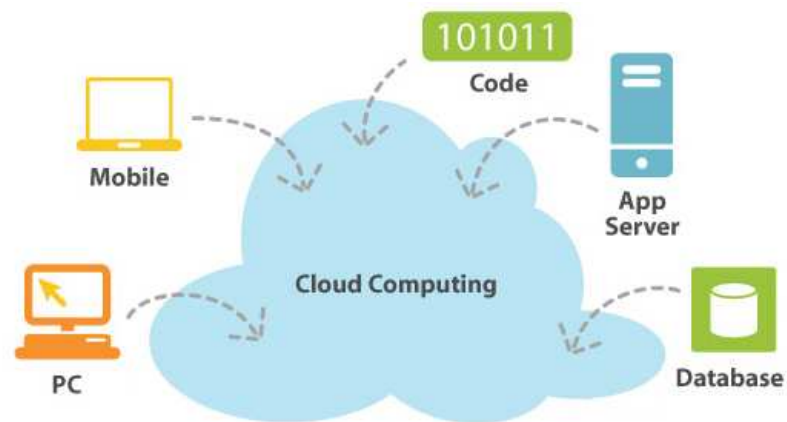




# Great Cows, Pythons and clouds

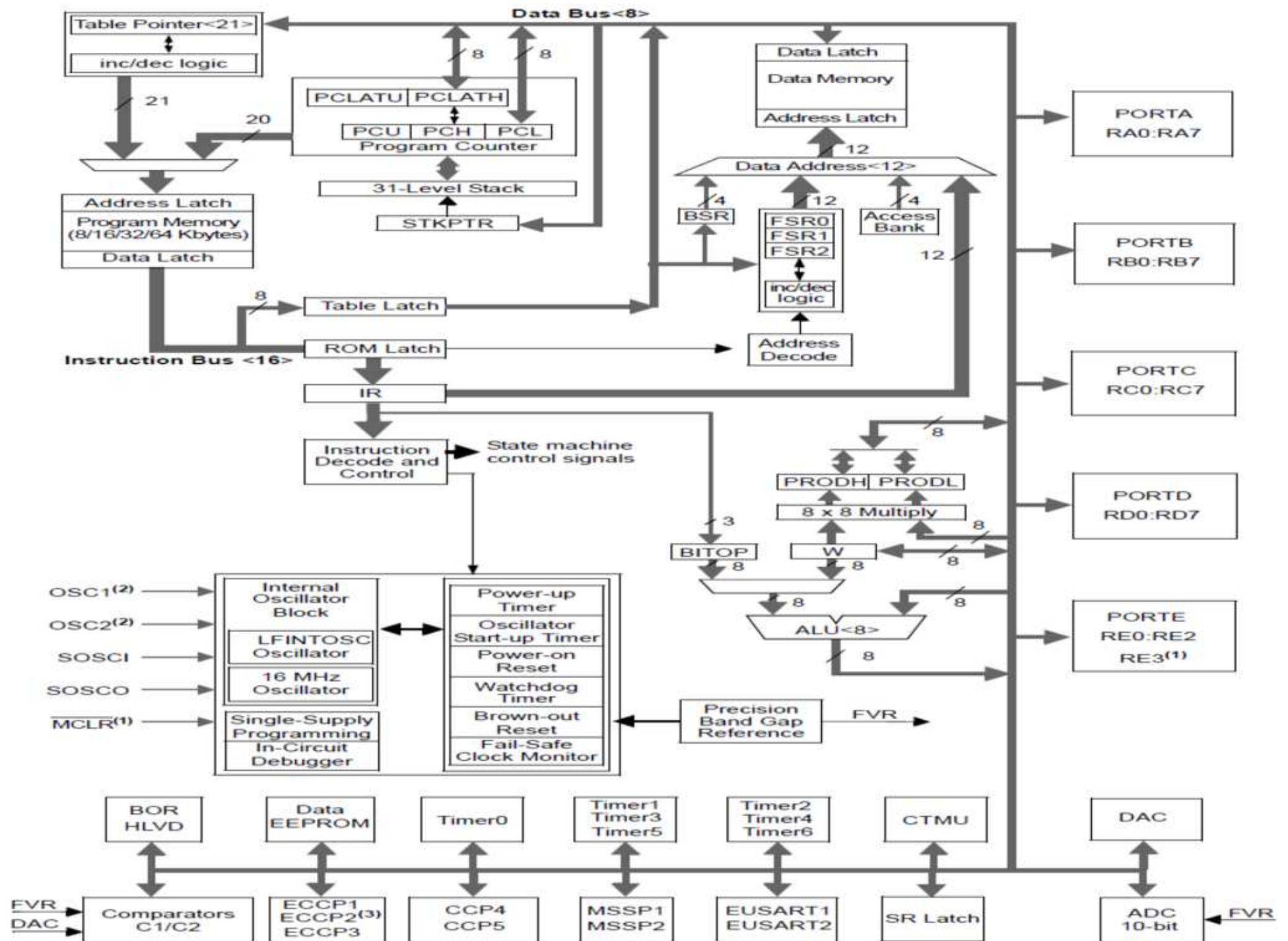


# A few years ago

- Fell out with my fellow directors and shareholders in a previous business
- Decided to start my own business from scratch (never done that before)
- Thought I would try my hand at developing new some new products (somewhat dictated by restrictions of trade with my previous business)
- Have not seriously done any real engineering (h/w and s/w design) for many years. Its just like riding a bike isn't it? How hard can it be.
- Came up with this idea to develop a device for early warning for flash flood events
- Very little funds so decided to do most of the work myself. It was fun!

# A new product is conceived

- Needed low cost, low power solution so chose Microchip PIC microprocessor as the core of the design
  - Highly integrated (great range of integrated peripherals)
  - Low power
  - Lots of industry support
  - Wide choice of cost effective development tools/platforms (many in the public domain space)



# But what software language?

Being a very lazy programmer, I looked for a product that would be easy and quick to get things going. I hate “C”

- I stumbled across Great Cow Graphical Basic. This looked like an easy way to get back into programming again.
  - Has a novel flow programming GUI + traditional text interface
  - Extensive library support for PIC/AVR micro families
  - Locally developed (Adelaide Uni student)
  - Very affordable (\$0)
  - Enhanced Basic programming language
  - Relatively easy to read and learn
  - Produces efficient code (size & speed)
  - But, initially was still a bit buggy!!

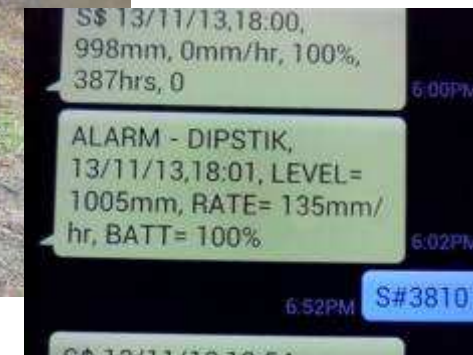
# Great Cow Graphical Basic

The screenshot displays the 'Great Cow Graphical BASIC' software interface. The main window title is 'Great Cow Graphical BASIC - [Untitled0]'. The menu bar includes 'File', 'Edit', 'View', 'Program', 'Tools', and 'Help'. On the left, there is a panel for 'Icons' with a 'Category' dropdown set to 'Input/Output' and a list of available icons: 'Dir', 'Set', and 'Pulseout'. At the bottom of this panel is an 'Add Command Manually' section with a 'Command' input field and an 'Add' button. The central workspace shows a vertical flowchart starting with a green cow icon labeled 'Main'. This is followed by a blue rounded rectangle labeled 'Direction PORTB.0 Out', a red semi-circle labeled 'Do Forever', a blue rounded rectangle labeled 'Pulse Out PORTB.0 1 s', a red octagon labeled 'Wait 1 s', and finally a red semi-circle labeled 'Loop'. On the right side, there is a 'Subroutine/Function' panel with a list containing 'Main' and buttons for 'Add', 'Edit', and 'Delete'. Below that is the 'Icon Settings' panel, which includes fields for 'IO Pin' (set to 'PORTB.0'), 'Time Length' (set to '1'), and 'Time units' (set to 's').

I was able to get the first version developed, sold and into the field within a matter of months!



# My DipStik flood monitoring solution





# New features required

- I now needed to expand the basic product to include new features:
  - Image capture and posting to the cloud
  - Remote firmware updates
- Problem - DipStik has very minimal resources:
  - 64kbyte Flash PROM for program code
  - 3.5k bytes RAM
  - 31 level stack
  - 4 Mhz clock (to keep power consumption down)
  - Only RS232 connection to simple 3G modem (but has basic level socket support via AT commands)

# Cloud Solution?

- As the market requirements were growing more complex, I needed more computing power.
  - Not desirable to scrap existing DipStik design
  - Chose to use cloud server to add functionality
  - Signed up with Digital Ocean (\$5/month)
  - Set up minimal Ubuntu server with its own IP address and domain name ([www.dipstik.info](http://www.dipstik.info))
  - Decided to design my own unique comms protocol to fit with minimal resources on DipStik units
  - Found some simple serial cameras that could be easily bolted onto the existing DipStik design
  - Looked at outsourcing server development but ultimately chose to do the work myself with some advice from MLUG members

# Python sockets

- Never coded in Python
- Never used sockets
- How hard can it be?

```
# Echo server program
import socket

HOST = '206.128.37.3' # Host IP address
PORT = 50007 # Arbitrary non-privileged port

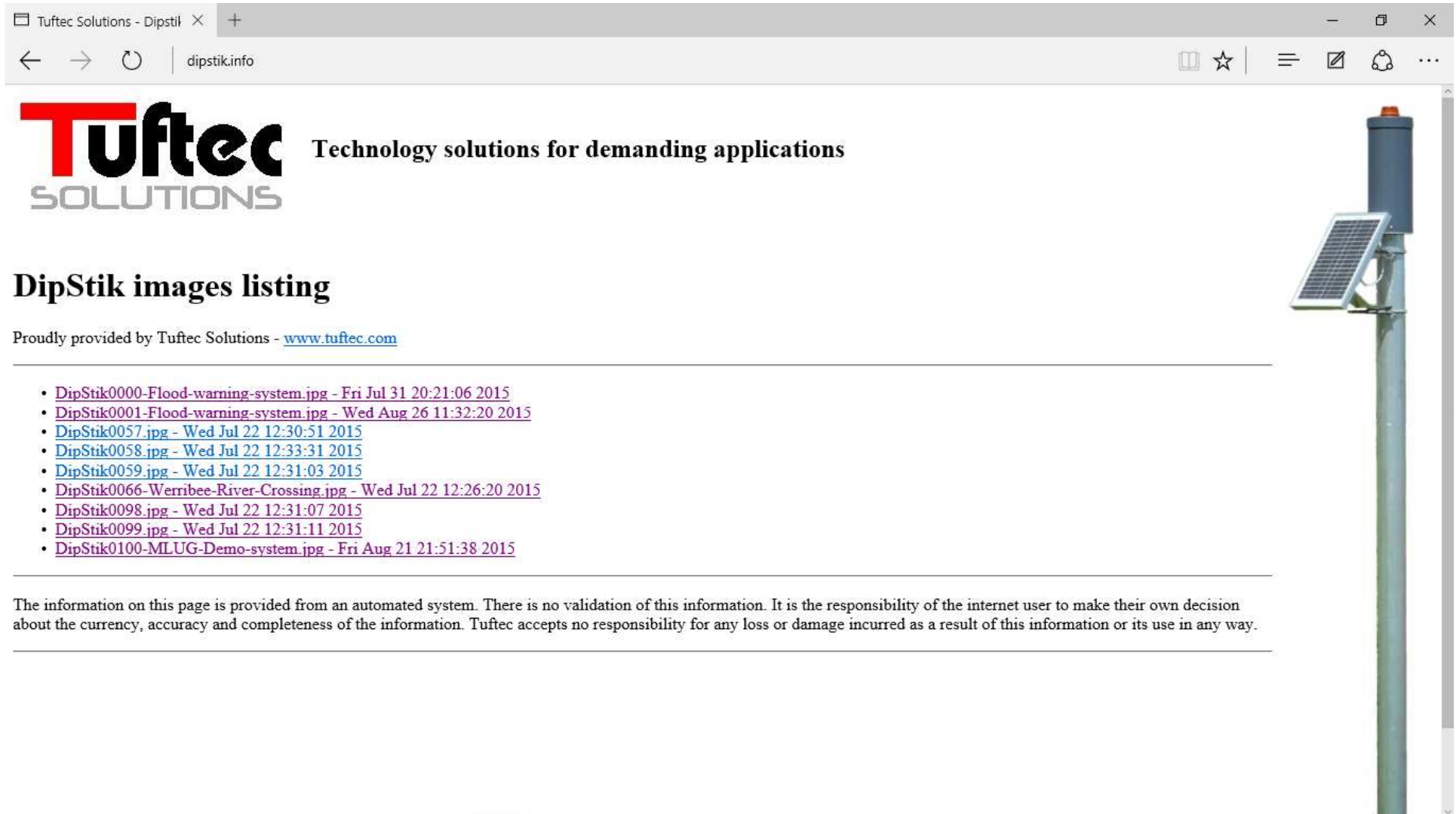
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()
```

Looks easy doesn't it!

# TCP/IP

- Used TCP/IP sockets as they provide reliable end to end comms with little application overhead
- TCP/IP sockets supported on my 3G modems
- Once socket connection established, modem can run in simple transparent mode (just looks like direct serial connection)
- Worked really well for capturing image data and just transferring as one long data stream direct from the camera (no space in DipStik to store image data)

# Web server public interface



Tuftec Solutions - Dipstil x +

dipstik.info

**Tuftec**  
SOLUTIONS

Technology solutions for demanding applications

## DipStik images listing

Proudly provided by Tuftec Solutions - [www.tuftec.com](http://www.tuftec.com)

- [DipStik0000-Flood-warning-system.jpg](#) - Fri Jul 31 20:21:06 2015
- [DipStik0001-Flood-warning-system.jpg](#) - Wed Aug 26 11:32:20 2015
- [DipStik0057.jpg](#) - Wed Jul 22 12:30:51 2015
- [DipStik0058.jpg](#) - Wed Jul 22 12:33:31 2015
- [DipStik0059.jpg](#) - Wed Jul 22 12:31:03 2015
- [DipStik0066-Werribee-River-Crossing.jpg](#) - Wed Jul 22 12:26:20 2015
- [DipStik0098.jpg](#) - Wed Jul 22 12:31:07 2015
- [DipStik0099.jpg](#) - Wed Jul 22 12:31:11 2015
- [DipStik0100-MLUG-Demo-system.jpg](#) - Fri Aug 21 21:51:38 2015

The information on this page is provided from an automated system. There is no validation of this information. It is the responsibility of the internet user to make their own decision about the currency, accuracy and completeness of the information. Tuftec accepts no responsibility for any loss or damage incurred as a result of this information or its use in any way.

# Captured image with overlaid flood data

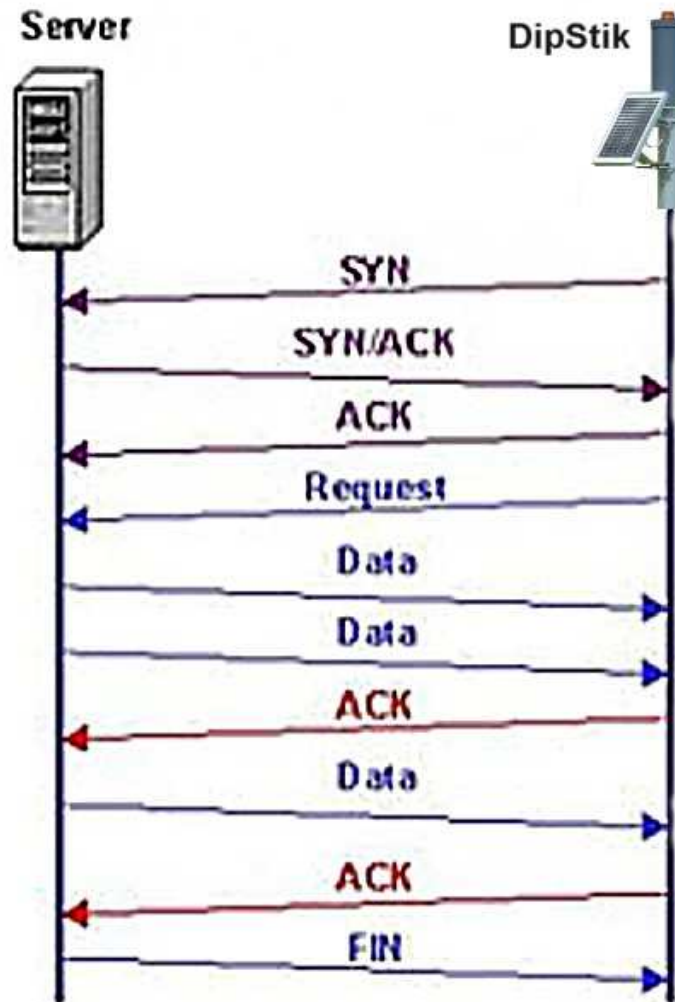




# Remote firmware download challenges

- Internet communications has horrible latency issues
- DipStik needed to securely/reliably write to its own Flash program memory. A slowish process
- DipStik must always be able to recover without any user intervention
- Need to split existing DipStik code into BootLoader and Application areas
- BootLoader must be able to work reliably over the mobile phone network

# Cloud comms in the real world



- Do not underestimate network latency
  - Server in Singapore
  - 3G connected Dipstik in Australia
  - Up to 600ms end to end packet response
  - For a small 88k byte file with 43 byte records, this would take 20 + mins
  - Ouch!!!!
  - Not practical for solar/battery device!

# A simple comms solution

- Remove all packet handshaking to get rid of latency issue
- DipStik is fairly dumb and needs some time between packets to process the data. (During Flash ROM programming the CPU actually stops working!!!)
- The Solution:
  - Add a fixed delay between packets to enable DipStik to do its stuff
  - Easiest way to add a fixed delay is to just add redundant stuffing bytes into the data stream.
  - 20 stuffing characters (null padding bytes) equates to a delay of approx 20ms at 9600 bps.
  - Data packets are simple ASCII coded (Intel HEX) so easy to add redundant stuffing bytes.



# Final solution

- Linux server code spread across 3 programs to handle:
  - DipStik image capture
  - Custom web server to enable clients to view images
  - Firmware download server to automatically handle code updates to remote DipStik units.
  - All application code uses multithreading to enable simultaneous device and user connections
- Firmware updates can now be achieved in a matter of minutes
- As the installed device network grows, the cloud server solution can easily be upgraded to maintain performance

Peter Stone



[www.tuftec.com](http://www.tuftec.com)

[peters@tuftec.com](mailto:peters@tuftec.com)