

APPLICATION NOTE 193

Extending 1-Wire® Range with Network Proxies

Mar 28, 2002

Abstract: This document will cover a methodology for extending the 1-Wire range to a limit of near infinity using proxies. The best analogy to describe the basic proxy is to imagine the proxy server in most companies (sometimes referred to as the company firewall). In this setup, only one computer (the proxy) has an actual connection to the Internet and the rest of the user machines only have access to the proxy. All network packets on the LAN that are intended for computers outside the LAN are repeated on the WAN and vice-versa. This idea is transferable to 1-Wire networks. Connecting to a 1-Wire network through a proxy requires a couple of software modules: the client module and the host module. The host module is the code that runs on the 1-Wire Proxy Server (a PC, MxTNI™, or microcontroller) that has hardware access to the 1-Wire network. The client module is the code that runs on the distant machines that have only network access to reach the host module. It is possible that the host module, rather than having hardware access to the 1-Wire network, could have access to a client module, which would refer to yet another host module. In this way it is possible to chain together the proxy for reaching the 1-Wire network.

Introduction

The original design of the 1-Wire protocol arose from the desire to communicate with nearby devices on a short connection. Although through careful considerations, it is possible to extend the length of this connection further than originally anticipated, the limits are still very real. For more information about the limitations, worst-case considerations, and possible solutions, see application note 148, "[Guidelines for Reliable Long Line 1-Wire® Networks](#)" and application note 159, "[Software Methods to Achieve Robust 1-Wire® Communication in iButton® Applications](#)."

This document will cover a methodology for extending the 1-Wire range to a limit of near infinity using proxies. The best analogy to describe the basic proxy is to imagine the proxy server in most companies (sometimes referred to as the company firewall). Generally, a company provides their employees with a LAN (Local Area Network), for connecting to each other's computers and sharing files. It is usually necessary for the company to provide a means of accessing the Internet (a very Wide Area Network). In this setup, only one computer (the proxy) has an actual connection to the Internet and the rest of the machines only have access to the proxy. The proxy makes all requests for Internet data on the behalf of all computers on the LAN. In this example, the proxy acts as an interface between the WAN and the LAN. All network packets on the LAN that are intended for computers outside the LAN are "repeated" on the WAN and vice-versa.

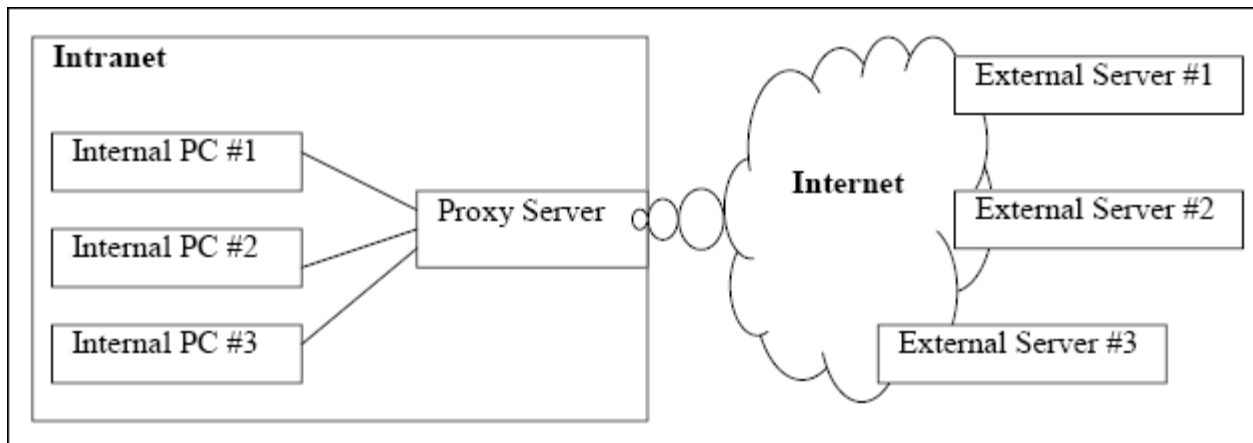


Figure 1. Proxy server example.

Figure 1 illustrates the fundamental service provided by the proxy server and that is to give access to a network that is not normally accessible by the internal computers. More specifically, the internal computers, which can reach the proxy via the LAN, have no other means of reaching the WAN without the help of the proxy. By routing information requests through the proxy server, the previously unreachable external servers can now be reached. Although a lot of the specifics change, this same fundamental idea can

be applied to 1-Wire Networks. Figure 2 illustrates the basics of this application.

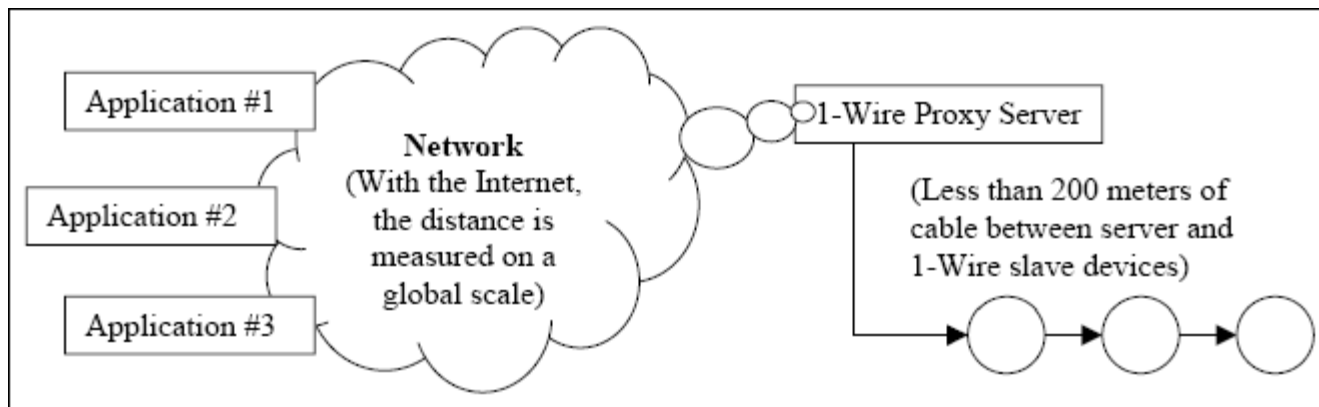


Figure 2. 1-Wire proxy example.

The applications in Figure 2 represent any applications that want access to the 1-Wire Network, but (for whatever reason) cannot access it directly. The reasons could be that they are too far from the network or they are operating in an environment that does not allow for hardware access (i.e., a Java™ application running on a Java-powered cell phone). Due to electrical limitations, an application using a 1-Wire network would typically be running on hardware that is located less than 200 meters from the 1-Wire slave devices. In the scenario depicted in Figure 2, all an application needs is access to the network medium and a suitable 1-Wire Proxy Server. The network medium could be anything such as a RF wireless network, an IR point-to-point link, or a TCP/IP Ethernet network. Thanks to the immensely large Wide Area Network known as the Internet, the distance between the external computers and the actual 1-Wire Network is not an issue anymore.

Overview

Connecting to a 1-Wire network through a proxy requires a couple of modules: the client module and the host module. The host module is the code that runs on the 1-Wire Proxy Server (a PC, MxTNI, or microcontroller) that has hardware access to the 1-Wire Network. The client module is the code that runs on the distant machines that have only network access to reach the host module. It is possible that the host module, rather than having hardware access to the 1-Wire Network, could have access to a client module, which would refer to yet another host module. In this way it is possible to chain together the proxy for reaching the 1-Wire Network. Figure 3 illustrates the interactivity between the 1-Wire application, the client module, the host module, and the 1-Wire Network.

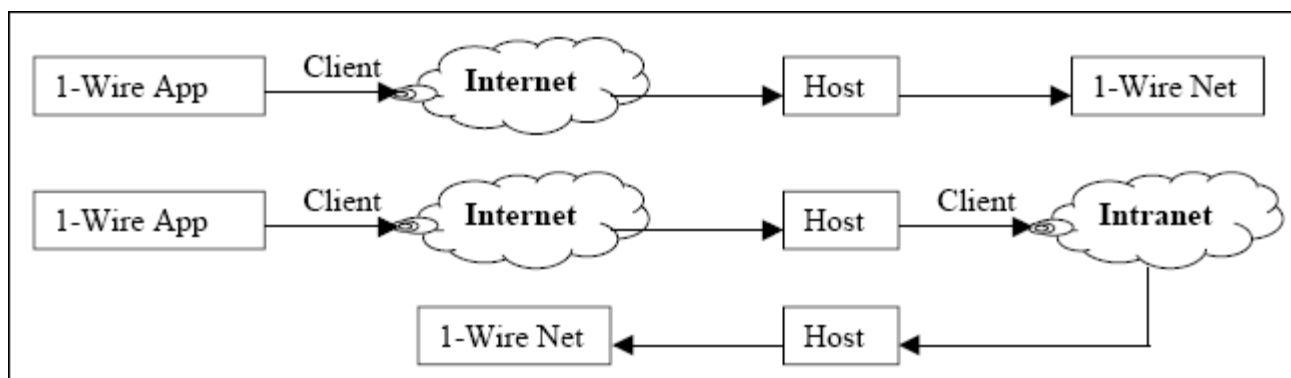


Figure 3. Client/host configurations.

The way a PC normally communicates on the 1-Wire Network is through the use of an adapter, usually with a suitable 1-Wire Master for producing correct timings. The DS9097U adapter, for example, uses a DS2480B Serial 1-Wire line driver chip. The PC is connected to the DS9097U using the serial port. Commands sent through the serial port reach the DS2480B and are then translated into 1-Wire Network commands. The DS2480B returns the output of any 1-Wire Network commands through the PC's serial port. However, when communicating with the 1-Wire Network through a 1-Wire Proxy, the PC will instead call on the client module to send commands to the host module. These commands will be transmitted through the network medium (e.g., the Internet) to the host module. The host module will then communicate through a 1-Wire Master with the physical 1-Wire Network. The results of the 1-Wire communication will be returned in a buffer to the client module back through the network medium.


```
Client Module transmits a serial write command to reset the DS2480B
  Packet = {XMIT_SERIAL, DS2480B_COMMAND_MODE, DS2480B_CMD_RESET}

Server Module writes bytes to serial port.

Client Module transmits a serial read command with a number of bytes to read
  Packet = {RECV_SERIAL, NUM_BYTES_TO_RECV}

Server Module transmits bytes read from serial port, representing return value of reset command
  Packet = {NUM_BYTES_TO_RECV, however many bytes...}

Client Module interprets return value
```

For each packet sent, add a multiplier for the packet latency. If multiple successive calls were made to the DS9097U in this manner, it's obvious that it can add up to a lot of network traffic rather quickly. At the presentation-layer, that exact same 1-Wire activity could be reproduced as follows:

```
Client Module transmits reset command
  Packet = {ADAPTER_CMD_RESET}

Server Module transmits reset command to serial port. Then reads bytes back from serial port, and
interprets the return value of reset command. Server Module transmits return value
  Packet = {RET_SUCCESS}
```

Moving the 1-Wire Proxy implementation to the application-layer could significantly reduce the packet latency problem. But since an implementation at this layer would be highly non-portable (i.e., an implementation at the application-layer would inherently be tied to a specific application), there is no reference implementation currently available. Just for the purposes of illustration, let's examine the difference with one specific application: temperature polling. A possible network packet exchange for a temperature conversion would generate numerous packets at both the link layer and the presentation layer. It would be necessary to transmit a '1-Wire Reset' command, a 'Match ROM' command, and a 'Do Temperature Convert' command. The client module would be responsible for interpreting the return value of each command as it is received in the form of a network packet from the host module. If implementing a 1-Wire Proxy at the application layer, it would only be necessary for the client module to transmit a 'Perform Complete Temperature Conversion' command. The return value from the server, in a single packet, could be just the result of the temperature conversion.

Software Interface Example

Using the client module in the 1-Wire API for Java Kit is as simple as creating an instance of the NetAdapter and passing a connection string to an initialization method. The format of the connection string is:

```
<hostname>:<port>:<shared secret>
```

Hostname is either the hostname of the PC (or MxTNI) that is running the host module or the IP address of that computer. The port is the TCP/IP port that the host is listening on. The shared secret is used for a simple form of authentication. When the client module connects to the host module, the host issues a random challenge to the client. The client then finds the CRC-16 of the random challenge bytes and the shared secret. This CRC-16 is then transmitted back to the host for verification. If the CRC-16 matches what the host calculated, the user is considered authenticated. Note that there is an additional initialization method available in NetAdapter that allows further steps for securing your connection by accepting an already established TCP/IP socket, which could be an encrypted connection.

In the 1-Wire API for Java Kit, there is a sample program that facilitates the use of the host module. This application takes all the parameters for the host module as arguments on the command-line, and uses these parameters to create an instance of NetAdapterHost. A pre-built binary is included for both the desktop and the MxTNI. Figure 5 illustrates the command line for starting the host module on the desktop using a DS9097U adapter on the COM1 serial port. The current working directory is the application folder for StartNetAdapterHost in the 1-Wire API for Java Kit.

```
java -cp ".;<path to onewireapi.jar>" StartNetAdapterHost -adapterName DS9097U
  -adapterPort COM1 -listenPort 6161 -secret "this is my secret"
```

Figure 5. Starting the host module on a PC.

Executing the line shown in Figure 5 on the PC will start the NetAdapterHost, which will create a listening TCP/IP socket on port

6161. The shared secret used for the simple authentication is "this is my secret". Starting the host module on a MxTNI is nearly identical. Simply ftp the StartNetAdapterHost.tini file to the MxTNI machine. Then log on and use the command-line shown in Figure 6.

```
java StartNetAdapterHost.tini -adapterName TINIExternalAdapter
    -adapterPort serial1 -listenPort 6161 -secret "this is my
secret"
```

Figure 6. Starting the host module on a MxTNI.

Once the host module has started, incoming connections will be accepted from the client module. Figure 8 shows a remote temperature demo that demonstrates how to connect to the host and interact with a 1-Wire device. This application will connect to the specified host and look for any temperature devices. It then polls each device and displays the current temperature in both degrees Celsius and Fahrenheit. To use the sample app, simply copy-and-paste the code into a file called RemoteTemperatureDemo.java. Change the constants at the top of the class to reflect the actual setup (i.e., make sure the hostname variable has the hostname of the computer that NetAdapterHost was started on). Then execute the commands shown in Figure 7 to compile and execute this class.

```
javac -classpath ".;<path to onewireapi.jar>" RemoteTemperatureDemo.java
java -classpath ".;<path to onewireapi.jar>" RemoteTemperatureDemo
```

Figure 7. Compile and run remote temperature demo.

The output of this program should be the 64-bit 1-Wire address of each temperature device that was found followed by the current temperature. If no device was found, an error message will be displayed indicating just that.

For more information about the use of NetAdapter and NetAdapterHost, see the JavaDocs for these classes. The JavaDocs for all 1-Wire classes are included with the 1-Wire API for Java Kit.

```

import com.dalsemi.onewire.adapter.*;
import com.dalsemi.onewire.container.*;
import com.dalsemi.onewire.utils.*;
import java.util.Enumeration;

public class RemoteTemperatureDemo {
    // update these variables to reflect your setup
    public static final String hostname = "shughes.dalsemi.com";
    public static final int port = 6161;
    public static final String secret = "this is my secret";

    public static void main (String[] args) {
        OneWireContainer owc = null;
        TemperatureContainer tc = null;

        // create the NetAdapter object
        NetAdapter adapter = new NetAdapter();

        try {
            // connect to the NetAdapterHost
            adapter.selectPort(hostname + ":" + port + ":" + secret);
            adapter.beginExclusive(true);

            // find the first temperature device
            Enumeration e = adapter.getAllDeviceContainers();
            while (tc==null && e.hasMoreElements()) {
                // get the next container
                owc = (OneWireContainer)e.nextElement();
                // check if it is a temperature device
                if(owc instanceof TemperatureContainer) {
                    tc = (TemperatureContainer)owc;
                    System.out.println("Device: " +
owc.getAddressAsString());

                    // poll the temperature device
                    byte[] state = tc.readDevice();
                    tc.doTemperatureConvert(state);
                    double temp = tc.getTemperature(state);

                    // display temperature result
                    System.out.print(" " + temp + " C (");
                    System.out.println(Convert.toFahrenheit(temp) + "
F)");
                }

                // if no temperature devices were found
                if(tc==null)
                    System.out.println("No temperature devices found!");
            } catch (Exception e) {
                System.out.println(e.getMessage());
            } finally {
                adapter.endExclusive();
                try {
                    adapter.freePort();
                } catch(Exception e) {}
            }
        }
    }
}

```

Figure 8. Remote temperature demo.

References

1. [General-Purpose Microcontrollers](#)
2. Application note 704, "[Asynchronous Serial-to-Ethernet Device Servers](#)"
3. [1-Wire API for Java](#)
4. [IEEE 1451.4 Transparent Protocol \(PDF, 147kB\)](#)

1-Wire is a registered trademark of Maxim Integrated Products, Inc.

iButton is a registered trademark of Maxim Integrated Products, Inc.

IEEE is a registered service mark of the Institute of Electrical and Electronics Engineers, Inc.

Java is a registered trademark and registered service mark of Oracle and/or its affiliates.
MxTNI is a trademark of Maxim Integrated Products, Inc.

Related Parts

DS1411	Serial Port iButton Holder	
DS2480B	Serial to 1-Wire Line Driver	Free Samples
DS2490	USB to 1-Wire Bridge Chip	
DS9097U	Universal 1-Wire COM Port Adapter	

Next Steps

EE-Mail	Subscribe to EE-Mail and receive automatic notice of new documents in your areas of interest.
Download	Download, PDF Format (76kB)
Share	Other Channels E-Mail this page to an associate or friend.

More Information

For Technical Support: <http://www.maxim-ic.com/support>

For Samples: <http://www.maxim-ic.com/samples>

Other Questions and Comments: <http://www.maxim-ic.com/contact>

Application Note 193: <http://www.maxim-ic.com/an193>

APPLICATION NOTE 193, AN193, AN 193, APP193, Appnote193, Appnote 193

Copyright © by Maxim Integrated Products

Additional Legal Notices: <http://www.maxim-ic.com/legal>