and Win32 GUI toolkits, and thus allows Twisted to integrate with these toolkits' event loops. Python's support for sockets, select() and other low-level APIs are wrapped to create Twisted's networking core. Additional Python libraries which wrap C code are also used for various functionality (PyOpenSSL for SSL and TLS, PyCrypto for cryptographic algorithms and so on). Additional low-level functionality is easy to integrate due to the simplicity of extending Python with C or C++.

3 The Event Loop

The event loop is the core of Twisted. It implements a pluggable interface to OS-level functionality such as networking, timers and certain commonly available utilities such as SSL encryption. There are two ways of implementing networking event loops. In one approach, event handlers are called in response to readable or writable events on sockets and then an attempt is made to read or write as much as possible. This method is commonly used with non-blocking sockets. The other approach used is fully asynchronous I/O, where event notification happens when a read or write is finished (e.g. POSIX AIO or Windows' I/O Completion Ports). Currently Twisted implements several non-blocking event loops. The event loop APIs are designed to accommodate fully asynchronous I/O as well, but as yet, no implementations have been released.

An object that implements an event loop in Twisted is called a "reactor". Twisted provides reactors that run on top of select(), poll(), kqueue and Win32 Events. Additionally, it provides reactors that use these same low-level mechanisms, but access them through the APIs of the graphical toolkits, such as GTK+ and Qt. This enables Twisted to run within a graphical application written with these toolkits with no performance impact.

For toolkits which do not provide networking APIs, such as Tk and wxWindows, Twisted provides support modules which will run any reactor at brief intervals as the GUI event loop is running.

On Jython (an implementation of Python written in Java) Twisted provides a Java API based reactor that emulates an event loop using threads.

The reactor implementation may be chosen at runtime, depending on which ones are available and what functionality is required. A Twisted reactor object implements functionality for working with at least some of the following systems:

- TCP
- SSL/TLS
- UDP
- multicast
- Unix sockets
- generic file descriptors
- Win32 events
- process running
- scheduling
- threading.

For example, all of these interfaces are supported on Unix when using the default select() based reactor, except Win32 event support. However, when running on Windows, the same reactor will not support generic file descriptors and Unix sockets.

This support for reactor-specific functionality does not mean that all applications written with Twisted are not portable. The programmer can choose to use platform specific functionality (e.g. use the file descriptor support on Unix to write a curses-based console interface), or to use only those interfaces that are cross-platform and supported in all reactors (e.g. use TCP support to write a telnet-based console interface). The choice is made by the programmer, not the framework.

4 Networking

Twisted has a small networking core, twisted.internet (a Python package), which aims to provide a highly portable socket multiplexing API. This API is based around four fundamental principles.

1. All methods should be named in platform neutral, self consistent and semantically clear ways.
2. The API should be as small and abstract as possible.
3. Low-level functionality should not be disabled or obscured in any way.
4. The same events should be provided from multiple different sources, to allow the same objects to communicate with any semantically identical objects.

Each of these four features has an important impact on the resulting framework's ease of use and implementation. Each will be explored in detail.

4.1 Method Naming

While this may sound like a small detail, clear method naming is important to provide an API that developers familiar with event-based programming can pick up quickly. Obscure names like "kqueue" and "/dev/poll" litter the multiplexing landscape, which can make the already-intimidating concept of event-based programming seem even more arcane.

Since the idea of a method call maps very neatly onto that of a received event, all event handlers are simply methods named after past-tense verbs. All class names are descriptive nouns, designed to mirror the is-a relationship of the abstractions they implement. All requests for notification or transmission are present-tense imperative verbs (see Fig. 1).

The naming is *platform neutral*. There is no reference to select, poll, kqueue, completion ports, or even multiplexing. This means that the names are equally appropriate in a wide variety of environments, as long as they can publish the required events.

It is *self-consistent*. Things that deal with TCP use the acronym TCP, and it is always capitalized. Dropping, losing, terminating, and closing the connection are all referred to as "losing" the connection. This symmetrical naming allows developers to easily locate other API calls if they have learned a few related to what they want to do.

It is *semantically clear*. The semantics of `dataReceived` are simple: there are some bytes available for processing. The semantics remain the same even if the lower-level machinery to get the data is highly complex.

4.2 Small, Abstract API

The methods described above are all extremely abstract, high-level versions of their OS-centric cousins. This allows Twisted to support a wide range of platforms, even those that differ on such fundamental details as statefulness of the low-level API. Application authors are encouraged to only subscribe to events that are relevant to their application, which almost never involves low-level networking events.

By keeping the API small, end-user code is also kept small, by allowing simple ideas to be expressed in very few lines of code. Concise expression can be seen in the canonical example of an extremely brief echo server (see Fig. 2), which uses several of the methods and classes described above. While compa-

Notification	Method
Data received from peer.	dataReceived(data)
Connection established with peer.	connectionMade()
Previously-established connection dropped.	connectionLost(reason)

Request	Method Name
Send data.	write(data)
Listen on TCP port, produce a protocol when connections established.	listenTCP(port, factory)
Connect to remote TCP port.	connectTCP(host, port, factory)
Drop a previously-established connection.	loseConnection()
Run the main reactor in an a loop until it is stopped.	run()

Abstract Object	Class Name
Stream-oriented protocol parser.	Protocol
Stream-oriented protocol parser with extensions for POSIX processes.	ProcessProtocol
HTTP request object.	http.Request
Reactor Pattern interface for TCP operations.	IReactorTCP

Figure 1: Examples of Naming

```
from twisted.internet import protocol
from twisted.internet import reactor

# protocol implementation, writes
# what it receives
class Echo(protocol.Protocol):
    def dataReceived(self, data):
        self.transport.write(data)

# protocol factory, used for state that
# needs to be shared between protocol
# instances. here, that isn't much.
class EchoFactory(protocol.ServerFactory):
    def buildProtocol(self, addr):
        return Echo()
# listen on port 9990 using TCP - listenSSL
# could be used for SSL or TLS
reactor.listenTCP(9990, EchoFactory())
# start the event loop
reactor.run()
```

Figure 2: Echo server

rable servers in other frameworks or languages may be as short or as simple, the authors believe that more complex servers will benefit to a larger degree, as in the Conch server (see Section 8).

4.3 Make Low-Level Information Available

While Twisted provides a great deal of wrapping to allow portability, certain low-level features are often required. Twisted attempts to provide a multi-level approach to accessing this information. For example, connection failures are described by an exception type. However, sometimes the exact errno is more useful information than the exception type. So, developers may detect, in order of specificity:

- That a connection failed.
- How the connection failed, in a general and portable manner.
- What exact error the operating system reported, if the underlying API provides it.

4.4 Similar Events, Different Sources

Run-time polymorphism is a basic feature of object-oriented programming that is too rarely utilized to provide flexible interfaces.

Twisted takes advantage of Python's dynamic typing to minimize the number of different types of events (method calls) that are necessary. At the lowest level, this allows Twisted to generate `dataReceived` calls from both non-blocking- and true asynchronous-style low-level APIs. Similarly, Protocol implementors can ignore most of the distinctions between a TLS and TCP connection, or even a TCP socket and an in-memory class used for testing purposes. All three of these transport types generate `connectionMade`, `dataReceived` and `connectionLost` events, and accept `write` and `loseConnection` events.

Users of the framework can take advantage of this architecture by creating wrappers that indirect these events by supporting the wrapped object's interface. This can be used to establish bandwidth throttling, connection limiting and other such features that are portable both across any implementation of the reactor object and any kind of protocol.

5 Concurrency

5.1 Threading

Despite previously-mentioned problems with Python's threading, sometimes using threads is necessary. It can be especially useful when managing threaded C code, since that code need not be affected by the global interpreter lock. An event handler (i.e. a method handling an event) can not run for more than a very short period, since this will stop all service from other clients of the server, freeze the GUI and so on. Since an event handler must not block, blocking or long running operations have to be delegated to threads. Support for threading does have a price, though — without threads there is no real need to make APIs thread-safe. Once threading is introduced, a whole new set of potential problems need to be dealt with to insure such things as race conditions, deadlocks, etc. do not occur.

Twisted provides a set of threading APIs designed to minimize these issues and isolate potential problem spots as much as possible. Operations that need to be run in a thread are added to a queue, which is then fed to a thread pool. If a thread needs to call a method that is not thread-safe, it can queue the operation via the reactor, and the operation will run in the event loop's thread in the next iteration of the event loop. This gives threads access to all of Twisted's APIs without having to worry about thread-safety issues.

```
from twisted.internet import reactor

c = 1
# this trivial example function is
# not thread-safe
def increment(x):
    global c
    c = c + x

def threadTask():
    # run non-threadsafe function
    # in thread-safe way -- increment(7)
    # will be called in event loop thread
    reactor.callFromThread(increment, 7)

# add a number of tasks to thread pool
for i in range(10):
    reactor.callInThread(threadTask)
```

Figure 3: Using threads

For example, RDBMS access, including Python's generic database access (DB-API), is almost always blocking. Twisted provides a wrapper around DB-API that uses the thread-pool support to allow easy RDBMS use from Twisted applications, hiding the blocking aspect of the API from the user.

5.2 Deferreds

In a threading framework, there is only one way to request data: make a function call, get a result as the return value. In an event driven framework, however, functions are divided into two categories; those whose result will be provided immediately as the return value and those whose result will provided later, as an incoming event.

For example, though a naive programmer may want to structure a request for an HTTP URL as a single blocking call, this would require each HTTP request to spawn a new thread. This approach does not scale beyond a few concurrent requests.

```
# blocking style
try:
    r = blockUntilResult()
except:
    print ``An error has occurred''
else:
    print r

# callback style -- notice lack of
# error handling
def gotResult(r):
    print r
callWhenResult(gotResult)


# Deferred style
def gotResult(r):
    print r
def gotError(e):
    print ``An error has occurred''
deferred = doSomething()
# gotError will also be called on
# exceptions in gotResult:
deferred.addCallback(gotResult)
deferred.addErrback(gotError)
```

Figure 4: Example of Deferred vs. other coding styles

5.2.1 Don't Call Us, We'll Call You

The standard solution for this issue is to refactor a function so that instead of blocking until data is available, it returns immediately, and the caller passes a callback function that will be called once the data eventually arrives. There are several things missing from this simplistic solution. There is no way to know if the data never comes back; no mechanism for handling exceptions. There is no way to distinguish between different calls to the callback function from different sessions. The Deferred class solves these problems, by creating a single, unified way to deal with deferred results — results that are not immediately available. Deferred encapsulates and extends the concept of a callback.

A Deferred instance is a promise that a function will at some point in the future have a result, and so a Deferred is returned from the function instead of the actual result. Callback functions can be attached to a Deferred object, and once it gets a result these callbacks will be called. In addition Deferreds allow the developer to register a callback for an error, with the default behavior of logging the error. This is an asynchronous equivalent of the common idiom of blocking until a result is returned or until an exception is raised. As was mentioned previously, multiple callbacks can be added to a Deferred. The first callback in the Deferred's callback chain will be called with the result, the second with the result of the first callback, and so on. Why is this necessary? Consider a Deferred returned by the Twisted RDBMS wrapper - the result of a SQL query. A web widget might add a callback that converts this result into HTML, and pass the Deferred onwards, where the callback will be used by Twisted to return the result to the HTTP client.

The Deferred abstraction is very powerful in that it allows code that looks quite similar to its blocking equivalent, thus making the code easier to understand and maintain. Even more important is the fact that by using the same abstraction for "blocking" results in all parts of the framework, integrating the various systems together is far easier.

For example, methods published via XML-RPC[7] may not be able to calculate the result of the method immediately. Since the XML-RPC framework supports Deferreds, a method published using the XML-RPC framework can simply return a Deferred that will eventually have the result of the method. Since the RDBMS wrapper also returns Deferreds on the result of e.g. a SELECT query, a XML-RPC method can do a SQL SELECT, attach a callback on the resulting Deferred that massages the result and then returns it. The resulting code is almost identical to the equivalent code in a framework that allows blocking.

6 Application Facilities

In addition to the lower-level facilities for implementing networked clients and servers, Twisted builds on this functionality to provide higher level libraries required in many applications. This paper will only mention some of the services provided by Twisted, but in addition Twisted provides deployment tools, object/relational mapping, RDBMS event loop integration, directory-based dbm-like storage, bandwith throttling, utility Python libraries and much more.

Protocol	Client	Server
HTTP	Y	Y
SSH	Y	Y
SMTP	Y	Y
IRC	Y	Y
Telnet	N	Y
SOCKSv4	N	Y
NNTP	Y	Y
FTP	Y	Y
POP3	Y	Y
XML-RPC[7]	Y	Y
SOAP[8]	N	Y
OSCAR (AIM and ICQ)	Y	N
IMAP	Y	Y

Table 1: Protocols implemented by Twisted. Since Twisted is an open source project, protocols are available based on contributions of code from users and developers.

6.1 The Middle Level - Protocols

In many cases, the main protocol in a networked application will be a common standardized protocol, not a custom designed one. Email clients use SMTP, POP3 and IMAP, news servers use NNTP and so on. Even if a custom protocol is being used, there is frequently a need to use standard protocols in addition. For example, the application may want to send out an email (and thus require SMTP), or provide a web control-panel (HTTP), or allow remote monitoring with SNMP. Twisted provides implementations of many common protocols, without specifying any policies on usage or backend implementation. This allows developers to easily use common protocols out of the box with Twisted, without having to spend the time developing them from scratch. At the same time, because Twisted exposes the lowest level of the protocol, developers can use the protocols without being restricted by policy decisions which may not fit their specific application.

6.2 The High Level - Frameworks

A protocol implementation by itself is not necessarily all that useful. Policies and framework support are needed before a protocol implementation can become a real application. To this end, Twisted includes a number of frameworks providing default policies and support code, implemented on top of the low-level protocol implementations, although of course developers need not use these policies if they so choose. Some (such as twisted.mail, a SMTP and POP3 server which can be deployed as smarthost and mail drop) are still in a preliminary development state, but others are more extensively developed and mature.

6.2.1 twisted.web

twisted.web is a web server framework. It is based on the idea of object publishing: a website is a tree of objects, with the URL corresponding to a specific branch and the resulting objects rendering the result of the HTTP request. twisted.web supports serving static content and CGIs, along with easy integration of objects generating dynamic content. Also supported are XML-RPC and SOAP[8] for simple object publishing (both are simplistic HTTP-based "RPC" protocols).

In addition, twisted.web supports distributed web servers. A HTTP request may be re-sent using Perspective Broker (see 6.3) to another web server that actually handles the request. On UNIX systems, this can be used to allow users to publish homepages with dynamic content (e.g. `http://www.example.com/~itamar/`). Each user runs their own twisted.web server that listens on a Unix socket for PB requests, and the main web server running on port 80 forwards requests for that user to their own web server which is running with the user's security limitations.

Web servers that have been decoupled in this manner provide several advantages. Users can run persistent web services that start up, shut down and restart without affecting the main web server. This approach also simplifies reliably separating privileges between individual users and the web server. The main web server does not need to change user-ID to the user in order to run subprocesses with their security restrictions, so it does not require super-user permissions for anything except the initial binding of port 80.

6.2.2 twisted.news

twisted.news is a NNTP server framework. It supports Usenet integration, moderation and other extended NNTP functionality. Storage is designed to

```
# example of testing SOAP server using
# Python interpreter:
#
# $ python soapserver.py &
#
# $ python
# > from SOAP import SOAPProxy
# > url = 'http://localhost:8080/'
# > p = SOAPProxy(url)
# > p.add(1, 2)
# 3
# > p.add(4, 9)
# 13
#
from twisted.web import soap, server
from twisted.internet import reactor

# all methods beginning with 'soap_'
# are published:
class SOAPAdder(soap.SOAPPublisher):
    def soap_add(self, a, b):
        return a + b
site = server.Site(SOAPAdder())
reactor.listenTCP(8080, site)
reactor.run()
```

Figure 5: SOAP server that publishes "add(a, b)" method

be pluggable; currently supported methods include using a RDBMS, and a simple backend for testing purposes that serializes Python objects to disk.

6.2.3 twisted.words

twisted.words is a messaging and chat framework, designed to work with multiple protocols. It includes a chat protocol implemented using Perspective Broker (see below) and IRC server support for chat clients. In also has a web based interface allowing users to sign up for accounts. This is an example of Twisted's power in integrating multiple protocols and services. In addition, the framework provides infrastructure for automated clients, known as bots. One such bot acts as a bridge to channels on other IRC servers, creating fake user objects corresponding to users on the remote server. This a much more powerful solution than what is possible with most IRC bots, since the bot is integrated into the server, instead of working on the protocol level.

6.3 Perspective Broker

Perspective Broker is a remote method invocation framework. It is a message-oriented, asynchronous, authenticated protocol designed for use with multiple languages. In addition to Python implementation included with Twisted, a full implementation exists for Java and support for other languages is underway.

PB was designed from a frustration with other remote object protocols. The central feature that PB supports is the combination of RMI-level flexibility with the connection of untrusted clients. Most RPC or RMI protocols are designed with the assumption that they will be used in a cluster of closely-bound and ultimately-trusted machines (such as CORBA and DCOM); those that are not (such as SOAP and XML-RPC) tend to be limited, extremely verbose, or both. In addition, authentication is an afterthought in many of these protocols. PB uses the simple, flexible and robust capability security model, inspired by the E language[9].

PB is designed to be the "native" protocol of Twisted, and eventually to combine all the abstractions that other protocols in Twisted support. The distributed webserver functionality in Twisted is implemented by representing HTTP protocol abstractions as PB objects and sending them between different Twisted servers. This approach allows greater flexibility to implement servers talking about a "legacy" protocol than the protocol itself would do: for example, PB/HTTP objects may later be extended to include metadata about load-balancing without breaking compatibility.

7 End-To-End: An Example Spanning Both High- and Low-Levels

As an example of how applications build upon both low-level and high-level framework components, here is an illustration of a simplified XML-RPC request.

When the reactor receives a new connection event from the operating system, it dispatches to the appropriate instance of a subclass of `Factory` to create a new instance of a subclass of `Protocol`; in this case, the factory is a `twisted.web.server.Site`

```
from twisted.web import server
from twisted.web import xmlrpc

class MyProc(xmlrpc.XMLRPC):
    def xmlrpc_add(self, a, b):
        return a + b

from twisted.internet import reactor
reactor.listenTCP(8080, server.Site(MyProc()))
reactor.run()
```

Figure 6: Creating a simple XMLRPC server.

which creates a `twisted.protocols.http.HTTP` instance.

A TCP `Transport`, encapsulating the newly-created socket, is attached to the `Protocol`, and automatically registered with the reactor's event loop. At this point, the reactor awaits read events from the socket. When such events occur, the `Transport` is notified. The `Transport` then reads as much data as possible from the socket, and calls the `Protocol`'s `dataReceived` method with that data. This allows the reactor to abstract very different low-level APIs, such as BSD sockets and POSIX AIO, from the `Protocol`.

At this point, the data is parsed by the Protocol. The HTTP protocol is a subclass of the `LineReceiver` class, which it uses to parse individual lines. `LineReceiver` also supports "raw mode", which enables HTTP to receive lines when parsing headers and raw data when parsing message bodies and/or chunked input.

When a full request is received, a `twisted.web.server.Request` instance is created. At this point, the URL path is used by the `Site` instance to locate a `Resource` instance which represents the HTTP Entity, or "endpoint", for the request. In this case, it is a `twisted.web.xmlrpc.XMLRPC` instance.

The `XMLRPC` instance loads in the XML payload and parses it. Based on the method name within the payload, it calls the appropriate handler method and returns the result.

The result is then generated as a Document Object Model tree, which is serialized to a sequence of bytes. These bytes are written to the `Request` object, which then writes it to the `Transport`. This gets buffered and upon write events on the socket, the `Transport` will be notified and write out the buffered data to the network.

```
from twisted.protocols.basic import LineReceiver

class ExampleHTTP(LineReceiver):
    def connectionMade(self, addr):
        self.state = 'command'
        self.req = SimpleHTTPRequest()

    def lineReceived(self, line):
        if self.state == 'command':
            (self.req.cmd, self.req.url,
             self.req.version) = line.split(' ',2)
            self.state = 'headers'
        elif state == 'headers':
            if line == '':
                if self.req.needsContent():
                    self.setRawMode()
                self.state = 'command'
            else:
                self.appendHeader(line)

    def rawDataReceived(self, data):
        self.req.bufferContent(data)
        if self.req.contentComplete():
            self.factory.dispatchRequest(self.req)
            self.setLineMode(
                self.req.extraContent())
```

Figure 7: `LineReceiver` example: a sketch of an HTTP Protocol

8 Case Study - Conch

Conch is a shell framework, impressive mainly for its implementation of the SECSH protocol (SSHv2). This framework allows users to take advantage of secure shell features in application-level code, as well as providing a full-featured, standalone SSH client and server.

```
class Transport:
    def write(self, data):
        self.buffer += data
        self.activate()

    def readyToWrite(self):
        count = self.skt.send(self.buffer)
        self.buffer = self.buffer[count:]
```

Figure 8: A sketch of a transport's write-buffering.

Conch is a good example of the Twisted design approach providing tangible benefits. Examining some attributes of this system as opposed to other SSH servers, the benefits of a high-level programming language and an asynchronous core will become clearer.

Twisted makes writing servers of this sort easy, since many of the simple features which are considered to be part of implementing a server are either available or unnecessary. For example, in the client, reading the user's password from a terminal is one function already included in Python. This is opposed to OpenSSH's readpass.h and readpass.c, comprising approximately 150 lines of code. Opening an outgoing TCP connection is implemented in Twisted's core, not Conch; in OpenSSH, there are 2 functions, `ssh_create_socket` and `ssh_connect`, which together are almost 200 lines.

Conch currently provides a large subset of OpenSSH's SSHv2 and is included with the Twisted distribution.

8.1 Statistics

While it is quite difficult to empirically substantiate claims like "faster development time" or "reduced code size", some heuristic measurements have been provided which indicate that Conch was implemented quickly, with little cost to efficiency. Since both Conch and all its competitors are ongoing projects with bugfixes happening regularly, it would be difficult to compare development time, but since all implement the same core protocols, implementation complexity can be compared.

Project	SLoC	People
Conch	5,000	1
J2SSH	20,000	7
OpenSSH	64,000	84

Initial benchmarking of Conch shows that it is about the same speed as OpenSSH, though it is faster for certain tasks and slower for others. It can start more sessions per second on the same hardware, but it can transfer fewer encrypted bytes per second (with the Psyco Python accelerator Conch is faster than OpenSSH on throughput and has a much faster connection rate). Considering the vastly simpler implementation and the fact that very little time has yet been spent on optimization, this is very encouraging. Connections per second and bytes per second were tested over loopback on an AMD Athlon 1800+, with 384MB RAM, running Debian GNU/Linux 3.0 (woody), and using OpenSSH client v3.4p1. OpenSSH did 3 connections/sec and 7.4MB/sec. Conch, with Psyco, did 11 connections/sec and 8.1MB/sec, and without Psyco 8 connections/sec and 3MB/sec.

8.2 Features

OpenSSH is not very portable, because it is bound directly to providing UNIX services. Each port is comprised of a separate, subtly different source tree, and all are synchronized against a central repository by using an OpenBSD compatibility layer. It does offer extended functionality such as SSHv1 protocol support. J2SSH is restricted in its functionality because it does not allow the standard operation mode of SSH as a remote shell server (the standard "login to your UNIX account and get a shell prompt" functionality).

Conch has a UNIX-login-savvy server that can replace OpenSSH, "out of the box" (the UNIX specific code in Conch is less than 1000 lines of code out of the total). It is also portable to native Win32, without using UNIX-emulation mechanisms such as Cygwin. The back-end for authentication is easily replaceable, making it feasible to quickly implement an authentication backend that uses Windows authentication information, or a Netinfo database on MacOS X.

Thus, by supporting high-level cross-platform functionality and low-level platform specific features, Conch can function as a replacement for both OpenSSH and J2SSH. It can integrate with the UNIX specific functionality most users expect of an SSH server and client, as does OpenSSH. At the same time, most of the SSH implementation runs on non-UNIX platforms and is easily extendable, as with J2SSH. Thus, one can implement a multi-user terminal based application running inside a single process and accessible via SSH, something which would be vastly more difficult with OpenSSH.

9 Related Work

There is a large amount of freely-available networking code on the internet today: so much so that the fundamental programs that drive our networks are the largest credit to the Open Source community. However, few frameworks endeavor to Twisted's scope and generality, and none that the authors are aware of provide as much integrated functionality as it does.

9.1 SEDA (Sandstorm)

SEDA is a very similar project to Twisted. As it describes itself:

> SEDA is an acronym for staged event-driven architecture, and decomposes a complex, event-driven application into a set of stages connected by queues.[4]

SEDA formally represents the notion of a "Stage" and therefore has more support for gracefully failing in the face of overwhelming load than Twisted does. Many of the abstractions involved are similar.

SEDA does have some multi-protocol support, with applications providing peer-to-peer (Gnutella), email (SMTP) and web (HTTP and HTTPS). While basic protocol functionality is in place, the high-level frameworks to deal with protocol-specific abstractions - similar to twisted.web - are rather light at this point, and Java does not yet provide good standard library support for event-based networking.

SEDA also supports two relatively low-level APIs, the now-standard `java.nio` and their home-grown `nbio`. Sadly, these two APIs do not represent a very abstract way of doing networking. The names of the central abstractions for multiplexing in these packages (`SelectSet` for `nbio`, `Selector` for `java.nio`) imply a certain prejudice in favor of legacy networking APIs. In addition, the choice of Java limits the platform specific functionality available.

Sandstorm or SEDA might be a more appropriate platform for an application which had very serious constraints on its behavior under catastrophic load.

9.2 POE

POE is a framework for creating multitasking programs in Perl[6]:

> POE parcels out execution time among one or more tasks, called sessions. Sessions multitask through cooperation (at least until Perl's threads become mainstream). [...] POE has become a convenient and quick way to write multitasking network applications.

POE supports many of the same features Twisted does, including GUI event loop integration and support for multiple protocols. POE v0.25 has approximately 28,000 lines of Perl code, but there are many components developed for it that are distributed separately whose equivalent would be distributed with Twisted. POE makes heavy use of the rather verbose Perl object-model, however, and issues with the Perl language are pervasive throughout the framework. The first question in the POE FAQ is illustrative:

> The way event handlers receive parameters looks strange at first, but it's a combination of a few common Perl features.

Many of Perl's features create unnecessary problems of communication between modules written by people with differing tastes. Parameter-passing is the most basic feature that an integration system can use. The authors feel that if even something so fundamental will "look strange" to a Perl programmer unfamiliar with POE, this is not an easy-to-use framework.

9.3 ACE[5]

> The ADAPTIVE Communication Environment (ACE) is a freely available, open-source object-oriented (OO) framework that implements many core patterns for concurrent communication software.

The ACE framework provides a very useful addition to the C++ programmer's toolbelt. However, it does not alleviate certain basic problems of the C++ language, such as portability issues arising from differing STL implementations, lack of memory protection and build-process integration issues. While it eases the task of writing communications modules, there is little to leverage outside of ACE to enable integration with multiple GUI toolkits or database drivers. ACE has about 480,000 lines of C++ code.

10 Summary

Twisted provides a wide range of APIs and functionality for implementing networked software, from low-level event loop support to high-level messaging frameworks. The choice to support such a broad range of functionality in one framework has proven to be the right one.

By supporting both high- and low-level operations, Twisted applications can have the best of both worlds – cross-platform functionality together with platform-specific additional functionality (as in Conch). By using a high-level language, development time can be significantly reduced. By using Twisted's integrated support for multiple protocols, developers can add web-based control panels or email alerts to their server applications or develop more novel programs (one application messages an IRC chat channel and sends out email notifications of commits to a CVS repository.)

Twisted is being used by a broad range of organizations and developers. Twisted is being used by government organizations such as NASA, by companies in the USA, France, Australia and other countries, by open source projects and by individual programmers. It is being used in applications such as batch document processing servers, web servers, chat clients, distributed hash tables, caching proxies, IRC bots, educational software, protocol testing, messaging frameworks and many others[10].

Twisted is available online at:

http://www.twistedmatrix.com.

References

[1] Kqueue: A generic and scalable event notification facility - Jonathan Lemon

[2] Java SDK 1.4.1: New IO APIs:
http://java.sun.com/j2se/1.4.1/docs/guide/nio/

[3] Netcraft Survey:
http://www.netcraft.com/survey/

[4] SEDA: An Architecture for Well-Conditioned, Scalable Internet Services (2001) - Matt Welsh, David Culler, Eric Brewer

[5] An Architectural Overview of the ACE Framework (1998) - Douglas C. Schmidt

[6] POE: a framework for creating multitasking programs in Perl: http://poe.perl.org

[7] XML-RPC: http://www.xmlrpc.com/

[8] SOAP: http://www.w3.org/TR/SOAP/

[9] E Language: http://www.erights.org/

[10] Twisted Success Stories:
http://www.twistedmatrix.com/services/success

[11] Why Threads Are A Bad Idea (for most purposes) - John Ousterhout:
http://www.softpanorama.org/People/Ousterhout/Threads/tsld001.htm