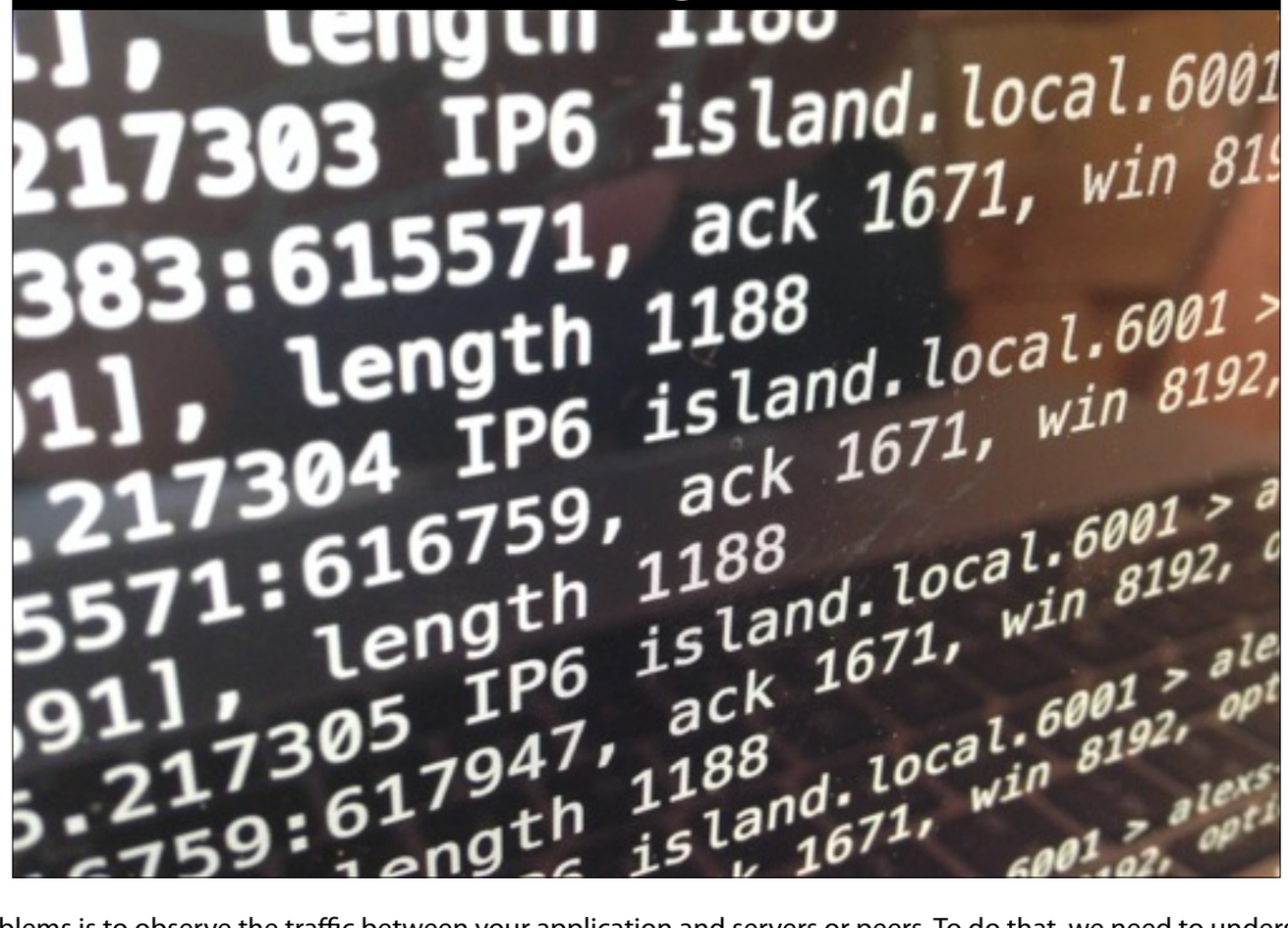
Debugging: Tricks, Tips & Tools

Alex Vollmer
http://alexvollmer.com
@alexvollmer

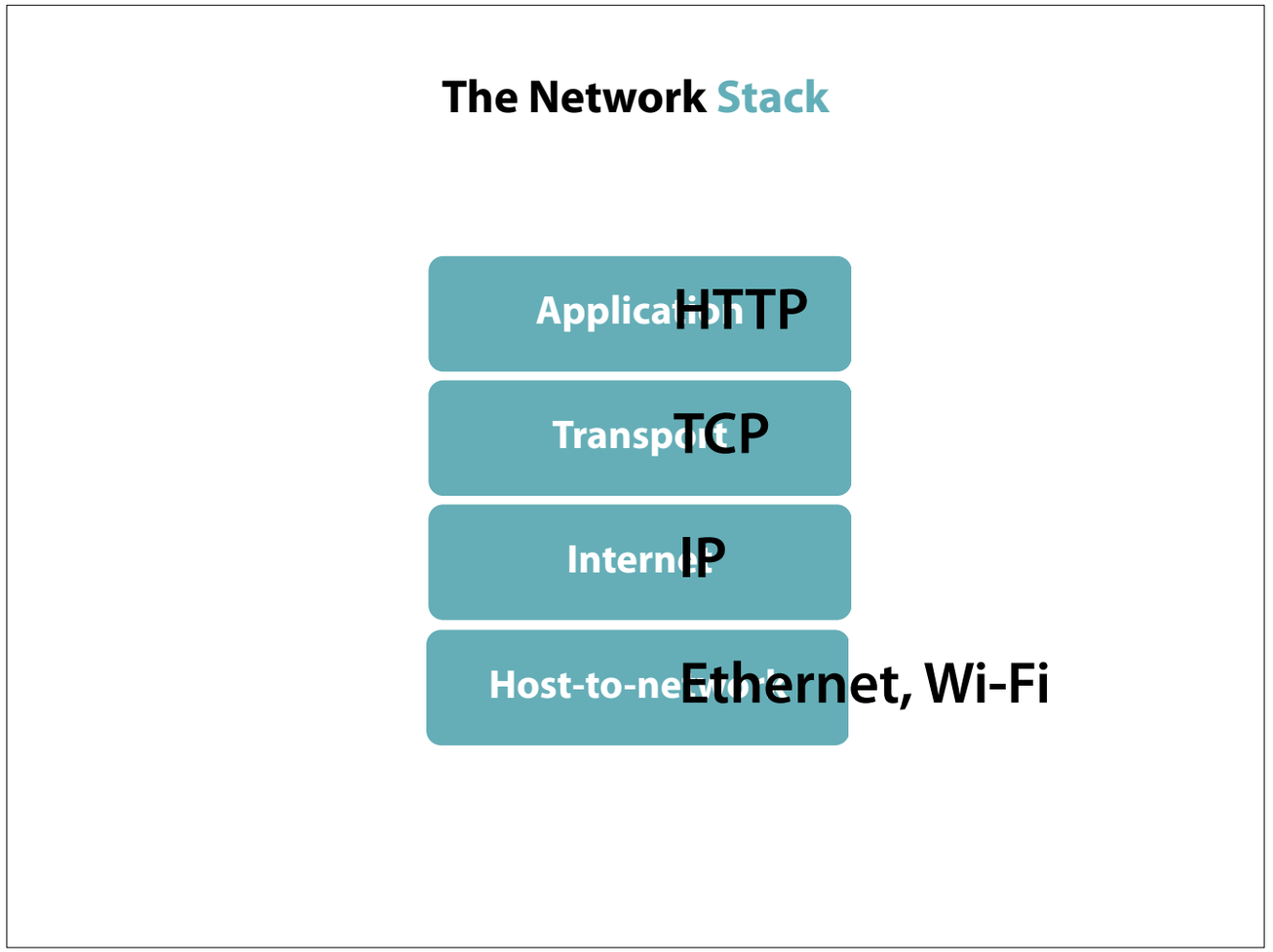
pluralsight
hardcore dev and IT training

Knowing the Foundation networking APIs isn't enough to build successful apps. You also need to know how to troubleshoot when things don't go as expected. In this module we're going to look at several techniques and tools for dealing with networking issues.

Observing Traffic



One of the best ways to debug networking problems is to observe the traffic between your application and servers or peers. To do that, we need to understand a bit about how networks are constructed.



Network engineers often describe networks in layers of abstraction.

¶

Textbooks often describe seven layers, but in the world of the Internet, there are just these four: Application, Transport, Internet and Host-to-Network

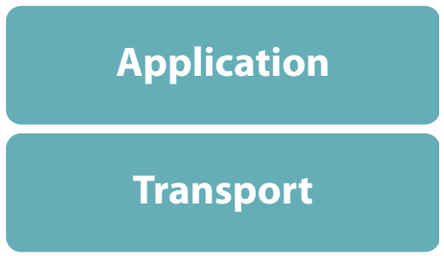
¶

Each layer deals with it's own specific details and relies on the abstractions of the layer below it.

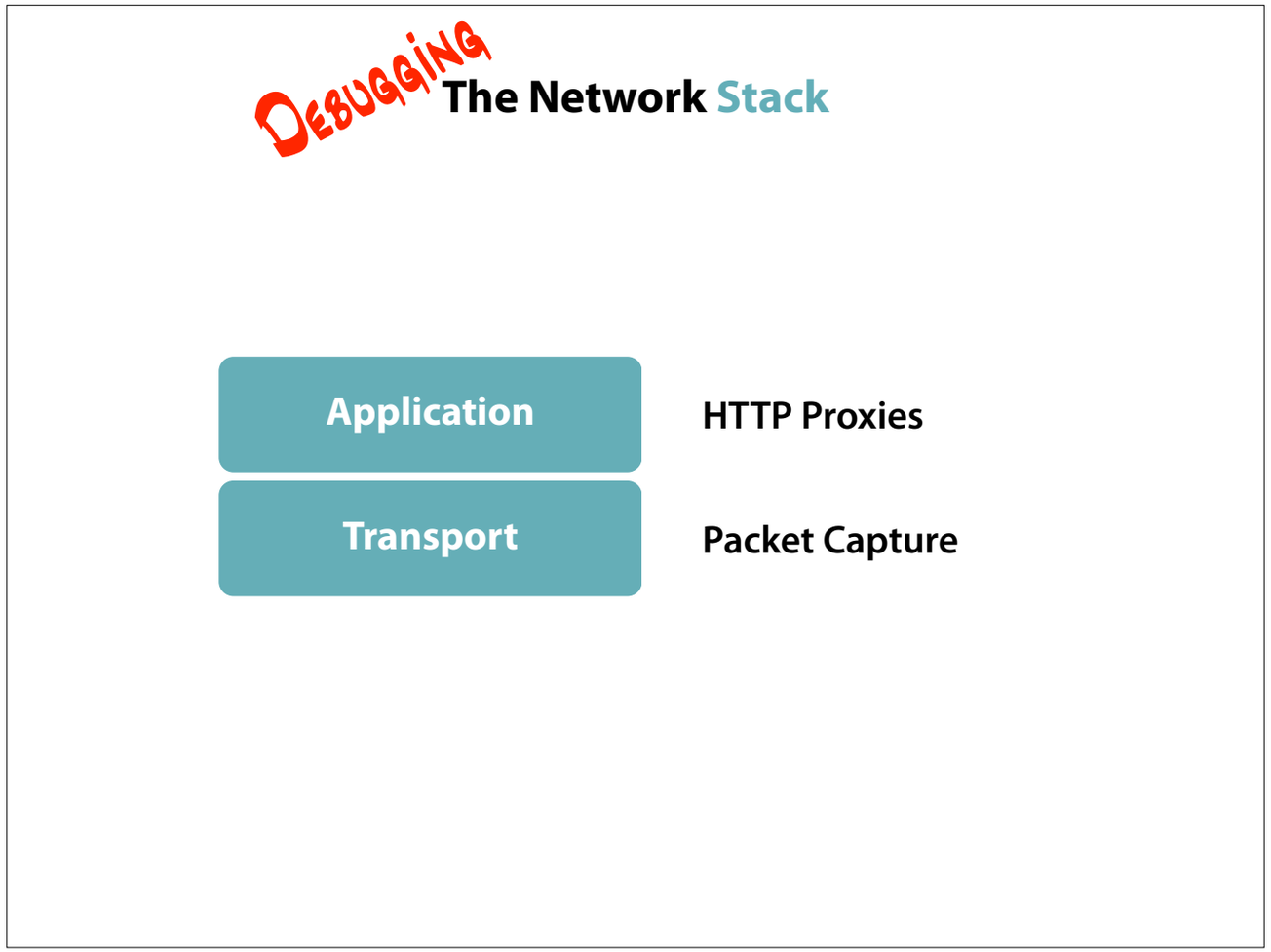
¶

Real-world examples of these layers are HTTP, TCP, IP and Ethernet or Wi-Fi

Debugging The Network Stack



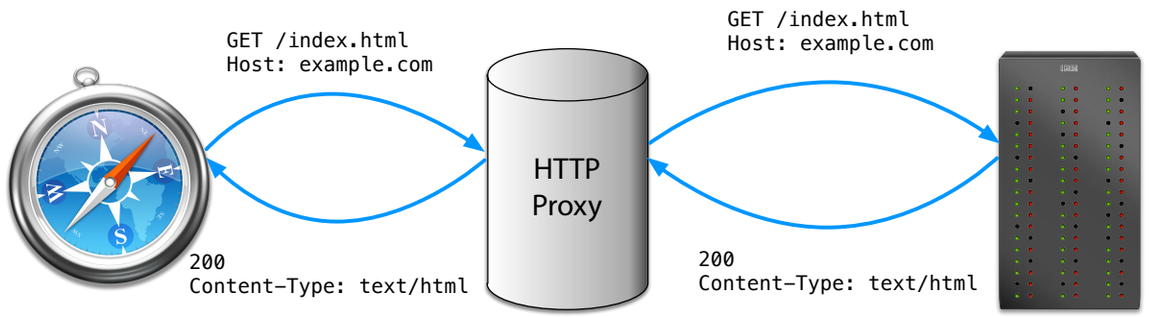
When we're debugging and monitoring the network, we're primarily doing it at the Application and Transport layers.



At the Application Layer, we'll use a technology called "HTTP Proxies".

At the transport layer we'll use a different technology called "Packet Capture"

HTTP Proxies



HTTP Proxies serve as “middlemen” between user-agents and web servers. A user-agent, like a browser, is configured to direct all outgoing web requests to a known proxy, which forwards them on to the final destination.

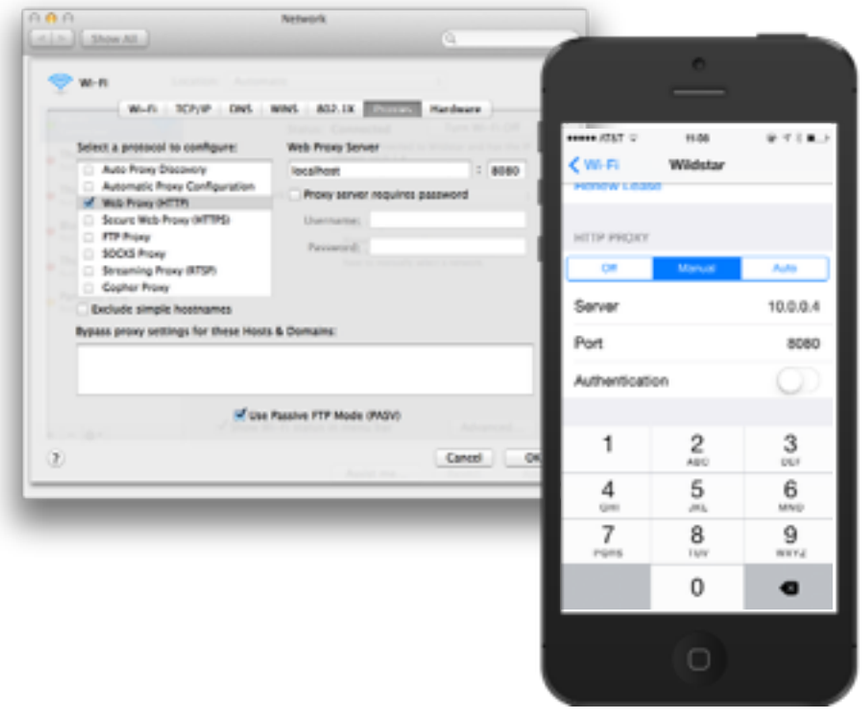
¶

The proxy then forwards the response back to the original requester

¶

This type of arrangement is often referred to as “man-in-the-middle”

HTTP Proxy Setup

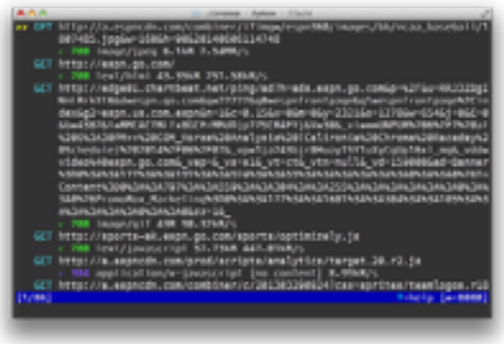


Proxies require some configuration depending on whether you're running your application in the simulator or a real device.

Later in this course we'll look at how to configure HTTP proxies for OS X and iOS.

HTTP Proxy Tools

Charles



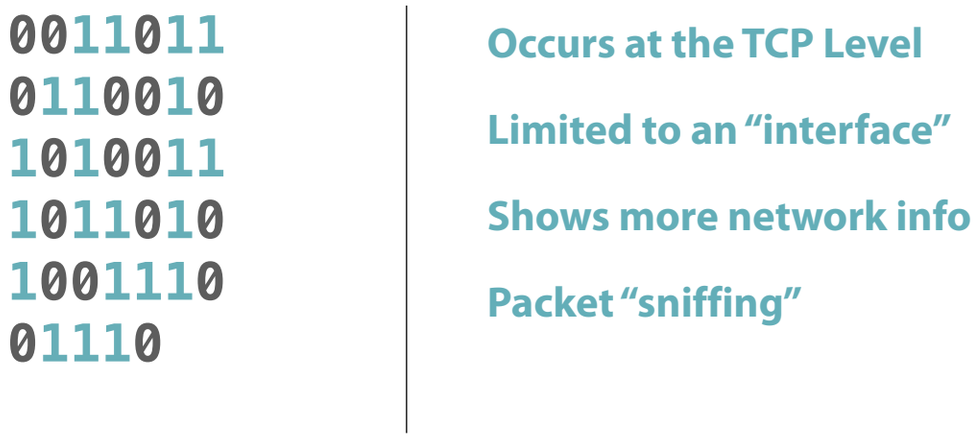
mitmproxy

For HTTP Proxies, we're going to look at two tools: Charles & mitmproxy

Charles is GUI client whereas mitmproxy is a command-line tool. For Windows developers, Charles is similar to Fiddler.

Charles is a commercial product, but mitmproxy is free. We'll look at both and review their similarities and differences.

Packet Capture



Packet capture occurs at the lower, TCP Level

¶

Since we’re at the TCP level, capture has to occur on a specific network interface. These are usually named things like “en0” or “en1” on your Mac.

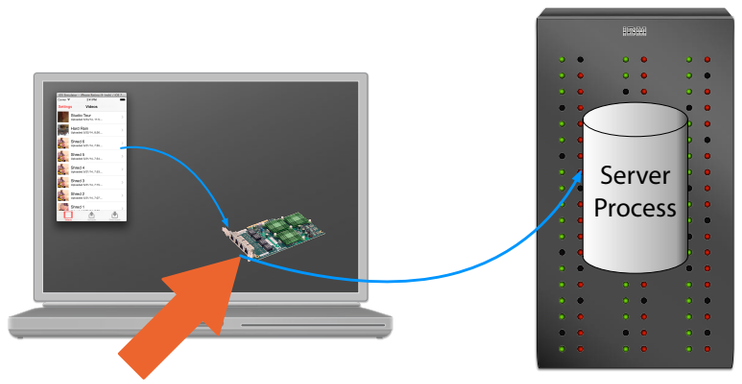
¶

Because we’re at a lower level, packet-capture shows more information such message framing, time-to-live and other TCP and IP data.

¶

Packet capture is sometimes also called “packet sniffing”, depending on how it’s deployed.

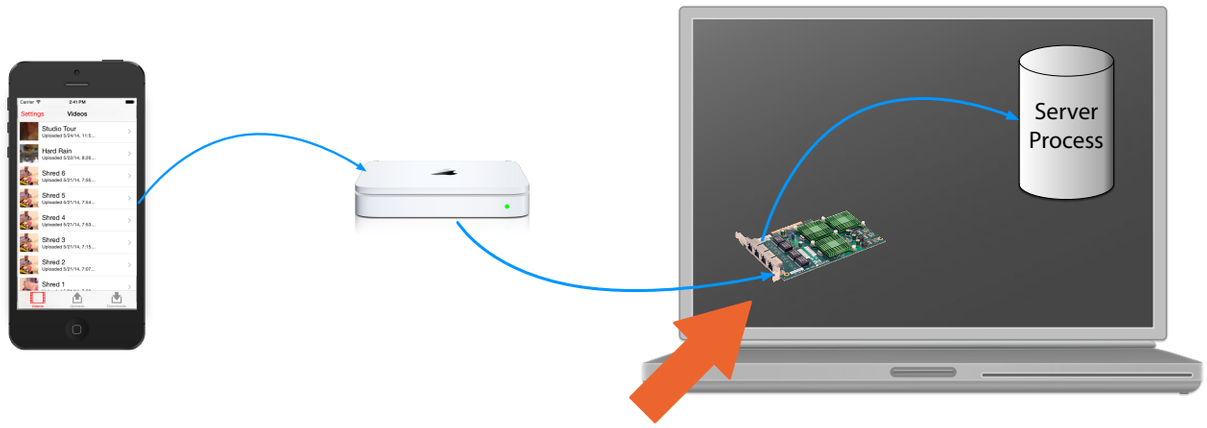
Packet Capture



Since packet capture is limited to a specific interface, you need to be aware of *how* your application is connecting to a server.

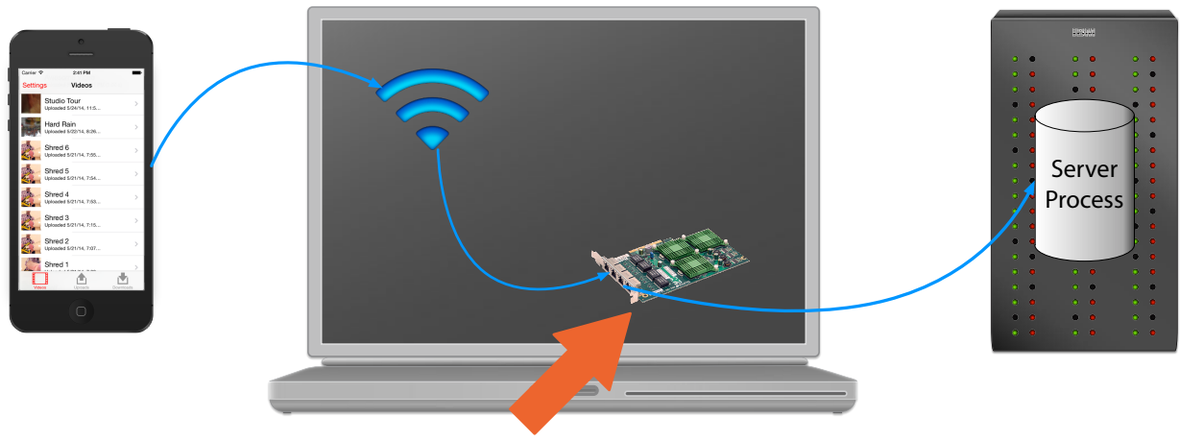
One common case is developing on your local Mac using the Simulator and connecting to a remote server. To capture traffic here, you need to observe traffic on your Wi-Fi or Ethernet device.
¶
In this case, you want to capture packets at the interface that is your outgoing connection: either your Wi-Fi antenna or Ethernet port.
¶

Packet Capture



If you're connecting a device to a server process running on your local machine, you need to capture packets on the incoming interface: either your Wi-Fi antenna or Ethernet port

Packet Capture



Another way to capture packets from the device is to configure your Mac as a Wi-Fi base-station using Internet Connection Sharing.

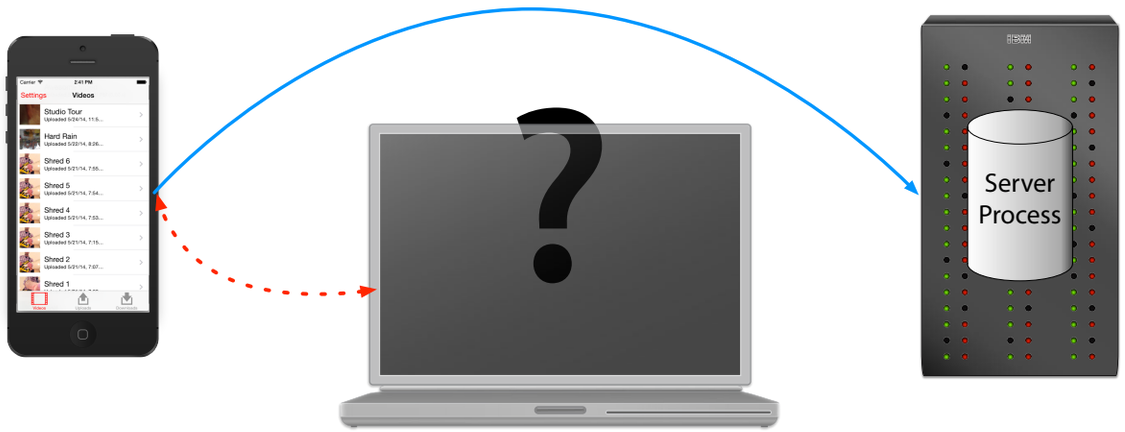
¶

The device connects to your Wi-Fi interface which is routed internally on your Mac to your Ethernet port.

¶

Then you can observe both incoming and outgoing traffic on the Ethernet port to and from the remote server. Note that this only works if you can use your Ethernet connection to get to the server process.

Remote Packet Capture



In some cases you may have an architecture where a real device *must* connect directly to a server. What do you do in that case?

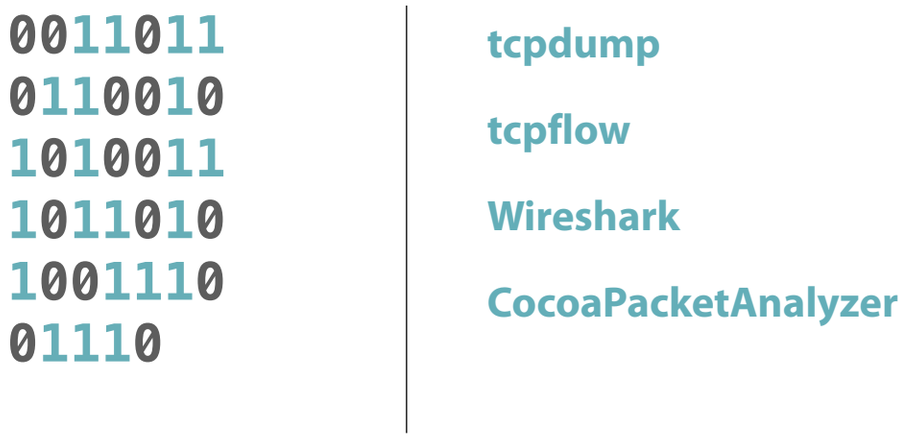
¶

We'll look at a little-know backdoor to capture packets directly off of your device with your Mac.

¶

All you need is a little white cable and few terminal commands.

Packet Capture Tools



When it comes to packet-capture tools, we have several options:

¶

The grand-daddy of all packet-capture tools is “tcpdump”. It can also be the most intimidating to use since it was built for networking engineers.

¶

“tcpflow” is a simpler tool which hides a lot of the low-level packet and framing information we typically don’t care much about as developers.

¶

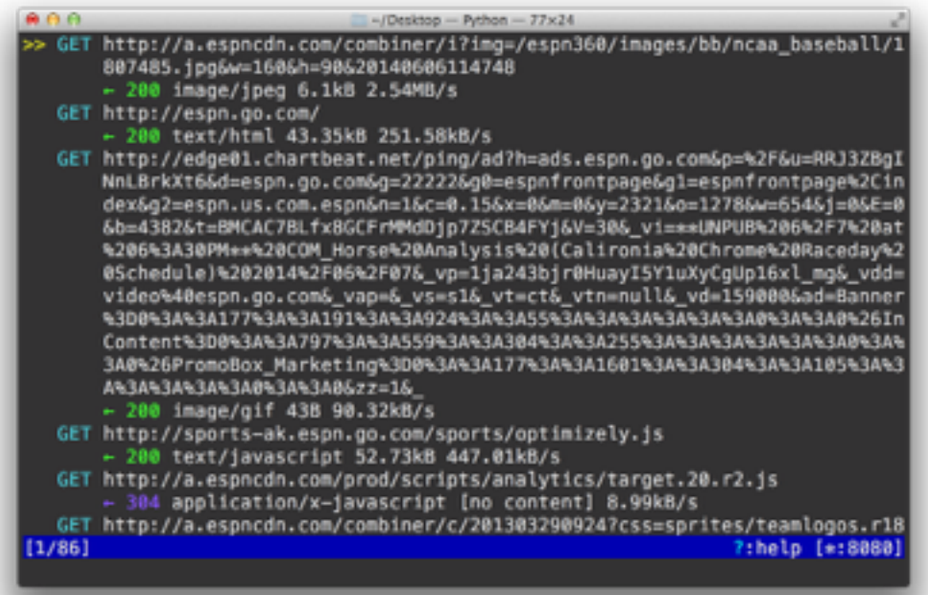
For graphical tools, we have two options. If you come from an open-source background you can run a modified version of Wireshark on your Mac.

¶

However an excellent fully-native alternative is CocoaPacketAnalyzer. All of these tools are free or donation-ware.

Let’s take a look at how both HTTP Proxies and packet-capture tools work in the real world.

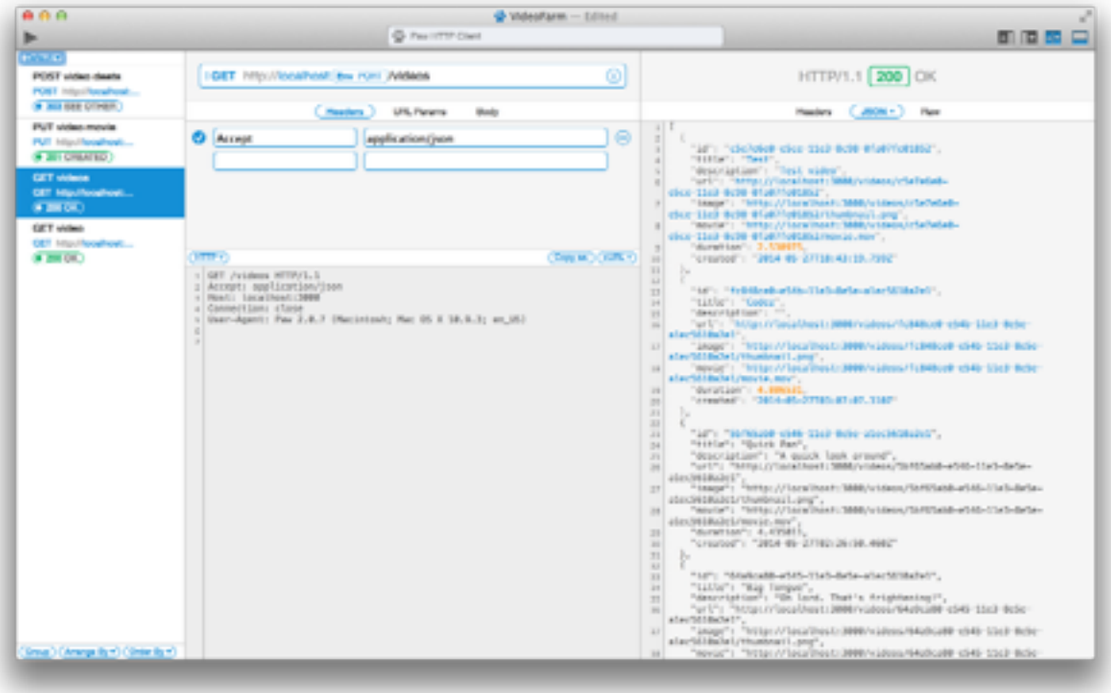
mitmproxy



Like Charles, mitmproxy has the ability to edit previous requests and replay them.

It also has the same limitations of being able only to work with previous requests.

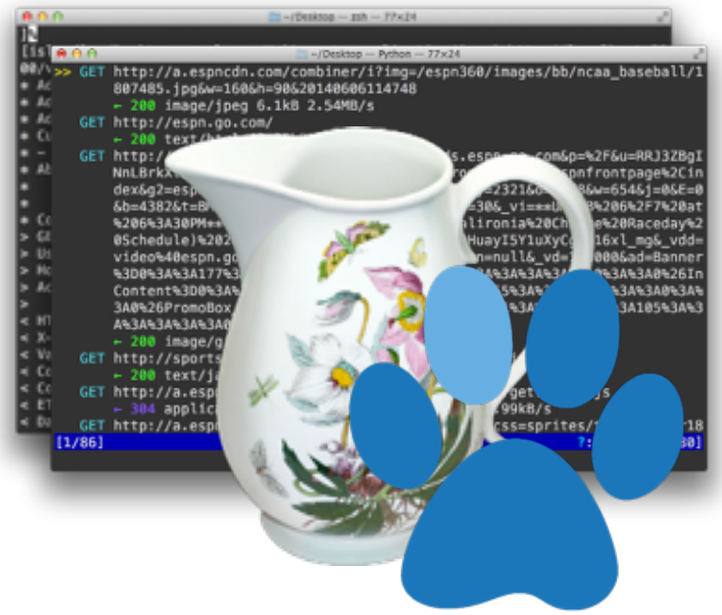
Paw



My favorite tool for this kind of exploration is a native Mac app called Paw.

Paw let's you save entire sessions for replay at a later time. It has a very elegant variable substitution mechanism and provides a great way to visualize your requests and server responses.

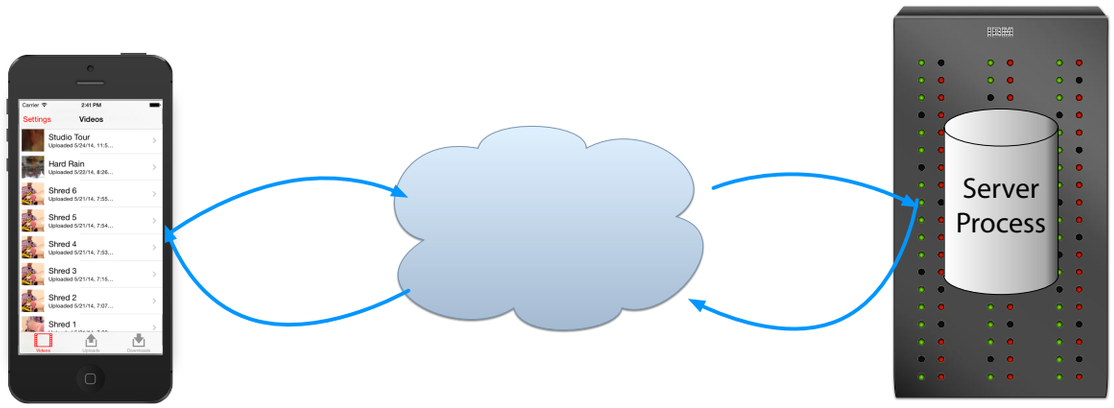
No Xcode



Keep in mind that, regardless of which tool we're using, we're going to connect directly to our server. There won't be any Xcode or iOS applications involved. Our goal here is to interact with a remote server, observe its behavior and then apply those observations to our code.

Let's get starting by looking at curl…

Traffic Shaping

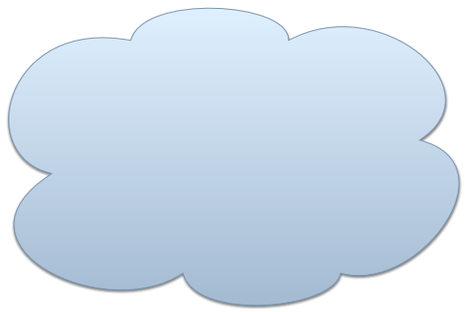


Traffic shaping is the process of modifying the requests and responses between clients and servers.

Traffic shaping is typically done by placing some kind of process between the application and server.

You may have several reasons for why you want to do this…

Traffic Shaping



Test special cases

API Compensation

Bandwidth Simulation

You may want to test special cases like generating server response codes that are difficult to initiate from the server-side

¶

You may also want to developer against a server API that doesn't exist yet. Rather than wait for server development to finish you can "fake it 'til you make it"

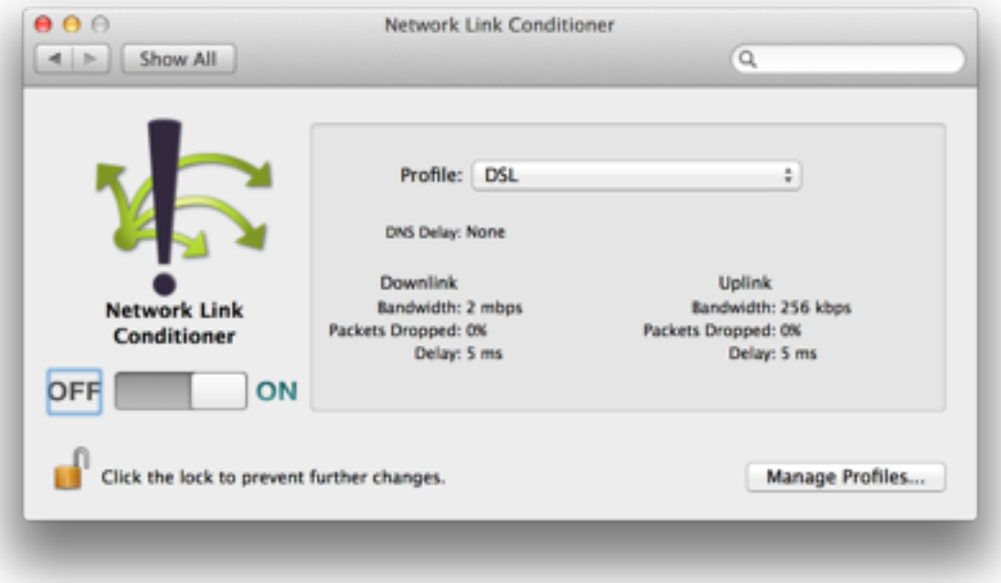
¶

Traffic shaping tools are also a great way to simulate various networking conditions to see how your application responds when bandwidth is very low or non-existent.

Charles

One of the many things that Charles can do is allow you fine-tune the speed of requests and responses. This is done with a tool Charles calls “throttling”

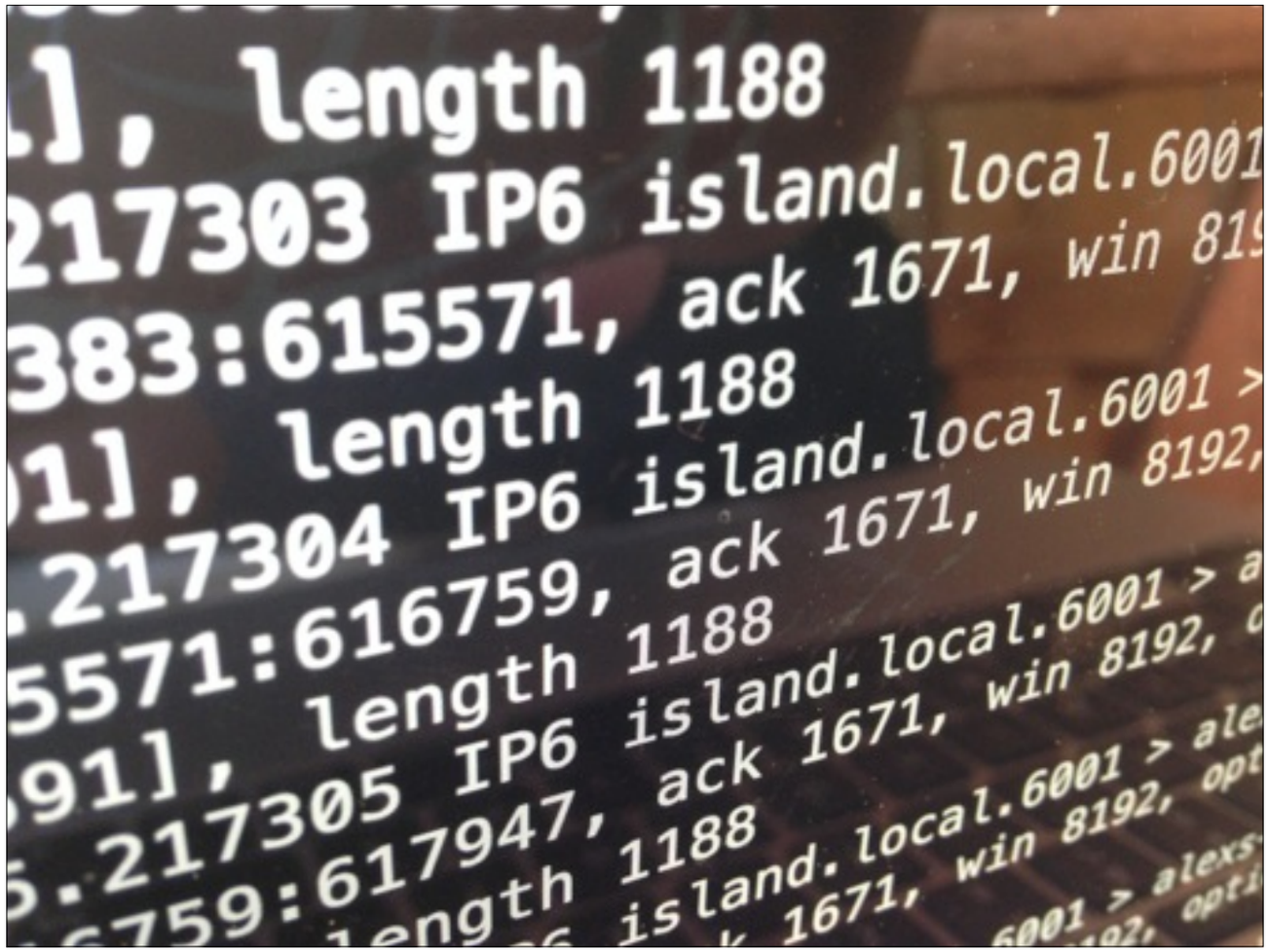
Network Link Conditioner



The Network Link Conditioner tool is a preference pane created by Apple. Unlike Charles, the Network Link Conditioner affects the entire bandwidth of your Mac.

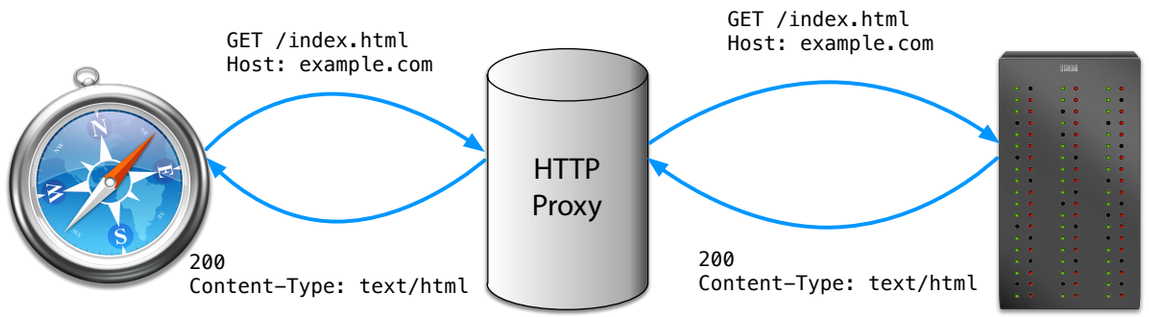
If you're running the Simulator this will automatically affect it's networking performance. If you're testing on a real device, you'll need to route network traffic through your Mac to affect your iOS device's network performance.

Let's take a look…



In this module, we've looked at a variety of tools and techniques to debug your networked applications and improve your productivity.

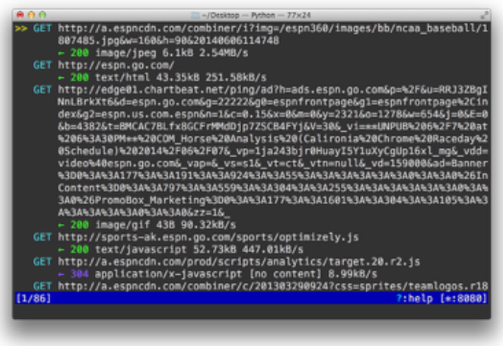
HTTP Proxies



We looked at HTTP Proxies as a way to observe, capture, replay and modify client/server traffic.

HTTP Proxy Tools

Charles



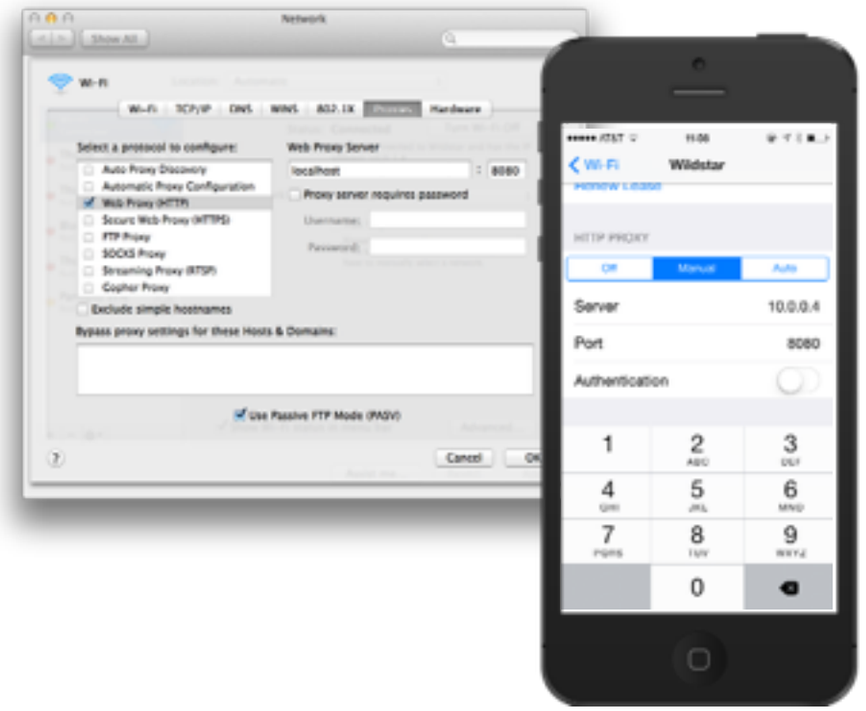
mitmproxy

We looked at two powerful HTTP Proxy Tools: Charles & mitmproxy.

Charles is a commercial GUI application, while mitmproxy is a free command-line app.

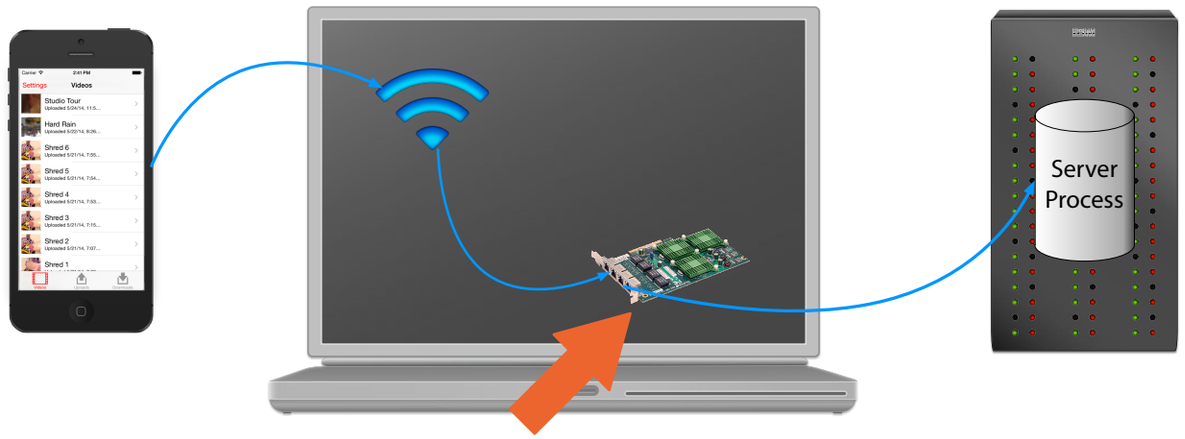
Both tools are capable of much more than we covered here and it would be worth your time to explore both.

HTTP Proxy Setup

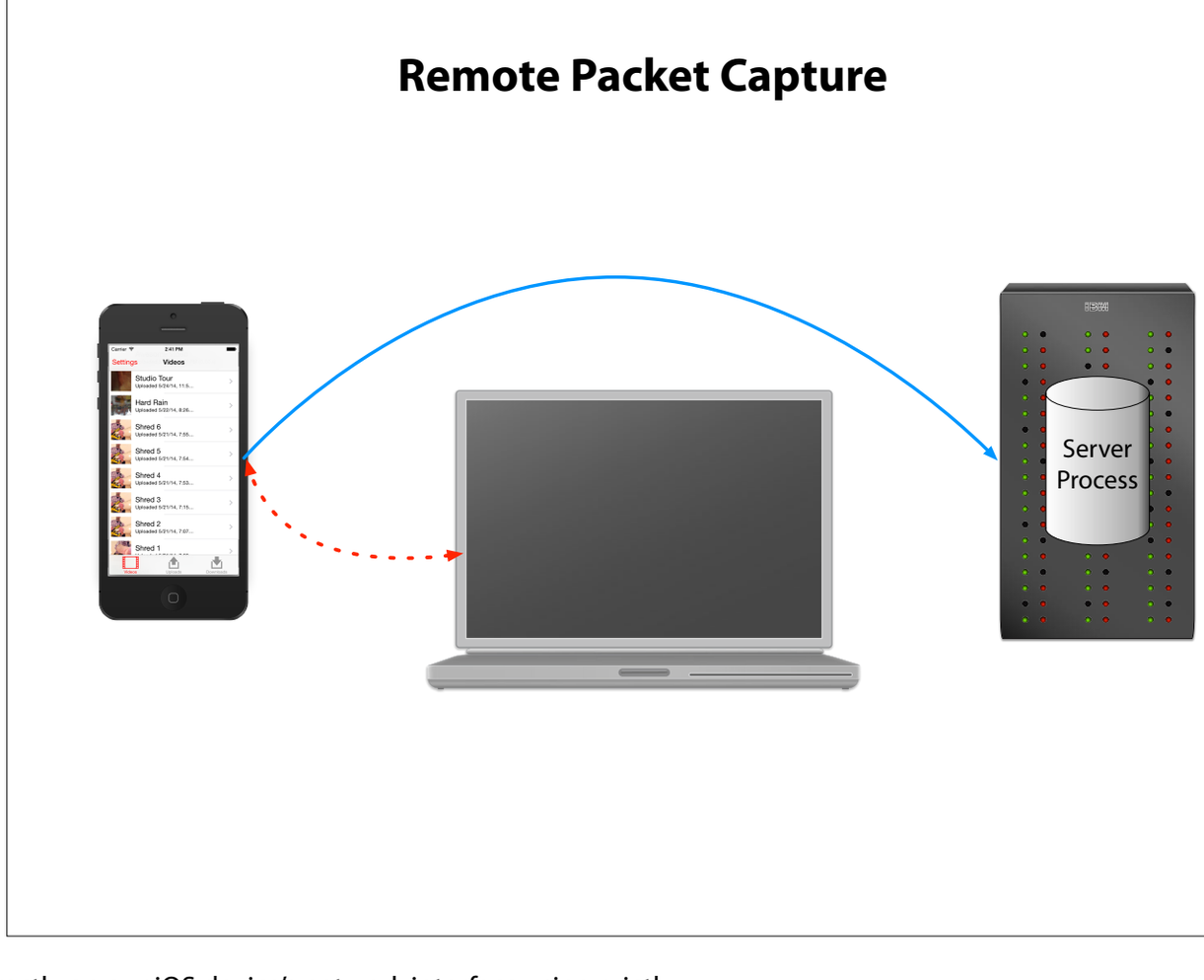


We looked at how to configure our devices for HTTP proxies so that we can observe what's happening between our apps and servers.

Packet Capture



We saw how packet capture is another way to observe traffic. Unlike HTTP proxies, packet-capture works at a lower level in the network stack.



We even saw how to remotely capture traffic directly on our iOS device's network interface using rvictl.

All of these tools give you a powerful view into your application's network traffic and behavior. This should give you a clearer picture of what's going on in your app how you can make it even better for your users.