Configuration of Networked Devices using *iproxy*

Horms (Simon Horman)
horms@verge.net.au

February 2002

http://verge.net.au/linux/iproxy/

Abstract

iproxy is designed to enable networked devices to be configured without prior knowledge of their network setup. This is intended to aid initial setup, where a network appliance is deployed on a customer's network. It is desirable for the customer to be able to configure the appliance without the need for DHCP infrastructure or console access of any kind. *iproxy* allows configuration to be done using existing TCP based administrative interfaces developed for the appliance, such as web-based configuration tools. Thus iproxy can be integrated seamlessly into the existing management framework for a device.

Introduction

It is desirable to configure network devices without the need for console access. In particular there is a need to configure the initial network settings so subsequent configuration using TCP based configuration interfaces, running over protocols such as HTTP, HTTPS, Telnet and SSH can be used.

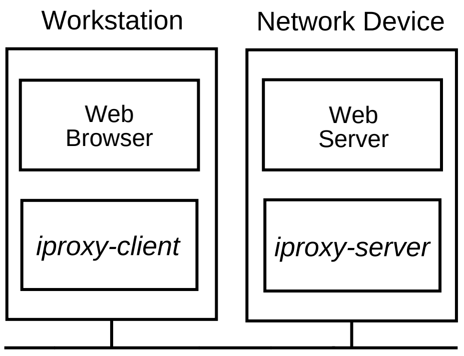
Having the device use DHCP to obtain its IP address is an obvious and desirable solution. However, a solution that does not rely on customer networks having a DHCP infrastructure in place is desirable. *iproxy* makes use of UDP to allow TCP-based administrative services to communicate with the network device, prior to its initial network setup. Thus, existing management interfaces that has been developed for use with the device can be used to configure the initial network settings.

As well as being useful for configuring the network device when it is first added to a customer's network, it is also useful for relocations. *iproxy* can be used to reconfigure the network device when it is relocated, to a different part of the same network, or an entirely different network.

Implementation

Given the goal of carrying TCP-based services over UDP, a number of options are available: PPP could be run over UDP. Alternatively a user-space TCP implantation could tunnell traffic over UDP. However, the implementation of either of these would require privileged access under Unix and implementations of this in user-space on *Microsoft Windows* would be difficult.

For this reason a much simpler approach, was taken. A pair of proxy servers convert TCP connections into UDP packets and vice versa. While this does not have some of the performance characteristics of the other solutions, this is not a problem as *iproxy* is intended to carry small amounts of data over a reliable, fast LAN. The result is a simple client-side implementation that can easily be ported to different operating systems.



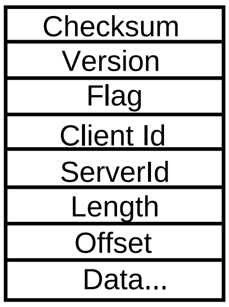
iproxy Architecture

iproxy-client runs on the user's workstation. It listens for incoming TCP connections. This may be from a web browser, SSH or Telnet client, or a client for any TCP protocol. The *iproxy-client* then encapsulates this connection as UDP packets. *iproxy-server* is shipped on the network device to be configured and listens for UDP packets sent by a *iproxy-client*. It then opens a connection to the local daemon, whether it be HTTP, SSH, Telnet or another TCP protocol and relays the encapsulated information from the user's client. Replies from the daemon follow the reverse path.

Broadcast and Multicast UDP

iproxy was originally conceived to run over broadcast UDP. The idea being that this can be used to communicate with hosts whose IP address is unknown as broadcast packets are received by all hosts on the network. However, there is the restriction that the hosts be on the same physical subnet. Multicast does not have this restriction, and enables communication with nodes on different subnets. *iproxy* supports both broadcast and multicast modes of operation.

Encapsulation



iproxy packet format

To enable a TCP connection to be tunnelled in UDP packets some identification of the packet is required. To achieve this iproxy packets have a header followed by up to 1048 bytes of data from the TCP connection. To maximise portability, each field in the header is a 32-bit unsigned integer in network byte order.

Checksum: Rolling 8-bit checksum of the header fields other than the checksum and the data.

Version: iproxy protocol version. Used to identify incompatible versions of the protocol in the future. Currently 1.

Flag: Flags to specify special packets, such as Acknowledgements, Keep Alive and Finish Packets.

Client Id: Unique identifier of the client.

Server Id: Unique identifier of the server.

Length: Total length of the packet including the body and data.

Offset: Offset. Incremented by the length of the data for each successive packet. Used in Lieu of a sequence number.

Server Discovery

Given that the motivation for *iproxy* is to enable configuration of hosts running *iproxy-server* without prior knowledge of their setup, auto discovery is an important feature of *iproxy*. This is implemented by iproxy-client sending a discovery packet with its own client id and a special *discovery* server id. All iproxy-servers that receive this discovery packet should reply using the client id in the packet and their own server id. Thus, an *iproxy-client* can readily find all the *iproxy-server*s on the network.

Connection Tracking

One of the key differences between TCP and UDP is that the former is connection oriented while the latter is not. This means that in order for *iproxy* to carry TCP services over UDP it must keep track of connections. In particular, as *iproxy-server* may be receiving UDP packets from the same *iproxy-client* for different connections at the same time it must be able to differentiate which connection each packet belongs to. To do this information encapsulated in the *iproxy* packet is used to differentiate connections and a table of active connections is kept.

Acknowledgements

For each UDP packet that is received an acknowledgement packet is returned. Until a packet is acknowledged, no additional packets will be sent. Though simple and inefficient, this scheme all but eliminates the possibility of out of order packets. Thus, the protocol does not have any provision for windowing of back-off. The simplicity of this approach is in keeping with the underlying criteria that *iproxy* be as simple as possible and not focus on performance issues. It is intended to be used infrequently to enable the configuration of the network for the network device.

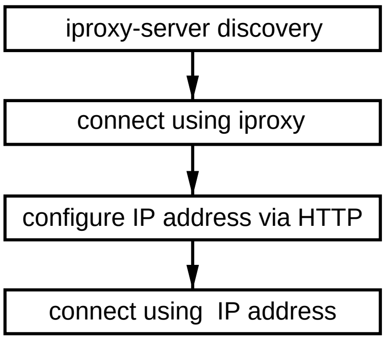
Timeouts

Each time a packet is received by a connection its expire timeout is reset. If the expire timeout passes without the receipt of another packet the connection is closed. There is also a resend timeout. The purpose of this is to resend a packet if this timeout expires before an acknowledgement packet is received.

Keep-Alive

In order to prevent protocols that may have be idle for a time, such as Telnet or SSH, from causing the connection to timeout keep alive packets are sent. This resets the timeouts for the connection.

Deployment



confiiguration using iproxy

To use *iproxy*, a user must install *iproxy-client* on their workstation. The user then uses their web-browser to connect to the local *iproxy-client*. In the reference implementation, server discovery is a manual process, but it is possible for this to be done via a web-interface generated by *iproxy-client*.

Once the end user has selected a networked device to configure using *iproxy*, they connect to the device by connecting to the local *iproxy-client*, which will tunnell their connection over UDP. This should bring up the networked device's HTTP administrative interface. This can be used to configure the IP address of the device. Once this is done, the end user can use their browser to connect directly to the device.

The same process holds for configuring the networked device using administrative interfaces running over SSH, HTTP, HTTPS or any other TCP protocol.

Conclusion

iproxy is able to carry TCP services over Broadcast and Multicast UDP enabling existing TCP-based administrative interfaces to access machines before they have been configured for a customer's local network. The client that needs to be installed on an end-user's machine is very simple, allowing easy porting between different operating systems.

iproxy has been prototyped using *Linux*. Implementations on other platforms, including *Java* are underway. More information is available from:

`http://verge.net.au/linux/iproxy/`