NSURLConnection and Beyond: Networking with Cocoa

A Brief History of IP

- Invented by Vint Cerf and Bob Kahn in 1974
- Two main transport protocols: TCP & UDP

TCP

- Part of the original IP specification
- Connection oriented
- Stateful
- Reliable
- Stream-based
- HTTP, SMTP, FTP, SSH

UDP

- Introduced in 1980
- Connectionless
- Stateless
- Unreliable
- Datagrams, not bytes
- VoIP, streaming media, online games

APIs on OS X and iOS

- `NSURLConnection`, `NSStream`, `NSNetService` (Foundation)
- `CFSocket`, `CFStream` (CFNetwork)
- `socket()`, `connect()`, `send()`, `recv()` (BSD)
- Game Kit
- Reachability
- Open source

NSURLConnection

- Synchronous and asynchronous
- Asynchronous uses delegation
- HTTP, HTTPS, FTP

NSURLConnection

- Cookie storage
- Response caching
- Credential storage and authentication
- Transparent support of proxies and SOCKS

Cookies

- Automatic storage of cookies in a response
- Sends stored cookies in future responses
- Finer-grained control of cookies via `NSHTTPCookie`.

Caching

- On-disk (OS X only) and in-memory cache
- Only HTTP and HTTPS
- `NSURLConnection` can control caching, as can its delegate

Authentication

- HTTP Basic and HTTP Digest
- Controlled persistence
- Set up ahead of time with `NSURLCredentialStorage` or on-demand when `NSURLConnection` delegate is called

NSURLConnection

```
- (void) connection:(NSURLConnection *)connection
    didReceiveData:(NSData *)data

- (void) connection:(NSURLConnection *)connection
   didFailWithError:(NSError *)error

- (void) connectionDidFinishLoading:(NSURLConnection *)connection
```

NSURLConnection

HTTP GET

```
NSURL *url = [NSURL URLWithString:@"http://example.com/image.png"];
NSURLRequest *request = [NSURLRequest requestWithURL:url];
NSURLConnection *conn =
   [[NSURLConnection alloc] initWithRequest:request
                                   delegate:self];
[conn start];
```

NSURLConnection

HTTP POST (application/x-www-form-urlencoded)

```
NSURL *url = [NSURL URLWithString:@"http://example.com/form"];
NSMutableURLRequest *request =
    [NSMutableURLRequest requestWithURL:url];
[request setHTTPMethod:@"POST"];
[request   setValue:@"application/x-www-form-urlencoded; charset=UTF-8"
 forHTTPHeaderField:@"Content-Type"];
NSData *formData =
    [@"foo=bar&thing1=thing2" dataUsingEncoding:NSUTF8StringEncoding];
[request setHTTPBody:formData];
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request
                                                        delegate:self];
[conn start];
```

NSURLConnection

HTTP POST (multipart/form-data)

```
NSURL *url = [NSURL URLWithString:@"http://example.com/file-upload"];
NSMutableURLRequest *request =
    [NSMutableURLRequest requestWithURL:url];
[request setHTTPMethod:@"POST"];
[request   setValue:@"multipart/form-data; boundary=mymultipartboundary"
 forHTTPHeaderField:@"Content-Type"];
NSData *formData = ???
[request setHTTPBody:formData];
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request
                                                        delegate:self];
[conn start];
```

NSURLConnection

What's multipart/form-data look like?

```
--myboundary
Content-Disposition: form-data; name="field"
Content-Type: text/plain; charset=UTF-8

this is the field value
--myboundary
Content-Disposition: form-data; name="another-field"
Content-Type: text/plain; charset=UTF-8

this is another field value
--myboundary
Content-Disposition: form-data; name="image"; filename="image.jpg"
Content-Type: image/jpeg

<<raw binary data goes here>>
--myboundary--
```

NSURLDownload

- Only on Mac OS X
- Downloads directly to a disk file instead of memory
- Responses bypass the cache system
- Supports resume for some protocols and MIME types
- No synchronous interface

NSURLDownload

- Part of the URL loading system just like `NSURLConnection`
- Cookie and credential storage, authentication, proxies and SOCKS

NSStream

- Lower level than `NSURLConnection`
- Toll-free bridged to `CFStream`
- OS X: `getStreamsToHost:port:inputStream:outputStream:`
- iOS: `CFStreamCreatePairWithSocketToHost()`

NSStream

- Must call -open before reading or writing
- Streams are blocking
  - You must buffer your reads and writes
- Schedule them on a run loop
  - Polling is possible, but it's neither CPU- nor event loop friendly

NSStream

Can easily wrap any connection with TLS!

```
[inStream setProperty:NSStreamSocketSecurityLevelNegotiatedSSL
               forKey:NSStreamSocketSecurityLevelKey];

NSDictionary *properties =
    [NSDictionary dictionaryWithObjectsAndKeys:
        [NSNumber numberWithBool:NO],
        kCFStreamSSLValidatesCertificateChain,
        nil];
CFReadStreamSetProperty((CFStream *)inStream,
                        kCFStreamPropertySSLSettings,
                        (CFTypeRef)properties));
```

NSStream

- What about UDP?
- `CFStreamCreatePairWithSocket()`

NSStream

Get the native socket with `CFReadStreamCopyProperty()` + `kCFStreamPropertySocketNativeHandle`

```
CFDataRef property = CFReadStreamCopyProperty(stream,
    kCFStreamPropertySocketNativeHandle);
CFSocketNativeHandle socket =
    *((CFSocketNativeHandle *)CFDataGetBytePtr(property));
CFRelease(property);
```

Bonjour discovery & advertisement

- Servers advertise a service, clients discover it
- Clients don’t hard-code addresses or ports
- Only works on the local subnet

Bonjour Advertisement

1. Set up the service's socket port.
2. Initialize an `NSNetService` object and assign a delegate.
3. Publish the service.
4. Respond to messages sent to the delegate.

Bonjour Advertisement

```
int port;

NSSocketPort *socket = [[NSSocketPort alloc] init];
// set up socket for listening

struct sockaddr *addr = (struct sockaddr *)[[socket address] bytes];
if(addr->sa_family == AF_INET)
    port = ntohs(((struct sockaddr_in *)addr)->sin_port);
else if(addr->sa_family == AF_INET6)
    port = ntohs(((struct sockaddr_in6 *)addr)->sin6_port);

NSNetService * service =
   [[NSNetService alloc] initWithDomain:@""
                                   type:@"_myservice._tcp"
                                   name:@""
                                   port:port];

[service setDelegate:delegateObject];
[service publish];
```

Bonjour Discovery

- `NSNetServiceBrowser` finds services

```
[serviceBrowser searchForServicesOfType:@"_myservice._tcp"
inDomain:@""];
```

- Once resolved, `NSNetService` objects contain address and port to connect to service

Berkeley Sockets

- First appeared in 4.2BSD in 1983
- *De facto* C API
- Blocking and non-blocking
- Use it on a background thread

Game Kit P2P

- `GKSession` can create an ad-hoc Bluetooth or Wi-Fi network
- Apps can be servers (advertise a service), clients (searches for services) or peers (do both)
- As with Bonjour, a service type differentiates apps on the network

Game Kit P2P

Server

```
GKSession *session =
    [[GKSession alloc] initwithSessionID:nil
                                displayName:nil
                                sessionMode:GKSessionModePeer];

session.available = YES;

- (void) session:(GKSession *)session
didReceiveConnectionRequestFromPeer:(NSString *)peerID
{
    [session acceptConnectionFromPeer:peerID error:NULL];
}
```

Game Kit P2P

Client

```
GKSession *session =
    [[GKSession alloc] initWithSessionID:nil
                                 displayName:nil
                                 sessionMode:GKSessionModePeer];

session.available = YES;

- (void) session:(GKSession *)session
            peer:(NSString *)peerID
  didChangeState:(GKPeerConnectionState)state
{
    [session connectToPeer:peerID withTimeout:5.0];
}
```

Game Kit P2P

```
- (BOOL) sendData:(NSData *)data toPeers:(NSArray *)peers
withDataMode:(GKSendDataMode)mode error:(NSError **)error
```

```
- (BOOL) sendDataToAllPeers:(NSData *)data withDataMode:
(GKSendDataMode)mode error:(NSError **)error

- (void) receiveData:(NSData *)data fromPeer:(NSString *)peer
inSession:(GKSession *)session context:(void *)context
```

Reachability

- Notifies app of the state of the local network and the theoretical reachability of a remote host
- Network state includes presence of cellular or Wi-Fi network
- Asynchronous, should schedule on thread's run loop
- Use this to know the network state before attempting a connection

Third party code

ASIHTTPRequest

- http://allseeing-i.com/ASIHTTPRequest/
- Wraps `CFNetwork` API
- Callbacks via delegate or blocks
- HTTP
- Amazon S3 request signing, Rackspace Cloud Files, bandwidth throttling, more
- BSD License

Three20

- http://three20.info/
- Wraps `NSURLConnection`/`NSURLRequest`
- Simplifies HTTP POST requests
- Disk cache on iOS
- Apache license

AFNetworking

- https://github.com/gowalla/AFNetworking
- `NSOperation` and blocks
- HTTP GET and POST, image loading, simple REST client
- MIT License

HTTPFetcher

- http://cocoawithlove.com/2011/05/classes-for-fetching-and-parsing-xml-or.html
- Wrapper around `NSURLConnection`
- HTTPFetcher, XMLFetcher, JSONFetcher
- Open source, zlib-style license

OmniNetworking

- http://www.omnigroup.com/company/developer/
- Objective-C wrapper around Berkeley Sockets
- TCP, UDP, multicast clients & servers
- MIT License

AsyncSocket

- http://code.google.com/p/cocoaasyncsocket/
- Wraps `CFSocket`, `CFStream` for TCP and UDP sockets
- Delegate-based, uses run loop
- Public domain

Client Design

Latency

- Latency has a bigger impact on perceived network performance than you think
  - 500 sequential fetches with 100ms of latency = 50,000ms
  - 500 fetches batched into 1 request = 100ms

Mobile Considerations

- Data over 3G puts the radio into high-power (~ 1 W) mode for 4 seconds after last transmission
- Next 15 seconds, drops to approximately ½ of high-power (~ 0.5 W)
- From idle to high-power takes about 1 second

Source: B. Zhao, Q. Zheng and G. Cao, "Energy-Aware Web Browsing in 3G Based Smartphones"

Mobile Considerations

- Don’t assume network conditions based on link type
- Wi-Fi on a plane ≠ Wi-Fi in your house
- Shape behavior based on observed conditions

Server Design

Server Design

	Size	Parser LoC	Parse Time
XML	643.8 KB	9	2.177s
JSON	154.7 KB	1	0.653s
XML plist	249.3 KB	1	0.164s
Binary plist	53.2 KB	1	0.065s

500 element array. XML representation generated by Rails 3.
XML parsed by TouchXML, JSON by TouchJSON, plists by Foundation.
Times from iPhone 3GS.

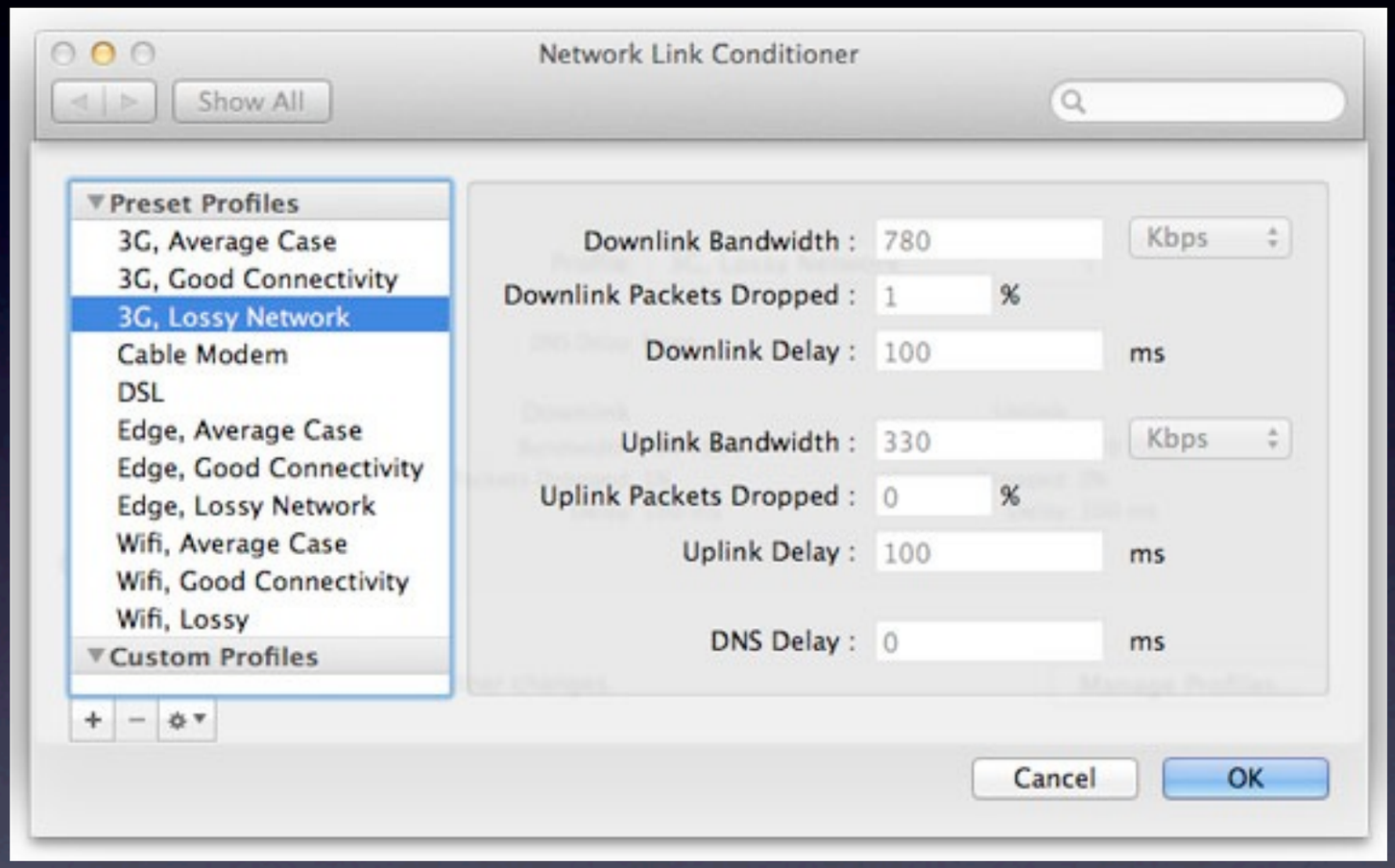
Server Design

	Size	Parser LoC	Parse Time	Size (deflate)
XML	643.8 KB	9	2.177s	32.6 KB
JSON	154.7 KB	1	0.653s	13.2 KB
XML plist	249.3 KB	1	0.164s	13.7 KB
Binary plist	53.2 KB	1	0.065s	32.1 KB

500 element array. XML representation generated by Rails 3.
XML parsed by TouchXML, JSON by TouchJSON, plists by Foundation.
Times from iPhone 3GS.

Tools

- nettop
- Network Link Conditioner



IPv6

- IPv6 added in Mac OS X 10.1, in iOS 4
- Ran out of IPv4 addresses in early 2011
  - IANA exhausted, RIRs will in 2011-12
- Good news: use high-level API and you won't notice!

Summary

- `NSURLConnection` is fine for simple fetches
- Consider a third-party solution for web services
- Non-HTTP over-the-wire protocols: `NSStream`
- Network efficiency is part of the user experience
- IPv6 is not a concern if you stick to higher-level APIs

Steve Madsen
steve@lightyearsoftware.com

Light Year Software, LLC
http://lightyearsoftware.com/

@sjmadsen