udptoserial

A tool for sending UDP packets over a serial line

# Introduction

For one of our industrial IoT projects, we needed to allow some client/server programs that normally communicate with UDP packets to communicate over RS-232 serial lines instead. This program, udptoserial, is a UDP server that converts UDP to serial and back.

If you are old, you know that the problem of sending packets over serial lines is a solved problem. Back in the olden days, we used protocols such as SLIP, CSLIP and PPP to set network data between machines. If you have machines and operating systems still support such arcana, you should use them.

This program was written because of two special conditions:

1. We wanted a solution that only specific ports’ UDP traffic down the serial link, leaving the rest of the networking untouched.
2. We wanted a program that could help throttle the traffic on the serial link to various baud rates, for testing.

Thus, this program was born.

# Tutorial

this is the tutorial

## Networking with one-sided conversion and no ephemeral ports

Imagine that two machines. One machine, called **foo**, is some random box with no native IP-over-serial capability. Another machine an old GNU/Linux distro that natively handles ethernet over serial called **bar.**

Given this setup, **foo** wants to send packets to a UDP echo server (RFC 862) on **bar**. The echo server on **bar** listens on port 7, and echoes back the UDP packets it receives.

With standard networking, it goes like this.

* an echo client on machine **foo** spawns an ephemeral port foo:XXXX -- where XXXX represents a random ephemeral port typically between 49152 to 65535 -- and sends UDP packets to bar:7. The packets have source address foo:XXXX and destination address bar:7
* the echo server on machine **bar** receives the packets
* the echo server on machine **bar** responds with packets. The packets have source address bar:7 and destination address foo:XXXX
* the echo client on machine **foo** receives those packets from **bar**

Now replace standard networking with IP-over-serial. **bar** is set up to natively work with IP-over-serial. To get the old GNU/Linux to distro to handle networking over a serial line might involve commands like the following. Your mileage may vary.

sudo slattach -s 19200 -p slip -dL /dev/ttyUSB0 # USB-serial  
sudo ifconfig sl0 10.0.0.1/24 up  
sudo route add default gw 10.0.0.254 sl0

The machine **foo** is set up with a copy of udptoserial listening on port 7.

Now, it goes something like this

* machine **foo** spawns a udptoserial client to listen on port 7
* the echo client on **foo** sends a UDP packet from an ephemeral port foo:XXXX to foo:7
* udptoserial receives the packet. It rewrites the destination address from foo:7 to bar:7 and sends it down the serial line.
* **bar** receives the packet using its native capabilities. The packet it receives has source foo:XXXX and destination bar:7
* the echo server on **bar** responds with a packet to **foo** via the serial link. The packet has source bar:7 and destination foo:XXXX
* udptoserial on **foo** receives the packet via the serial line. The packet source port is 7, so udptoserial will handle it. It rewrites the packet to have source foo:7 and destination foo:XXXX
* udptoserial sends the packet to foo:XXXX to the echo client

To set up udptoserial, the udptoserial.ini file should contain

port1 = 7

destination\_ip = <IP address of other machine>

ephemeral = false

mode = one-sided

## Networking with two-sided conversion and and ephemeral ports

Now consider the same scenario as the previous, but now, neither machine **foo** nor machine  **bar** will be using native IP-over-serial capabilities

Now, it goes something like this

* machine **foo** spawns a udptoserial client to listen on port 7
* the echo client on **foo** sends a UDP packet from an ephemeral port foo:XXXX to foo:7
* udptoserial on **foo** receives the packet. It rewrites the destination address from foo:7 to bar:7 and sends it down the serial line.
* udptoserial on **bar** receives the packet via the serial link. The packet it receives has source foo:XXXX and destination bar:7. udptoserial checks the source port XXXX, which is not a port it is managing. It spawns a new ephemeral port bar:YYYY. udptoserial’s internal translation table will now convert foo:XXXX to bar:YYYY. It sends out a packet with the source address bar:YYYY and the destination address bar:7.
* the echo server on **bar** receives the packet responds with a packet. The packet has source bar:7 and destination bar:YYYY
* udptoserial on **bar** receives the packet. The source address bar:YYYY is in its internal translation table. It rewrites the packet with source bar:7 and destination address foo:XXXX and sends it down the serial line.
* udptoserial on **foo** receives the packet via the serial line. The packet source port is bar:7, so udptoserial will handle it. It rewrites the packet to have source foo:7 and destination foo:XXXX
* udptoserial sends the packet to foo:XXXX to the echo client

## TFTP is the worst protocol in the world

TFTP is a complicated UDP protocol to handle because both the client \*and\* the server spawn ephemeral UDP ports. udptoserial needs to spawn ephemeral ports both coming and going and translate between the client/server ephemeral ports to the udptoserial ephemeral ports. I’ll try to explain it below, but, it gets murky fast.

Imagine there are two machines: **foo** and **bar**. **foo** uses a TFTP client to fetch files from **bar** which has a TFTPD server. TFTP, or Trivial File Control Protocol, is an old protocol that is still used in the booting or startup of some servers and some IoT platforms. TFTP uses UDP for transporting packets.

Normally, fetching a file via TFTP goes like this

* **foo** sends a TFTP read request to **bar** to **bar**’s UDP port 69
* **bar** responds with an acknowledgement from a newly chosen ephemeral port
* **bar** sends packets containing file information back to **foo**
* **foo** acknowledges to receipt of each packet

Each UDP packet has a source IP address, a source port, a destination IP address, and a destination port, so let’s look at how that plays out in the above scenario.

* **foo** constructs a socket, and connects to **bar** port 69 on that socket. The local port of that socket on **foo** is likely randomly chosen from one of the ephemeral ports. Let’s imagine that the ephemeral port is 50000. The packet is from foo:50000 to bar:69.
* **bar** creates a new ephemeral port to handle the response. Let’s imaging that the ephemeral port that was randomly chosen was 51000. **bar** sends an acknowledgement packet. The packet is from bar:51000 to foo:50000.
* **bar** sends packets with file information from bar:51000 to foo:50000
* **foo** responds with packets from foo:50000 to bar:51000

With udptoserial in the mix, several things happen

* thetftp client on **foo** constructs a socket, and connects to **foo** port 69 which is being monitored by udptoserial. The local source port of that socket on **foo** is likely randomly chosen from one of the ephemeral ports. Let’s imagine that the ephemeral port is 50000. The packet is sent from foo:50000 by tftp and received at foo:69 by udptoserial.
* udptoserial on **foo** modifies the packet so that the source is foo:69 and the destination is bar:50000. It sends that packet down the serial link to the other machine.
* udptoserial on **bar** receives that packet from the serial port.
* udptoserial on **bar** will need to create an ephemeral UDP port to handle this new connection. Let’s say the port that gets created is bar:51000 from which to send the packet. udptoserial on bar will keep track that when it receives packets to port bar:50000, it will send them from the new ephemeral port bar:51000. Then It modifies the packet so that the source is bar:51000 and the dest is bar:69 and sends it.
* ftpd on **bar** is listening on bar:69 and receives the packet. ftpd on **bar** creates a new ephemeral port to handle the response. Let’s imaging that the ephemeral port that was randomly chosen was 52000. ftpd on **bar** sends an acknowledgement packet. The packet is from bar:52000 to bar:51000. From the point of view of ftpd on **bar** its client is at bar:51000.
* udptoserial on **bar** is listening on bar:51000, and receives the packet from ftpd. It looks up in its map and finds that local 51000 maps to remote 50000.
* udptoserial on bar modifies the packet so that the source is bar:50000 and the destination is foo:52000
* udptoserial on **foo** will likely need to create a new UDP port to handle traffic from the new port 52000. So, it creates a new port foo:53000 and adds to its translation list that ports 52000 from serial is 53000 here. Then it modifies the packet so that the source is foo:53000 and the destination is foo:50000 and sends it.
* the ftp client on **foo** receives its packet from foo:53000 on its port foo:50000. From the point of view of the ftp client on foo, its server it at foo on port 53000.
* And from there, the rest of it should “just work”.

## Network Address Translation Modes

# Reference

this is reference material