Bonjour Network Discovery and Connectivity

Dr Stuart Cheshire
Bonjour Architect

Introduction

- Bonjour overview
  - History
  - Networking made simple
  - Ecosystem
  - Three operations
- Tactical and Practical
  - Coding explained
  - Multitasking in iOS
- Tips and reminders
- Q&A

Bonjour History

TCP/IP
1970s

1980s
AppleTalk

NBP/IP
Discussions
July 1997

November 1999
IETF Zeroconf
Working Group

Apple Announced
Zeroconf Adoption
May 2002

September 2002
First Bonjour Printer

Wide Area Bonjour
Mac OS X 10.4 Tiger
April 2005

June 2007
iPhone launch
iOS Bonjour

Sleep Proxy
Mac OS X 10.6
SnowLeopard
August 2009

June 2011
10th Bonjour
WWDC Presentation

Network Protocols

in 1990

Novell IPX, AppleTalk, TCP/IP, DECnet, Xerox XNS

X.25, ISDN, DSL, Ethernet, WiFi, ATM, Token Ring, LocalTalk

Network Protocols

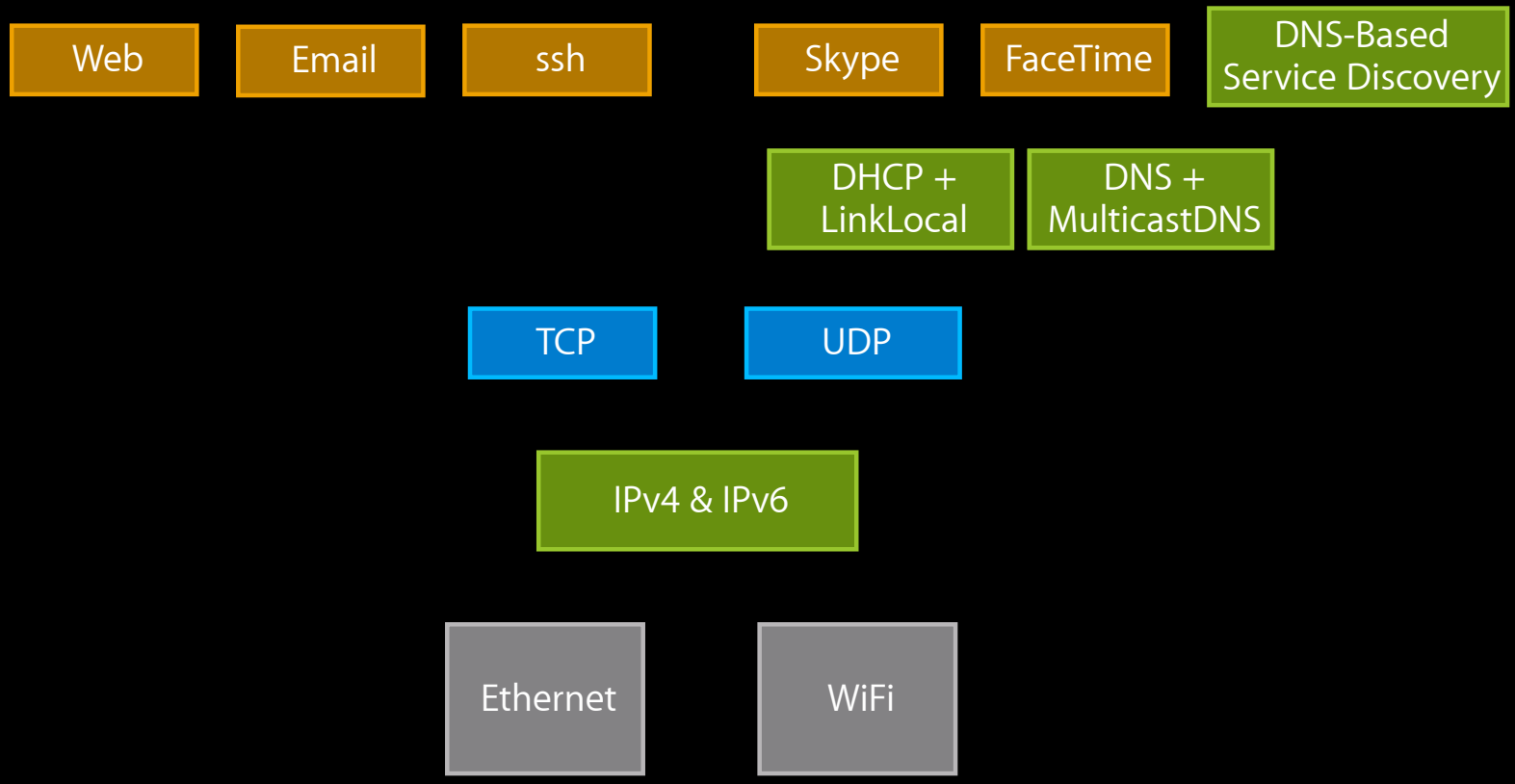
in 2011

Novell IPX, AppleTalk, TCP/IP, DECnet, Xerox XNS

X.25, ISDN, DSL, Ethernet, WiFi, ATM, Token Ring, LocalTalk

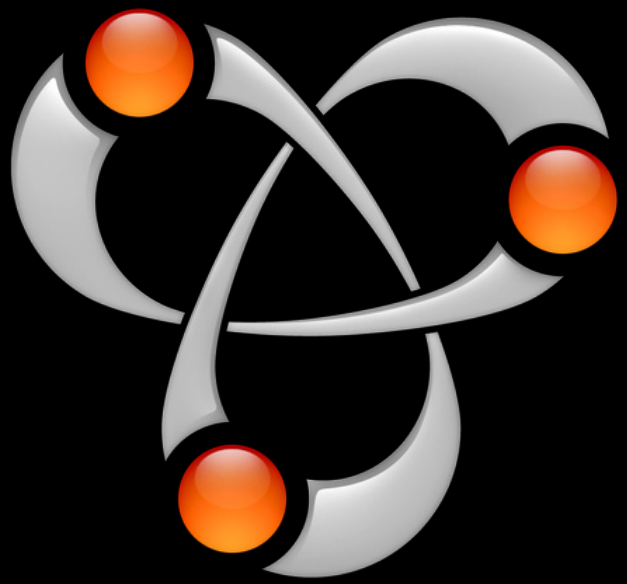
Network Protocols

Internet Protocols



Technology

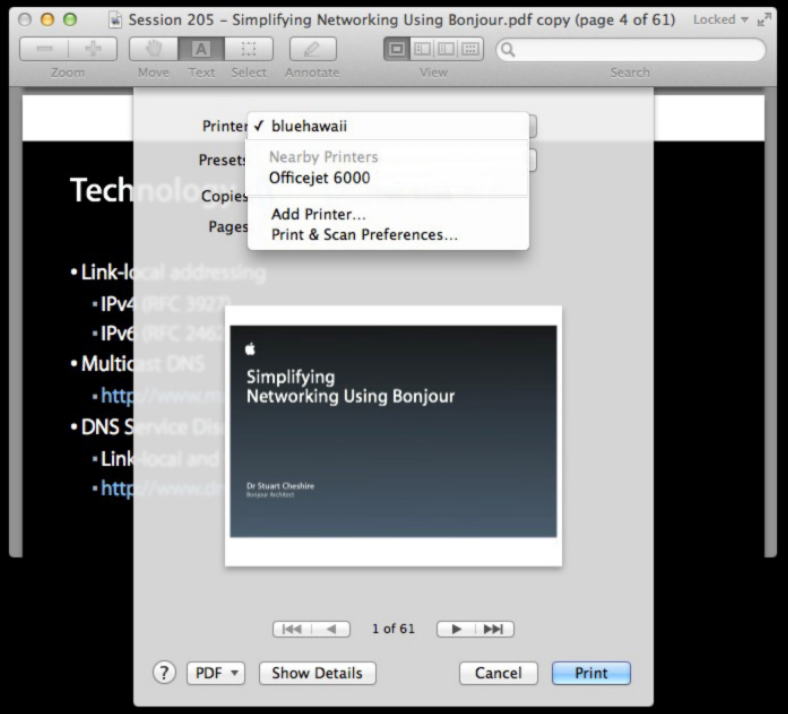
- Link-local addressing
  - IPv4 (RFC 3927)
  - IPv6 (RFC 2462)
- Multicast DNS
  - http://www.multicastdns.org/
- DNS Service Discovery
  - Link-local and wide-area
  - http://www.dns-sd.org/



Bonjour Ecosystem

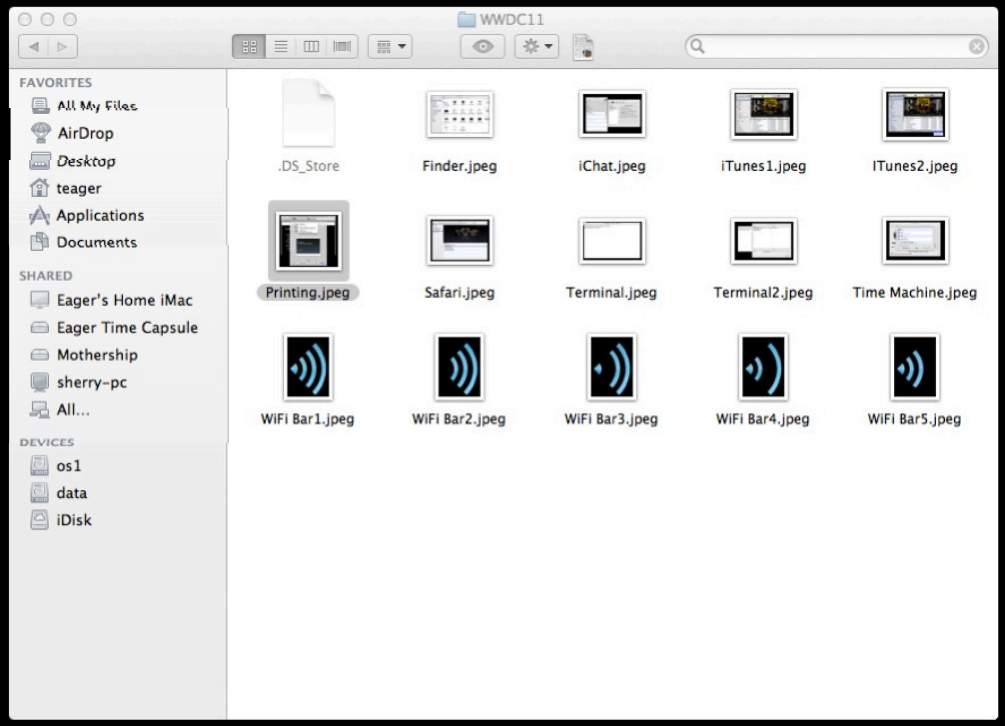
Networking Made Simple in Mac OS X

Printing



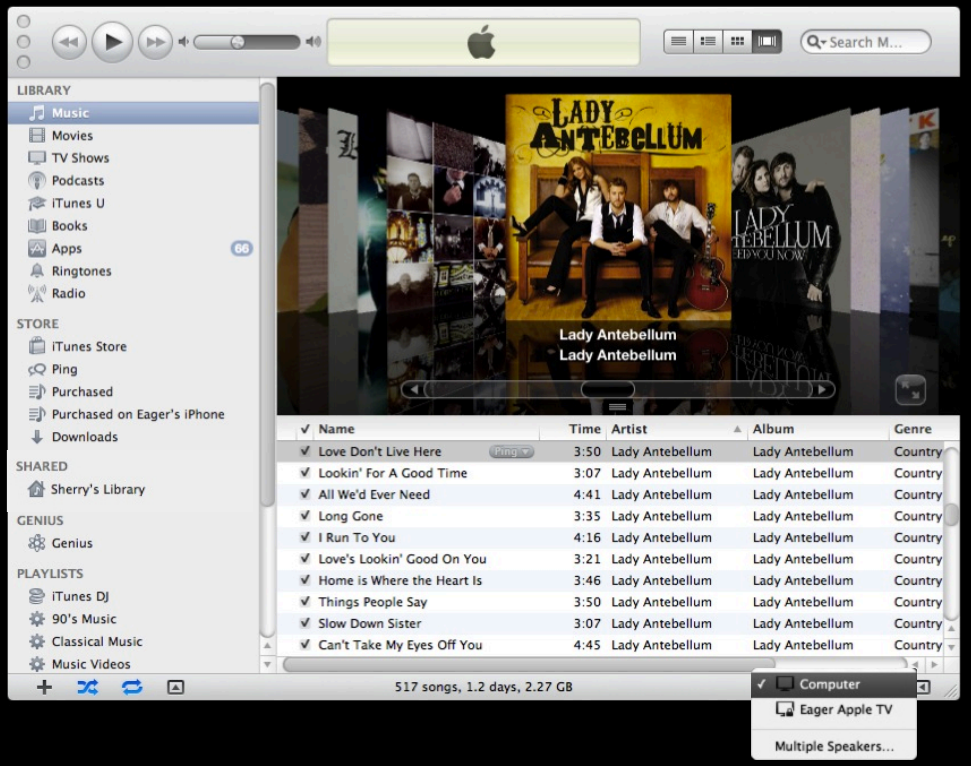
Networking Made Simple in Mac OS X

Finder



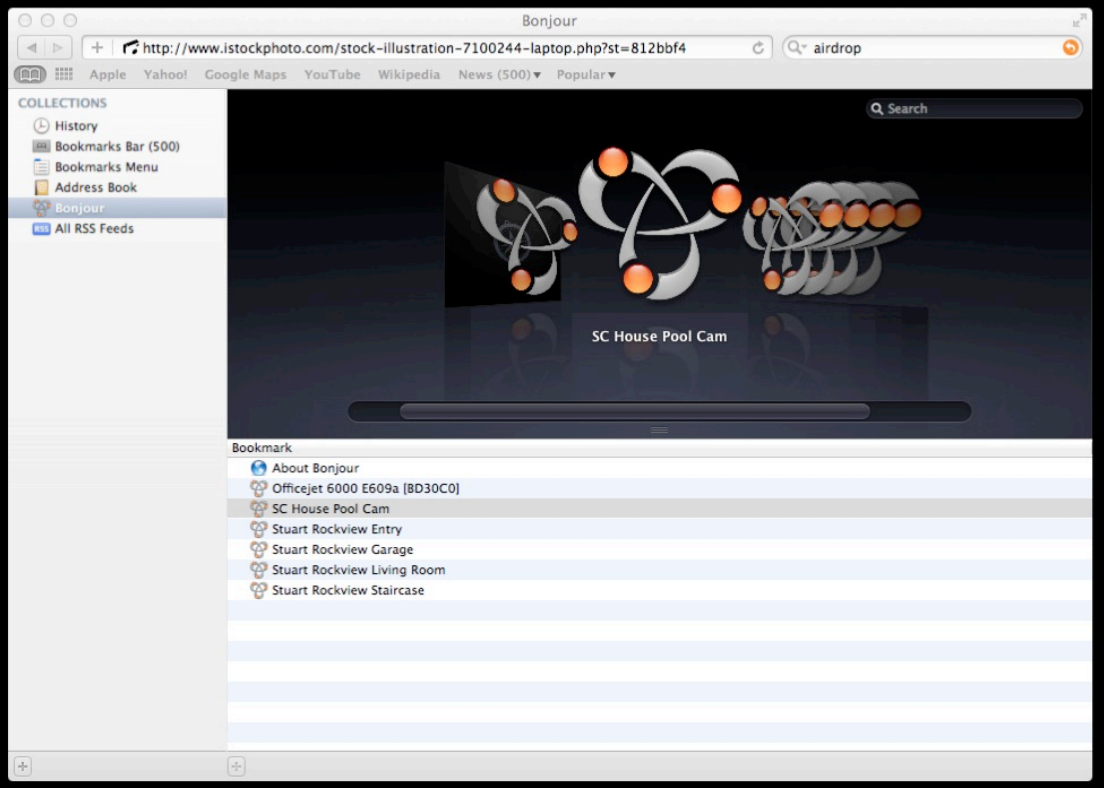
Networking Made Simple in Mac OS X

iTunes



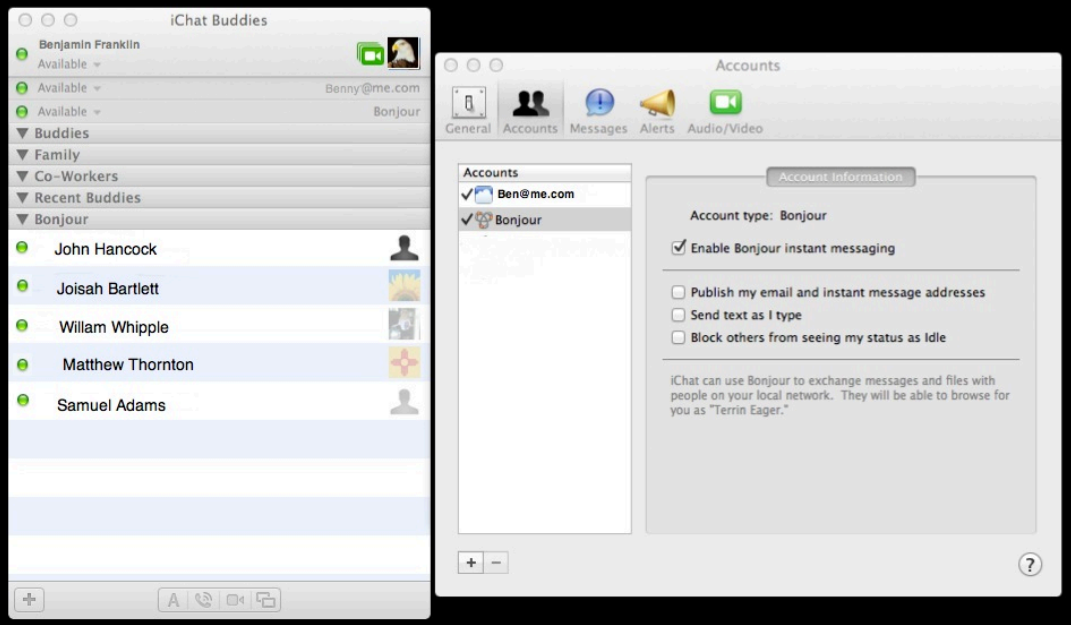
Networking Made Simple in Mac OS X

Safari

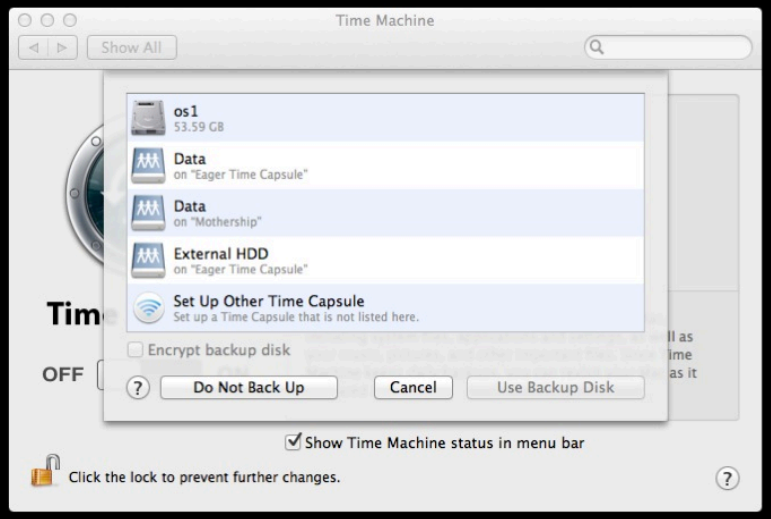


Networking Made Simple in Mac OS X

iChat



Networking Made Simple in Mac OS X



Time Machine

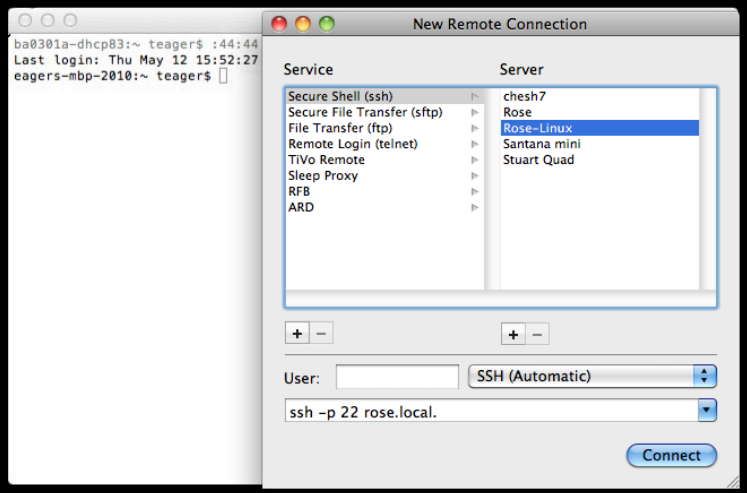
Networking Made Simple in Mac OS X







Terminal

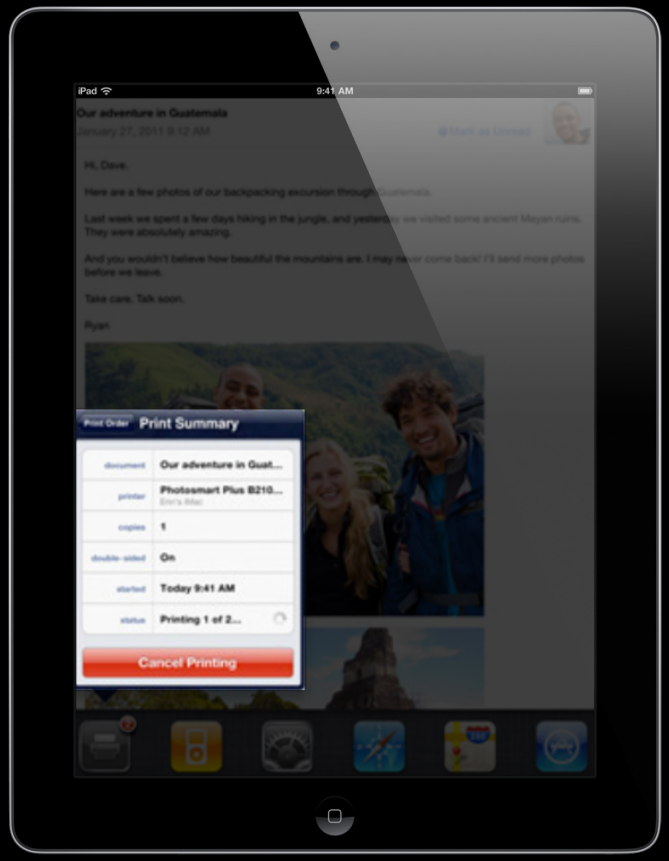


Networking Made Simple in Mac OS X

Sleep

Networking Made Simple in iOS

AirPrint

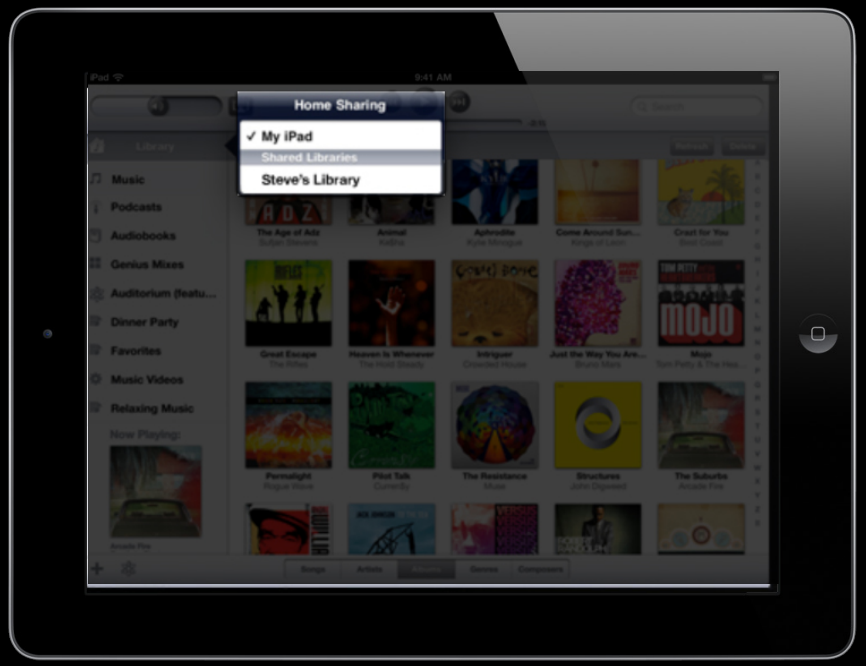


Networking Made Simple in iOS

AirPlay

Networking Made Simple in iOS

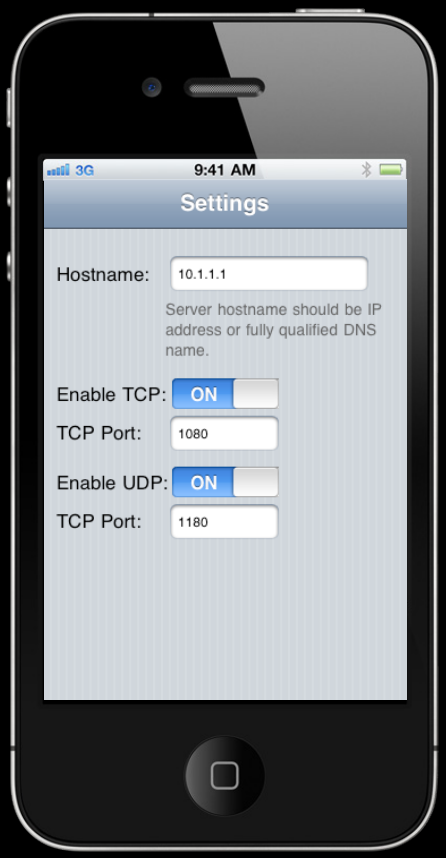
iTunes Home Sharing



Networking Made Simple in iOS

Remote App

Networking Made Simple - Your App?



Bonjour Three Operations

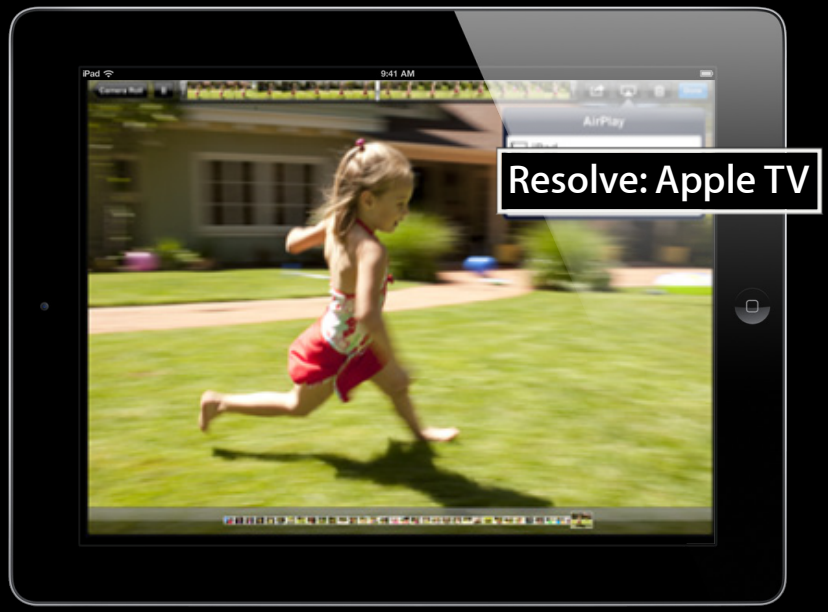
Register

Bonjour Three Operations

Browse

Bonjour Three Operations

Resolve



Browsing Via DNS PTR

DNS Query:

_ipp._tcp.local. PTR ?

Browsing Via DNS PTR

DNS Response(s):

_ipp._tcp.local. PTR

Sales._ipp._tcp.local.
Marketing._ipp._tcp.local.
Engineering._ipp._tcp.local.
3rd Floor Copy Room._ipp._tcp.local.

Components of Service Name

- User-Visible Name
  3rd Floor Copy Room._ipp._tcp.local.
- Service Type/Service Protocol Name
  3rd Floor Copy Room._ipp._tcp.local.
- Domain
  3rd Floor Copy Room._ipp._tcp.local.

Service Types

- A Service Type is identified by:
  - What it does, and
  - How it does it — i.e. what protocol it uses
- List of assigned service types

  http://dns-sd.org/ServiceTypes.html
- Applying for your own is easy
  - Before shipping, register your unique service type
    - So you don't accidentally pick "daap" for "Don's Action Adventure Playground"
- Responsibility moving to IANA (Internet Assigned Numbers Authority)

  http://www.ietf.org/id/draft-ietf-tsvwg-iana-ports-10.txt

Lookup Via DNS SRV

DNS Queries:

Sales._ipp._tcp.local. SRV ?
Sales._ipp._tcp.local. TXT ?

Lookup Via DNS SRV

DNS Responses:

Sales._ipp._tcp.local.	SRV	0 0 631 my-printer.local.
Sales._ipp._tcp.local.	TXT	pdl=application/postscript
my-printer.local.	A	169.254.12.34


AXIS
COMMUNICATIONS


iZON


Back
Baby's Room
08/23/2010
10:20:30 AM
REC
30

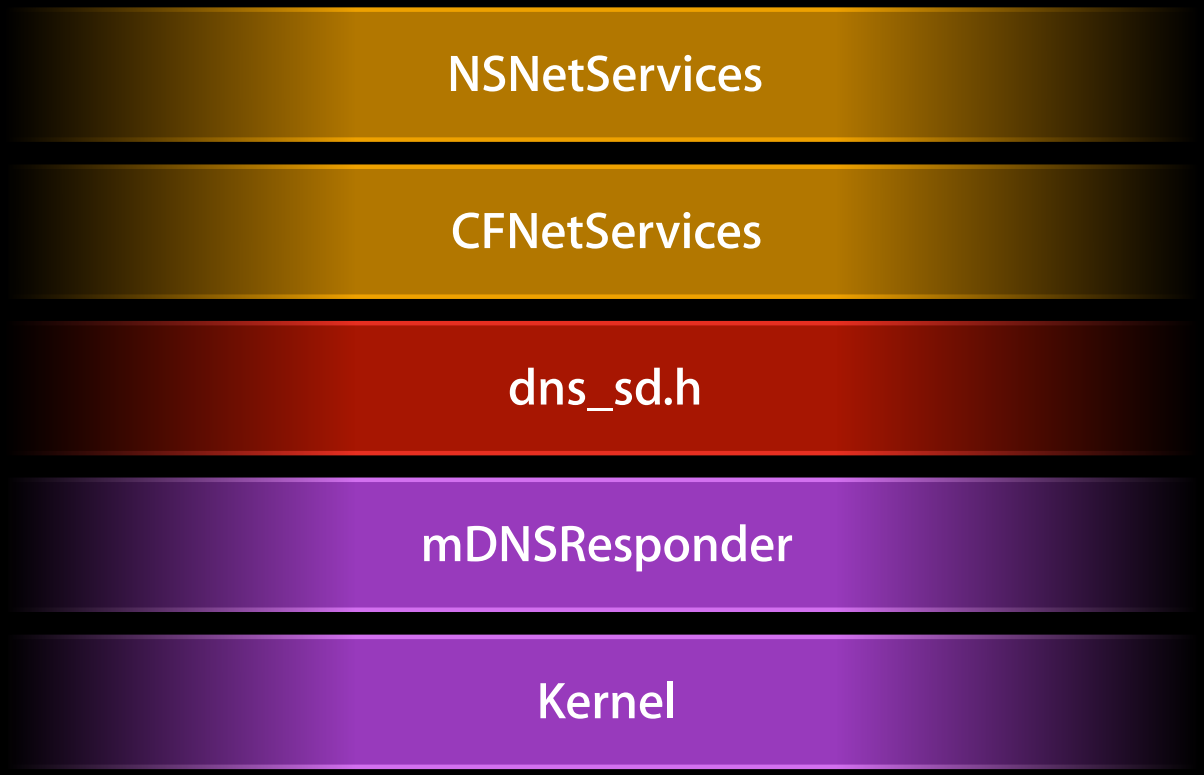
Bonjour - Demo

Dr Stuart Cheshire
Bonjour Architect

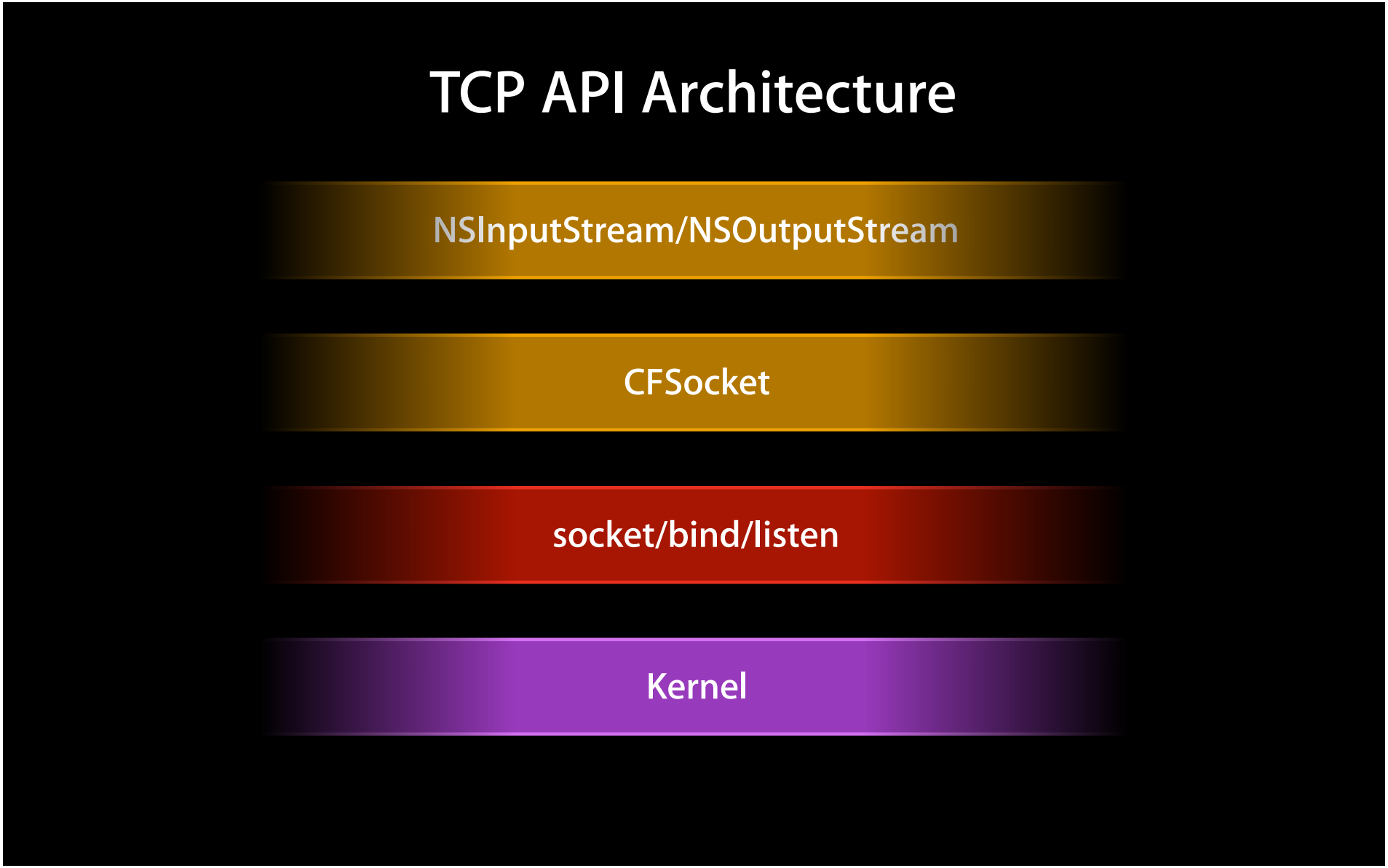
Bonjour—Tactical and Practical

Rory McGuire
Senior Packet Wrangler

Bonjour API Architecture



TCP API Architecture



Server Flow: Register

```
socket/bind/listen
CFSocketCreateWithNative
```

CFSocketRef

Server Flow: Register

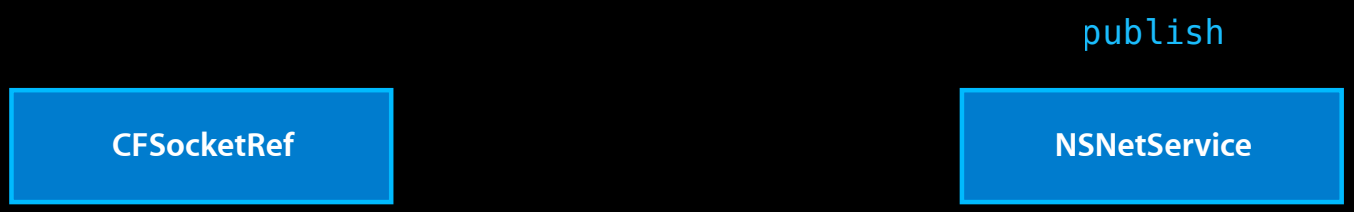
```
socket/bind/listen
CFSocketCreateWithNative
```

CFSocketRef

```
initWithDomain:type:name:port:
```

NSNetService

Server Flow: Register

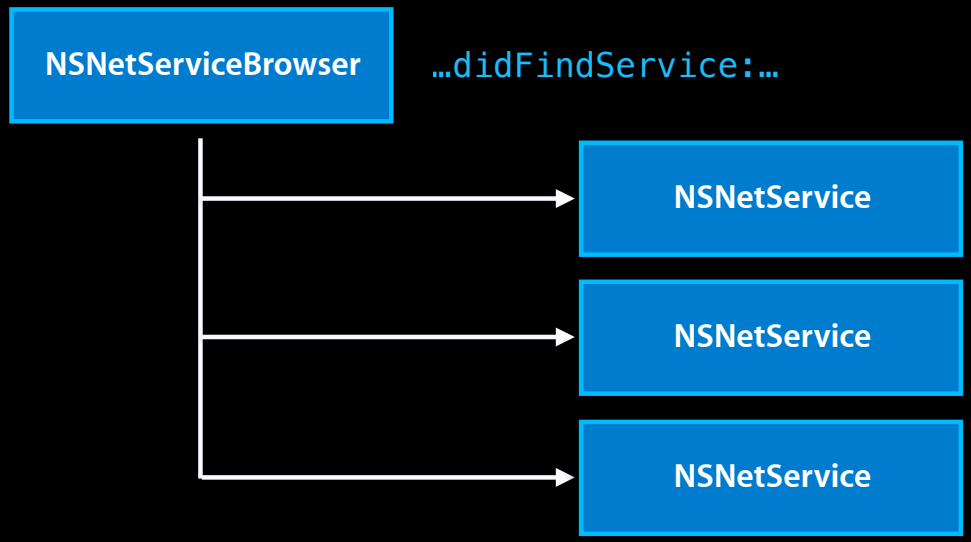


Client Flow: Browse

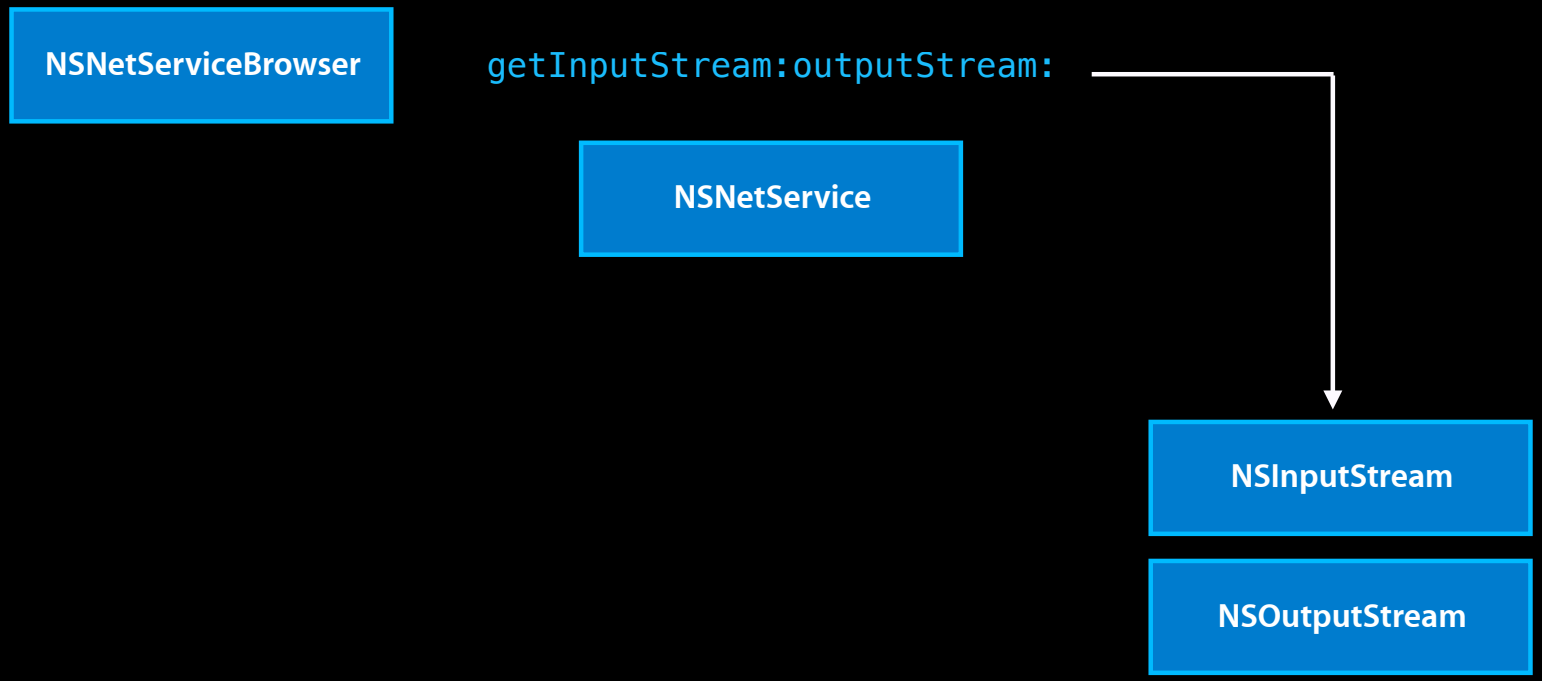
```
searchForServicesOfType:inDomain:
```

NSNetServiceBrowser

Client Flow: Browse



Client Flow: Connect



Client Flow: Connect

NSNetServiceBrowser

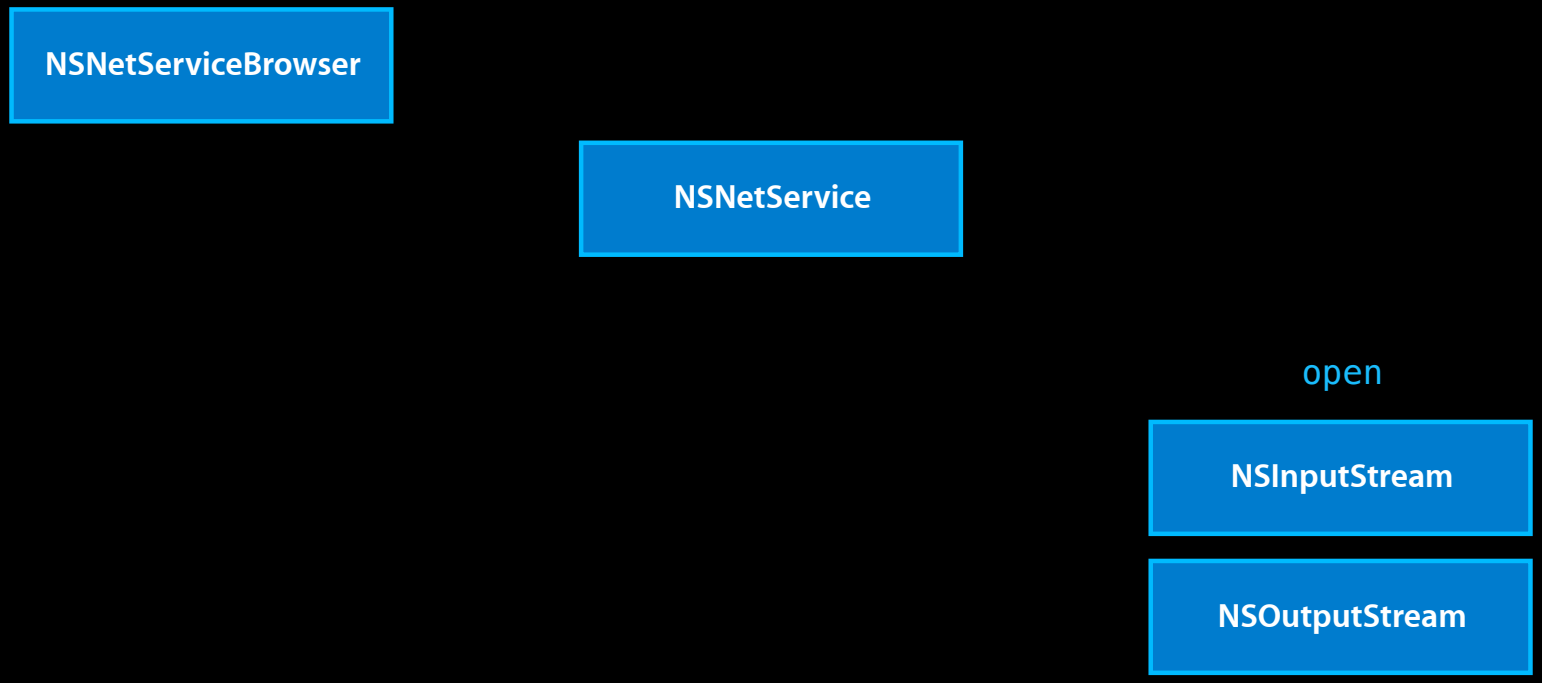
NSNetService

`scheduleInRunLoop:forMode:`

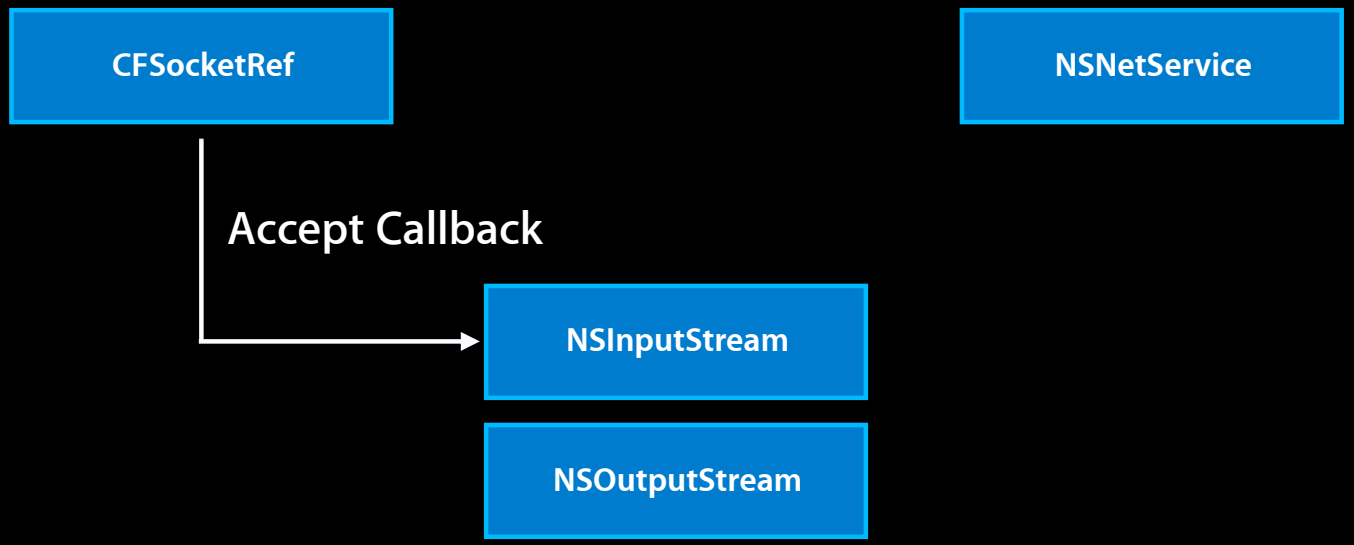
NSInputStream

NSOutputStream

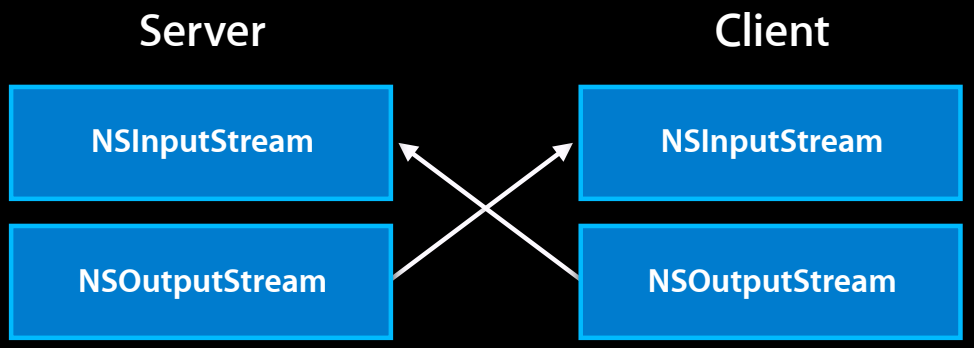
Client Flow: Connect



Server Flow: Accept



Connected


Server
Client
NSInputStream
NSOutputStream
NSInputStream
NSOutputStream

Server: bind/listen IPv4

```
int fd4 = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in sin;
memset(&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_len = sizeof(sin);
sin.sin_port = 0;

int err = bind(fd4, (const struct sockaddr *) &sin, sin.sin_len);

socklen_t addrLen = sizeof(sin);
err = getsockname(fd4, (struct sockaddr *) &sin, &addrLen);

err = listen(fd4, 5);
```

Server: bind/listen IPv4

```
int fd4 = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in sin;
memset(&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_len = sizeof(sin);
sin.sin_port = 0;

int err = bind(fd4, (const struct sockaddr *) &sin, sin.sin_len);

socklen_t addrLen = sizeof(sin);
err = getsockname(fd4, (struct sockaddr *) &sin, &addrLen);

err = listen(fd4, 5);
```

Server: bind/listen IPv6

```
int fd6 = socket(AF_INET6, SOCK_STREAM, 0);

int one = 1;
err = setsockopt(fd6, IPPROTO_IPV6, IPV6_V6ONLY, &one, sizeof(one));

struct sockaddr_in6 sin6;
memset(&sin6, 0, sizeof(sin6));
sin6.sin6_family = AF_INET6;
sin6.sin6_len = sizeof(sin6);
sin6.sin6_port = sin.sin_port;

err = bind(fd6, (const struct sockaddr *) &sin6, sin6.sin6_len);

err = listen(fd6, 5);
```

Server: bind/listen IPv6

```
int fd6 = socket(AF_INET6, SOCK_STREAM, 0);

int one = 1;
err = setsockopt(fd6, IPPROTO_IPV6, IPV6_V6ONLY, &one, sizeof(one));

struct sockaddr_in6 sin6;
memset(&sin6, 0, sizeof(sin6));
sin6.sin6_family = AF_INET6;
sin6.sin6_len = sizeof(sin6);
sin6.sin6_port = sin.sin_port;

err = bind(fd6, (const struct sockaddr *) &sin6, sin6.sin6_len);

err = listen(fd6, 5);
```

Server: Hook up IPv4 CFSocket

```
CFSocketContext    context = { 0, NULL, NULL, NULL, NULL };
CFSocketRef        sock;
CFRunLoopSourceRef rls;

sock = CFSocketCreateWithNative(NULL, fd4,
                                kCFSocketAcceptCallBack,
                                ListeningSocketCallback,
                                &context);
rls = CFSocketCreateRunLoopSource(NULL, sock, 0);
CFRunLoopAddSource(CFRunLoopGetCurrent(),
                   rls,
                   kCFRunLoopCommonModes);

CFRelease(rls);
CFRelease(sock);
```

Server: Hook up IPv4 CFSocket

```
CFSocketContext    context = { 0, NULL, NULL, NULL, NULL };
CFSocketRef        sock;
CFRunLoopSourceRef rls;

sock = CFSocketCreateWithNative(NULL, fd4,
                                kCFSocketAcceptCallBack,
                                ListeningSocketCallback,
                                &context);
rls = CFSocketCreateRunLoopSource(NULL, sock, 0);
CFRunLoopAddSource(CFRunLoopGetCurrent(),
                   rls,
                   kCFRunLoopCommonModes);

CFRelease(rls);
CFRelease(sock);
```

Server: Hook up IPv6 CFSocket

```
CFSocketContext    context = { 0, NULL, NULL, NULL, NULL };
CFSocketRef        sock;
CFRunLoopSourceRef rls;

sock = CFSocketCreateWithNative(NULL, fd6,
                                kCFSocketAcceptCallBack,
                                ListeningSocketCallback,
                                &context);
rls = CFSocketCreateRunLoopSource(NULL, sock, 0);
CFRunLoopAddSource(CFRunLoopGetCurrent(),
                   rls,
                   kCFRunLoopCommonModes);

CFRelease(rls);
CFRelease(sock);
```

Server: Register

```
NSNetService *service =
 [[NSNetService alloc]
  initWithDomain:@""
            type:@"_moodring._tcp."
            name:@""
            port:ntohs(sin.sin_port)];
[service scheduleInRunLoop:[NSRunLoop currentRunLoop]
                   forMode:NSRunLoopCommonModes];
[service setDelegate:self];
[service publish];
```

Server: Register

```
NSNetService *service =
 [[NSNetService alloc]
  initWithDomain:@""
            type:@"_moodring._tcp."
            name:@""
            port:ntohs(sin.sin_port)];
[service scheduleInRunLoop:[NSRunLoop currentRunLoop]
                   forMode:NSRunLoopCommonModes];
[service setDelegate:self];
[service publish];
```

Server: Register

```
NSNetService *service =
 [[NSNetService alloc]
  initWithDomain:@""
            type:@"_moodring._tcp."
            name:@""
            port:ntohs(sin.sin_port)];
[service scheduleInRunLoop:[NSRunLoop currentRunLoop]
                   forMode:NSRunLoopCommonModes];
[service setDelegate:self];
[service publish];
```

Server: Register

```
NSNetService *service =
 [[NSNetService alloc]
  initWithDomain:@""
            type:@"_moodring._tcp."
            name:@""
            port:ntohs(sin.sin_port)];
[service scheduleInRunLoop:[NSRunLoop currentRunLoop]
                   forMode:NSRunLoopCommonModes];
[service setDelegate:self];
[service publish];
```

Server: Register

```
NSNetService *service =
 [[NSNetService alloc]
  initWithDomain:@""
            type:@"_moodring._tcp."
            name:@""
            port:ntohs(sin.sin_port)];
[service scheduleInRunLoop:[NSRunLoop currentRunLoop]
                   forMode:NSRunLoopCommonModes];
[service setDelegate:self];
[service publish];
```

Server: Register Callbacks

```
- (void)netServiceDidPublish:(NSNetService *)sender {
 NSString *name = [sender name];
 NSLog("My name is: %@", name);
}

- (void)netService:(NSNetService *)sender
  didNotPublish:(NSDictionary *)errorDict {
 NSLog("Error publishing: %@", errorDict);
}
```

Server: Accept Callback

```
void ListeningSocketCallback(CFSocketRef sock,
CFSocketCallBackType type, CFDataRef address, const void *data,
void *info)
{
    int fd = * (const int *) data;
    CFReadStreamRef   readStream;
    CFWriteStreamRef  writeStream;
    NSInputStream  *  inputStream;
    NSOutputStream *  outputStream;

    CFStreamCreatePairWithSocket(NULL, fd, &readStream,
                                 &writeStream);
    inputStream  = [NSMakeCollectable(readStream ) autorelease];
    outputStream = [NSMakeCollectable(writeStream) autorelease];

    [inputStream setProperty:(id)kCFBooleanTrue
      forKey:(NSString *)kCFStreamPropertyShouldCloseNativeSocket];
}
```

Client: Browse for Services

```
NSNetServiceBrowser *browser =
 [[NSNetServiceBrowser alloc] init];
 [browser setDelegate:self];
 [browser
  searchForServicesOfType:@"_moodring._tcp."
  inDomain:@""];
```

Client: Browse for Services

```
NSNetServiceBrowser *browser =
 [[NSNetServiceBrowser alloc] init];
 [browser setDelegate:self];
 [browser
  searchForServicesOfType:@"_moodring._tcp."
  inDomain:@""];
```

Client: Browse for Services

```
NSNetServiceBrowser *browser =
 [[NSNetServiceBrowser alloc] init];
 [browser setDelegate:self];
 [browser
  searchForServicesOfType:@"_moodring._tcp."
  inDomain:@""];
```

Client: Browse Callbacks

```
- (void)netServiceBrowser:
   (NSNetServiceBrowser *)netServiceBrowser
  didFindService:(NSNetService *)service
  moreComing:(BOOL)moreComing {
 [self.model addObject:service];
 if (!moreComing) [self.tableView reloadData];
}
```

Client: Browse Callbacks

```
- (void)netServiceBrowser:
   (NSNetServiceBrowser *)netServiceBrowser
  didFindService:(NSNetService *)service
  moreComing:(BOOL)moreComing {
 [self.model addObject:service];
 if (!moreComing) [self.tableView reloadData];
}
```

Client: Browse Callbacks

```
- (void)netServiceBrowser:
   (NSNetServiceBrowser *)netServiceBrowser
  didRemoveService:(NSNetService *)service
  moreComing:(BOOL)moreComing {
 [self.model removeObject:service];
 if (!moreComing) [self.tableView reloadData];
}
```

Client: Resolve and Connect

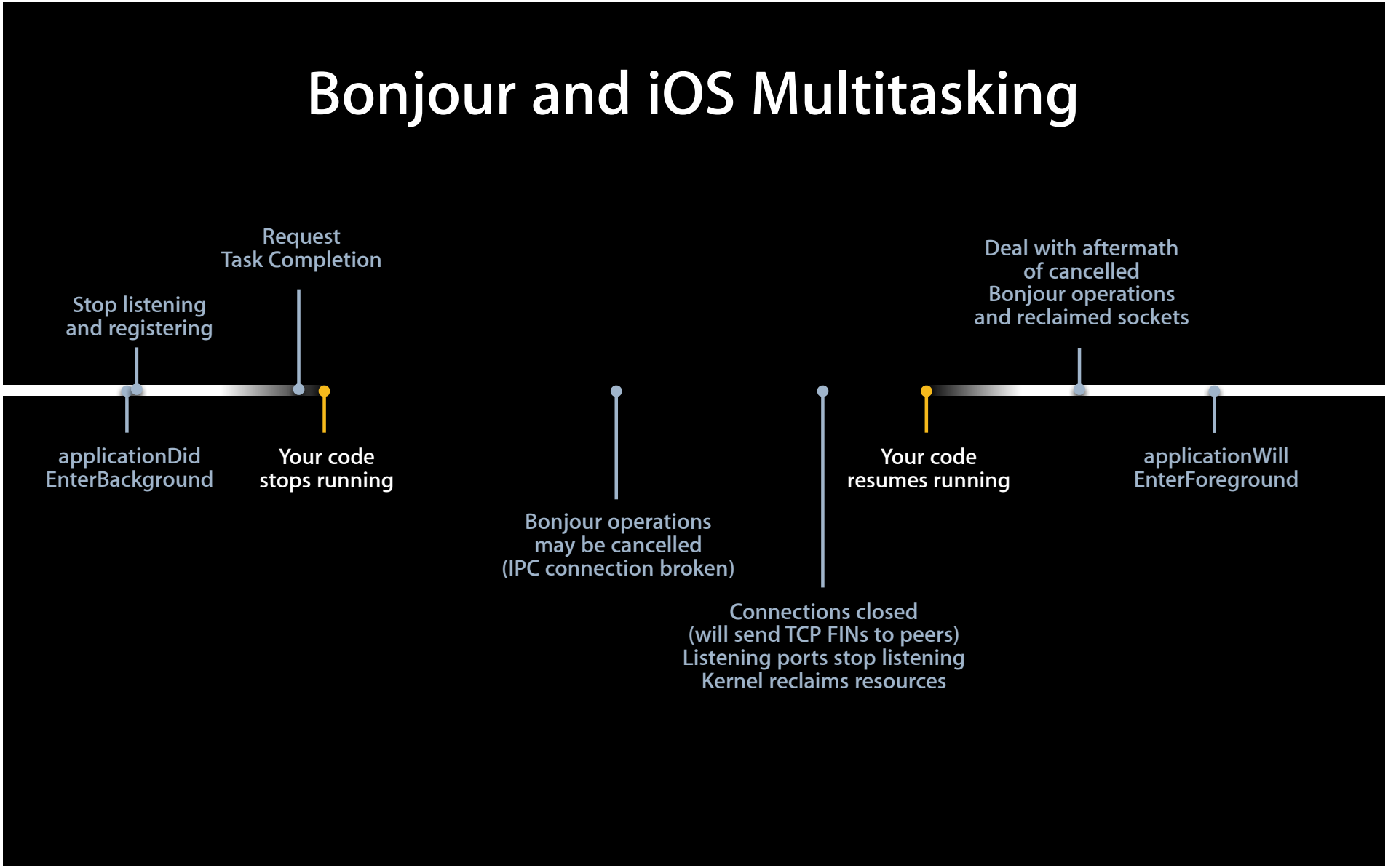
```
NSNetService   * service = TheServiceTheUserSelected();
NSInputStream  * inStream;
NSOutputStream * outStream;

[service getInputStream:&inStream
           outputStream:&outStream];

inStream.delegate = self;
outStream.delegate = self;

[inStream scheduleInRunLoop:[NSRunLoop currentRunLoop]
                    forMode:NSDefaultRunLoopMode];
[outStream scheduleInRunLoop:[NSRunLoop currentRunLoop]
                     forMode:NSDefaultRunLoopMode];

[inStream open];
[outStream open];
```

Client: Resolve and Connect

```
NSNetService   * service = TheServiceTheUserSelected();
NSInputStream  * inStream;
NSOutputStream * outStream;

[service getInputStream:&inStream
           outputStream:&outStream];

inStream.delegate = self;
outStream.delegate = self;

[inStream scheduleInRunLoop:[NSRunLoop currentRunLoop]
                    forMode:NSDefaultRunLoopMode];
[outStream scheduleInRunLoop:[NSRunLoop currentRunLoop]
                     forMode:NSDefaultRunLoopMode];

[inStream open];
[outStream open];
```

Bonjour and iOS Multitasking

Bonjour and iOS Multitasking



iOS Multitasking Aftermath

- After resume, all Bonjour and networking may be broken for your app
  - Registrations may be gone
  - Discovery may have stopped

Suggestion

If browse terminated by iOS, auto-dismiss your browsing UI

If the user wishes, they can browse again

- See Technical Note TN2277

Client: Browser Callback

```
- (void)netServiceBrowser:
   (NSNetServiceBrowser *)netServiceBrowser
  didNotSearch:(NSDictionary *)errorInfo {
 // Application resumed from background
 // and Bonjour browsing operation was cancelled
 // so we restart it now.
 [self restartBrowseAndUpdateUI];
}
```

Bonjour and Peer-to-Peer

P2P Optional

When linking against iOS 5 SDK

Opt-in flag exposed in dns_sd.h API

Tips and Reminders

Use Asynchronous Calls

- DNS timeout is 30 seconds
- iPhone watchdog timer is 20 seconds
  - Do a blocking DNS call on your main thread
  - DNS server is slow or unresponsive
  - Your app dies
- Use NS API and Grand Central Dispatch

Browse Call—The Right Way

- No refresh button
- No open-ended browsing
- Browse when requested by user, not constantly
- Stop browsing when not displaying browse UI
- UI design: Use Windows, not pull-down menus
  - Traditionally, menus not expected to change once displayed
  - Users generally more comfortable with window content updating
- Resolve and connect when requested by user, not every service you find

Browse vs. Resolve

Joe's Printer	joe.local	9100	pdl=application/postscript …
Sally's Printer	sally.local	9100	pdl=application/postscript …
Jim's Printer	jim.local	9100	pdl=application/postscript …
Penny's Printer	penny.local	9100	pdl=application/postscript …
Paul's Printer	paul.local	9100	pdl=application/postscript …
Mary's Printer	mary.local	9100	pdl=application/postscript …

Saving Services You Found

- Bad ideas
  - Save just the IP address
  - Save the IP address and port number
  - Save the host name and port number
- The right way
  - Late binding is the key
  - Service is identified by three-tuple: { Name, Type, Domain }
  - Save { Name, Type, Domain } tuple

Service Naming and Registering

- Unique identifier for every service type
- 15 characters max
- Letters, digits, and hyphens allowed

http://www.dns-sd.org/ServiceTypes.html

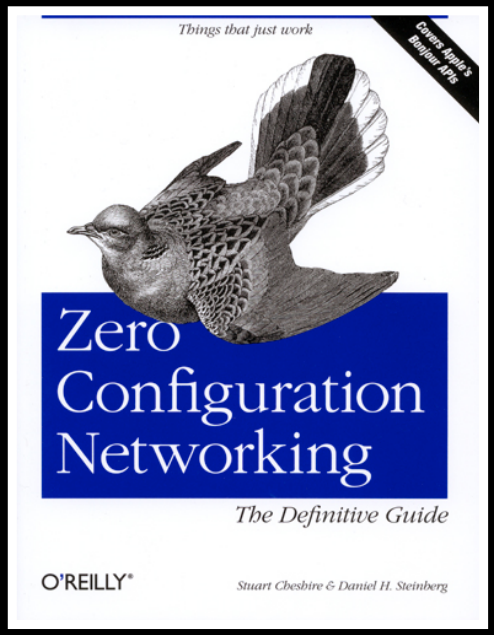
More Information

Craig Keithley
Bonjour Technology Evangelist
keithley@apple.com

Paul Danbold
Core OS Technology Evangelist
danbold@apple.com

Documentation
Bonjour Developer Web Page
BonjourWeb & WiTap Sample Code
http://developer.apple.com/bonjour

Apple Developer Forums
http://devforums.apple.com



Related Sessions

Core OS Networking, Key Principles	Marina Tuesday 9:00AM
Core OS Networking In-Depth	Pacific Heights Wednesday 11:30AM
Blocks and Grand Central Dispatch in Practice	Pacific Heights Wednesday 10:15AM
Mastering Grand Central Dispatch	Pacific Heights Thursday 10:15AM

Labs

Network Lab	Core OS Lab A Thursday 4:30PM-6:00PM

Summary

- Bonjour makes networking easy and reliable
- Use asynchronous calls—do not get bitten by the watchdog
- Let the user decide when they have waited long enough
- Use late binding—{ Name, Type, Domain } tuple
- Handle errors when application resumes from background
- Register your service types


WWDC2011