Embedding it better...

µTasker

µTasker Document

IPv6

Table of Contents

1. Introduction

IPv6 (Internet Protocol Version 6) was developed in the early 1990s as it became apparent that the growth of the Internet, based on IPv4, would eventually reach the limits of available address space and result in serious complications and restrictions. Originally known as IP Next Generation (IPng), IPv6 was expected to replace the original IPv4 in the early 2000's but the economical incentive to invest in the new technology was still missing and further development of techniques in the use of existing IPv4 technology, such as NAT (Network Address Translation) or DHCP (Dynamic Host Configuration Protocol) with limited lease times, allowed an ever increasing number of users to more efficiently share the limited space available. The useful life time of IPv4 was continuously being increased whilst the demand for its remaining space was continuously increasing – the predicted point in time when the IPv4 space would finally become exhausted proved to be always inaccurate and updated predictions kept giving it another few years. IPv6 importance would occasionally be hyped and then again extinguished in regular cycles from the end of the 1990's to the end of the 2000's.

But the facts do speak clearly as the Internet has developed and also experimental and research IPv6 infrastructure has been deployed and extensively tested. Some facts are that today there are about 300 million IPv4 addresses still available [2010] for allocation, but – even with the further development of ever improved methods to share the available resources – this address space is still depleting at a rate of around 150 million IP addresses per year. The need for new addresses will not suddenly drop in the foreseen future but instead will rather increase – *Asia still has a lot of catching up to do and new mobile applications, as well as many other imaginable technological improvements in the near future, will cry out for more space.* Probably there will be breakthroughs to help extend the life of IPv4 technology past the expected 495 days as is being predicted at this point in time [May 2010]; may be the breakthrough will keep it on life support for another year – maybe another two or three years, maybe longer, but this battle, no matter how noble and heroic it is, will have its end. IPv6 will be the network of the future and there is no reason why one shouldn't be ready for it when it is time and also no reason why not to already take advantage of its capabilities today.

The following document will do several things and not do several others. First of all it will *not* discuss IPv6 in a conventional way. It will *not* start with examples of the address space available to impress the reader of just how large it is. It will *not* start by discussing IPv6 addressing and sub-netting differences compared to IPv4 and it will *not* start by discussing how an IPv6 packet is constructed. There are enough such examples available in the IPv4 and IPv6 Internet and no need for further such works.

It *will* however start with very practical explanations of how one can start testing real IPv6 operation with a PC, observe it in action, and even get it running on a single-chip embedded device. Of course the details will be discussed on the way where necessary and of interest but the goal is to do serious work with the already existing capabilities, starting with the most simple of tools and resulting in maximum advances in understanding and practical achievement.

The reader is expected to have a basic understanding of Ethernet and working in a simple IPv4 network as well as connecting to the Internet from a PC through a typical router (gateway – like an ADSL or Cable modem, etc.). Furthermore, the practical work is based on the µTasker project to enable simulation of embedded processors and real-time simulation of the Ethernet connection used to make real network tests. Users of the µTasker project can further extend the examples and principles to real embedded devices and real project which can make use of the techniques and technology.

2. Can your PC connect to the IPv6 Internet?

If you have an old computer the chances are that it can't. The reason being that IPv6 support has really only been available since Windows XP Service Pack 2. Around the same time (maybe even a little earlier) Linux and MAC operating systems also became IPv6 capable. This document discusses details of working with Windows Vista – other operating systems may require slightly different commands and operation but there are usually good guides similar to the ones pointed to when Windows Vista is discussed, so users of other systems should be able to follow things without too much difficulty.

Even if your PC does support IPv6 it may not be enabled. It may also be that it could do but is restricted due to factors to do with the network that it is connected to. Furthermore it may be that it could but you have never tried. Maybe you never thought of trying or never needed to try or just don't have any idea of how to try. It may even be that it is using IPv6 but you never realised!

The typical PC user is probably not interested in IPv4 or IPv6 but rather more in being able to connect to the Internet to send and receive emails, or even more probably to browse information in the World Wide Web. Seeing that browsing the web (using Explorer, Firefox or any other of the available web browsers) is probably the number one use of the PC together with the Internet, it makes sense to have a go at browsing to a web site that supports IPv6. For more network oriented people "pinging" such a site is probably also rather exciting too. So why not have a try?

In your favourite web browser try to contact the with the adress http://www.kame.net/ or, from a DOS window, try pinging it with `ping -6 www.kame.net`. There are in fact quite a lot of web sites which support IPv6 – some are only available via IPv6 and others can be contacted by either IPv6 or IPv4. www.kame.net supports both and so, as long as you do have an operational Internet connection it will display – its speciality is an image of a turtle; if you connect using IPv6 the turtle will dance; if with IPv4 it will be rather lifeless.

Try also http://www.apnic.net/ This web site will display your IP address – either the IPv4 address or the IPv6 address. Ping it using `ping -6 www.apnic.net`.

If you see the dancing turtle and can ping the web sites successfully then your PC and network are already operating in a manner allowing IPv6 to be used. If not then it is time to get them sorted out!

3. Does your Internet Connection support IPv6?

One thing to bear in mind is that the Internet was constructed using IPv4. Not all of the networks used in the Internet can carry IPv6 traffic!

Often the connection to the Internet backbone, which will probably be able to carry IPv6 traffic, can only carry IPv4 traffic. That means that the connection at the other side of your router or gateway (often referred to as your NAT – Network Address Translator) can only carry IPv4 traffic, so, how are you expected to be able to use this super new technology? While there are good IPv6 backbones in Europe, for example, there are some continents where this is still completely missing.

Remember that it will also take some time to upgrade the networks in the Internet of today to seamlessly support IPv6. When the IPv4 address space eventually runs out and IPv6 becomes really popular no one will be throwing a switch to turn off the existing IPv4 network and switching on a brand new IPv6 one. There will instead be a gradual transfer phase while users start making more use of IPv6 capabilities and the network providers gradually upgrade their equipment until one day, in the rather distant future, one will realise that IPv4 has in fact completely died out and everything, including the networks involved, are all IPv6 capable. *Our grandchildren will probably laugh at the thought of a world based on IPv4 networking as our children today laugh at the thought of people who used to watch black and white televisions...*

This means, in fact, although we are just taking very first steps with IPv6 operation it turns out that it probably doesn't actually work today and this situation will also hardly be solved in the near future!

Therefore we are in fact not going to start with IPv6 at all. To do that would not be practical in most situations outside of a research establishment or various university campuses – we are in fact going to start with IPv4. However, we'll use it to make our own network connections start to really allow us to work with IPv6. Even if your first test showed that you can operate using IPv6 (*and it is not already using the technique that will be described to achieve this*) it probably only works where you happen to be right now, which would not be of much use to you when you happen to be travelling around South Africa demonstrating your own IPv6 capable equipment that you have developed for monitoring climate changes. For the foreseeable future the use of IPv4 to achieve your IPv6 operation will need to be respected too.

4. IPv6 Tunnelling

There are a number of methods available for operating IPv6 over IPv4 networks. These involve tunnelling the IPv6 traffic in IPv4 traffic, which simply means that the IPv6 part is carried by the IPv4 layer.

One technique which is used by Windows Vista as a standard method is to carry IPv6 within UDP frames. This is called the Teredo protocol on UDP port 3544 and requires the help of a Teredo server in the Internet, but allows clients to operate with no fixed IPv4 address in the Internet.

Another technique is called AYIYA (Anything in Anything) which can tunnel inside the data part of either UDP, TCP or SCTP frames. It also solves some additional security aspects.

Probably the simplest technique is called 6in4 and simply adds the IPv6 frame to an IPv4 header with the IPv4 protocol type declared as proto-41.

There are RFCs for these protocols and also discussions of their operation and configurations readily available in the Internet and so there will be no great comparison of the techniques made here. The following concentrates on 6in4 for tunnelling due to the fact that it is simple to understand, setup and can also be used to achieve a number of standard requirements. In particular, the following objectives were set and solved by using the simple techniques explained in the following sections of the document:

- Obtaining global IPv6 addresses for use by servers
- Enabling multiple IPv6 addressable servers to be deployed behind a simple IPv4 NAT
- Enabling multiple clients behind a simple IPv4 NAT to be able to communicate with IPv6 servers in the Internet

It is to be noted that many people are using NATs (their local Ethernet router/gateway) which have no understanding of 6in4 tunnelling. This means that there may be very limited configuration capabilities, but the following techniques show how it is still possible to achieve the specified objectives with any available NAT as long as it offers the capability of specifying a DMZ (Demilitarized Zone). *It is assumed that readers will have the rights to configure the DMZ of their own NAT or can arrange this configuration to a reserved IPv4 address in the local network.*

New NAT devices may support extended configurations which make it possible to achieve the same results but at the moment such capabilities tend to be not well defined and therefore the worst-case scenario is discussed, so that a solution should always be possible.

5. How 6in4 operates

To begin the discussion of how 6to4 actually works a typical IPv4 situation will be presented. As example, three PCs are connected on a local network and all have access to IPv4 servers on the Internet via a simple NAT device. The NAT device allows all PCs to share a single IPv4 connection by monitoring traffic being sent out from the local addresses and routing returning traffic back to the same source – the process is known as Network Address Translation. This is shown in figure 5-1, whereby example IPv4 addresses are given to each local PC, the NAT and some servers in the Internet.

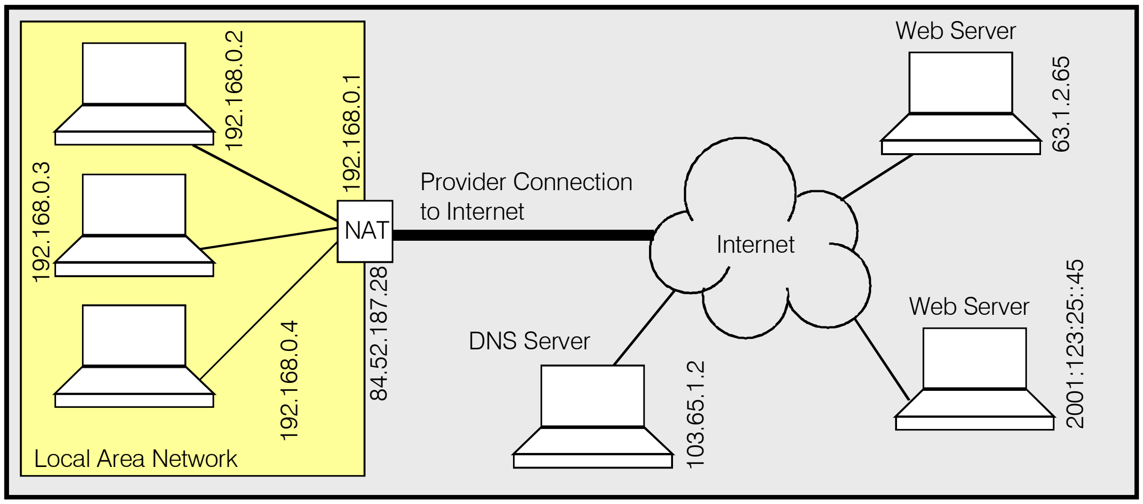


Figure 5-1 Simple LAN connected to Internet via NAT

Consider the case when two of the local PCs (local IP addresses 192.168.0.2 and 192.168.0.3) want to connect to the same web server in the Internet. The name of the web server is generally used – example http://www.webserver.com – which is simply entered as URL in the web browser. The PC will probably not know where this web server is and so will first ask its DNS server, a server recommended for use by the Internet provider since it is located locally to the connection. In this example it is a server at the IP address 103.65.1.2, which is an external address and so the request will be routed via the gateway (NAT) with the local (gateway) address of 192.168.0.1.

The NAT has two ports; the local IP address in the LAN and a global IPv4 address in the Internet. The IP address in the Internet can be either a fixed IP address as allocated to the owner of the connection or a dynamic address, which is temporarily provided to the NAT by the Internet provider's DHCP server. *For the rest of the discussion it will be assumed that this is effectively a fixed address since it is also possible to configure a dynamic DNS entry so that this is emulated via a URL entry.*

The DNS server will answer the request with the IP address of the server (63.1.2.65) so that the web browsers can then establish connections with that server, which could be situated anywhere in the Internet. During this process all returned frames matching the transmitted protocol and port numbers of the requests will be returned locally to the PC from which they originated from, resulting in the two PCs (or more in the local network) being able to share one Internet connection, even if operating at the same time.

The operation of the NAT when the frames originate from the Internet and there is no previous request originating from within the LAN will be discussed later when servers in the LAN are considered.

It is important to note that the network address translation taking place in the NAT is usually based on UDP and TCP ports. The behaviour in cases when the frames don't use UDP or TCP based protocol may not be defined and will typically result in the NAT failing to achieve translation and any frames originating from the Internet to be discarded. This shortcoming of many existing NATs is a potential problem to the use of 6in4 tunnelling and a method to overcome it is discussed later.

This initial example was based on a web server in the Internet that is reachable on an IPv4 address. There are however some web servers which have both IPv4 and IPv6 addresses and even those with only IPv6 addresses. These details are also maintained by the DNS server, which will respond accordingly with additional information concerning this connectivity.

- If a web server has only an IPv4 address, this IPv4 address will be returned. This is known as an A type address.
- If a web server has only an IPv6 address, this IPv6 address will be returned. This is known as an AAAA address.
- If a web server has both an IPv4 and an IPv6 address, both addresses will be returned. The answer thus contains an A and an AAAA address.

Should the PC not be configured to work with IPv6 it will ignore any IPv6 address information and only use the IPv4 address. Pure IPv6 addresses will result in connection failure in such cases.

If the PC is configured to work with IPv6 (as well as IPv4 – known as dual-stack) it will tend to try to connect to the IPv6 address and only drop back to the IPv4 address if the IPv6 connection attempt fails.

In figure 5-1 a web server at the IPv6 address 2001:123:25::45 is shown. If the PCs were configured for IPv6 operation (which is likely when modern operating systems are in operation), the NAT passes IPv6 traffic correctly and the Internet provider's connection into the Internet can carry IPv6 raw traffic into the Internet's IPv6 backbone, it would be possible to connect also to this web server with no further consideration.

As discussed in section 3 this is however not likely to be the case for the majority of users at the time of writing and so the addition of a 6to4 tunnel needs to be considered as illustrated in figure 5-2.

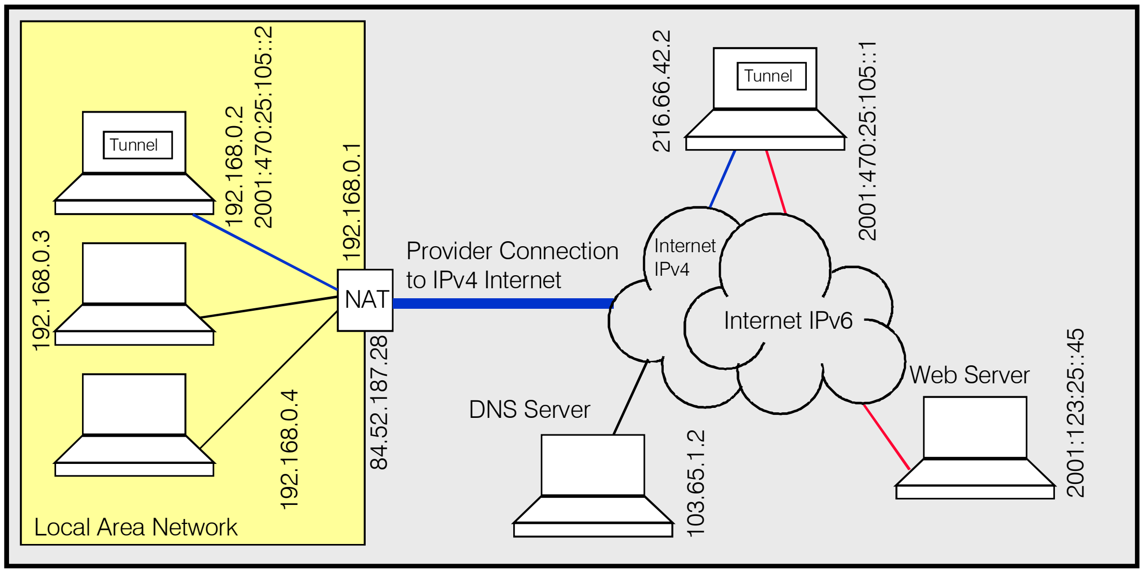


Figure 5-2 Simple LAN connected to Internet via NAT and carrying tunnelled IPv6 traffic

One PC in the LAN is shown configured with a software tunnel. This packs the IPv6 frames to be sent to the remote web server into IPv4 frames. *Examples of the actual frame content will be shown later but for the moment one simply needs to envisage these as IPv4 datagrams with an IPv6 payload*. The IPv4 frames are sent to a tunnelling server in the Internet, whos IPv4 address is known and which, on its part, also knows the address of the NAT in the IPv4 Internet corresponding to the PC's global IPv6 address. This server has one connection to the IPv4 Internet (216.66.42.2), which receives these frames, and one in the IPv6 Internet (2001:470:25:105::1), from which it sends the raw IPv6 frames after it has removed the IPv4 header.

Return frames go through the reverse process so that the PC in the LAN can achieve tunnelled IPv6 operation. The user of the PC can work with IPv6 web sites by URL or also by IPv6 address and doesn't notice the tunnelling operation taking place.

The next section informs of how to use a tunnel server (known as a tunnel broker), how to configure a Windows Vista PC for its operation, and how to temporarily overcome a potential deficiency in common NAT capabilities.

6. Tunnel Broker Service and Configuring a Windows Vista PC

There are a number of IPv6 Tunnel Brokers offering network tunnels of the type described generally in the previous section. This section gives a step-by-step guide on how to simply obtain IPv6 address space from a tunnel broker and how to start using this. The description and testing tools are based on a single IPv6 Tunnel Broker, *Hurricane Electric*, and doesn't express any preference for this provider nor does it signify that other providers offer worse services – experiences with Hurricane Electric have shown however that they offer a straight-forward and un-bureaucratic solution for first IPv6 work.

The Internet connection, as described in the example in the previous section, has a global IPv4 address. This is a unique address within the IPv4 address space, which may have been assigned as fix address or may have been dynamically assigned by the Internet service provider – a fixed IPv4 address may not always be available or possibly only by paying extra fees.

The first thing that we will do is obtain an IPv6 address. In fact this will not be restricted to a single IPv6 address but will be a block of address space which is typically a /64 prefix, meaning that its size is 2^64 = 18'446'744'073'709'551'616 (18 quintillion). Compared to the existing IPv4 address space this is about 16 billion times larger. This is also free of charge. Once the tunnel has been set up it is possible to get your own /48 prefix, 64'000 times larger again to allow further sub-netting many of these huge address spaces.

Hurricane Electric's web site can be found at http://tunnelbroker.net/. Register by filling out the simple form with name, address, email address, phone number and a password. A confirmation email will immediately be sent to your email address and then you can log in to set up your tunnel.

To set up the tunnel use the User Function "`Create Regular Tunnel`". A form will be displayed showing your present viewing IPv4 address and suggesting a tunnelling server to be used from your location – use the suggested one if it is looks reasonable or else select one that is closest to you. Enter the address of your NAT as IPv4 endpoint – this will usually be the same as the IPv4 address being shown as your viewing address.

Press "Submit" and the tunnel will be created. You will also get a list of commands which should correspond to these tunnel settings on various operating systems so copy the ones that are appropriate and save them in case you need to refer to then again later. The following details the specific case for Windows Vista.

Note that you can always regenerate these details by logging in and clicking on your tunnel. The tunnel's settings are displayed and example OS configurations can be generated by selecting the operating system.

The tunnel has 4 endpoints:

- Server IPv4 address: the tunnel server which you have selected as being closest to your location. This is `216.66.42.2` in figure 5-2.
- Server IPv6 address: the tunnel servers IPv6 address – `2001:470:25:105::1` in figure 5-2.
- Client IPv4 address: this is your NAT's global IPv4 address – `84.52.187.28` in figure 5-2.
- Client IPv6 address: this is the IPv6 address that will be assigned as global IPv6 address to your PC. This is `2001:470:25:105::2/64` in figure 5-2.

This means that the tunnel in the Internet is now available for use but the local PC will not yet be configured to use the new global IPv6 address. Furthermore the PC will not work directly with the IPv6 network but instead needs to be configured to use the 6in4 tunnelling mechanism. These configurations are therefore the next step to be accomplished.

The Windows Vista configuration for the example configuration (the addresses will be exchanged with the ones assigned to your own tunnel) will look like this:

netsh interface teredo set state disabled

netsh interface ipv6 add v6v4tunnel IP6Tunnel ~~84.52.187.28~~ 216.66.42.2

netsh interface ipv6 add address IP6Tunnel 2001:470:25:105::2

netsh interface ipv6 add route ::/0 IP6Tunnel 2001:470:25:105::1

Important – these commands can only be executed when the user has administrator rights. To start a DOS window with administrator rights do the following:
Click the Windows Start button and then enter `cmd` in the search window. The DOS command will appear (it is called `cmd.exe`). Using the right mouse click on it and select that it should be started in administrator mode. This will then need to be confirmed as the program starts.

The first command disabled the Teredo tunnelling which is probably enabled by default in Vista. The second command installs the tunnel. However, this is not correct for a configuration which is behind a NAT. Rather than the global IPv4 address of the NAT the PC's IPv4 address in the local network should be entered. In the example this is

```
netsh interface ipv6 add v6v4tunnel IP6Tunnel 192.168.0.2 216.66.42.2
```

The third command assigns the global IPv6 address to this tunnel.

The final command adds a routing entry for the tunnel server's global IPv6 address.

At this point it is theoretically possible to start using the PC to communicate with IPv6 servers in the Internet across the tunnel. It is also theoretically possible to run an IPv6 service on the PC, whereby it can be contacted by any IPv6 compatible client at its unique global IPv6 address. Whether this works or whether it partly operates depends now on the NAT in use and possibly on firewall settings.

The next sections shows how to check what is already working and what not, then how to work-around a NAT that does not already support protocol 41.

The following are further useful command to delete installed tunnels:

```
netsh int ipv6 help

netsh int ipv6 delete interface IP6Tunnel

netsh int ipv6 delete route ::/0 IP6Tunnel 2001:470:25:105::1

netsh int ipv6 delete address IP6Tunnel 2001:470:25:105::2
```

7. Testing Tunnelled IPv6 Connectivity

For the following tests it is advisable to monitor network traffic using Wireshark or another network sniffer. Wireshark can be downloaded from http://www.wireshark.org

You may like to first check to see whether you can see the dancing turtle at www.kame.net. If it now dances you are in luck because it means that the configuration does indeed already work for this PC. If not, you will however experience a difference to an attempt made before configuring the tunnel – the difference is that the connection to the web site has become very slow. It may take up to one minute for the static turtle to finally be seen. This can be explained by the fact that the PC is now attempting to establish a connection with the IPv6 address of the remote server; it knows its IPv6 address since this was reported by the DNS response obtained by from the standard IPv4 DNS server. The PC now tries but – for a not yet known reason – fails. After some attempts it finally drops back to the IPv4 connection. The same slowness will be experienced for any web server in the Internet that 'could' be reached on its IPv6 address. For this reason it is also useful to be able to disable the failing tunnelling operation if it proves to disturbing – this is accomplished as follows:

```
netsh int ipv6 delete interface IP6Tunnel
```

To reinstall it again later only the following single command is required:

```
netsh interface ipv6 add v6v4tunnel IP6Tunnel 192.168.0.2 216.66.42.2
```

The following assumes that the IPv6 tunnel operation was not successful in establishing a connection with the test web server. In this case the network sniffer should be started to check what is taking place. A ping of the web site can be commanded using:

```
ping www.kame.net
```

This should ping the web site on its IPv4 address to verify that basic contact is possible.

```
ping –6 www.kame.net
```

This will force the ping test to be sent to its IPv6 address. Presumably this will fail.

Studying the network recording will show that the address was resolved by requesting the AAAA address of the destination web site before sending an ICMPv6 echo request. Wireshark will display the echo request as an IPv6 frame. This is good when only the IPv6 content is of interest but it hides the fact that it is indeed tunnelled IPv6 – this details is only visible when actually expanding the frame layers to show that it is built up from a frame containing Ethernet II + IPv4 + IPv6 + ICMPv6 layers; expanding the IPv4 layer content reveals that it is protocol 41 (0x29), even though Wireshark may interpret this as IPv6 rather than protocol 41. The IPv4 source and destinations are the local PC's IPv4 address and the tunnel server's IPv4 address. The IPv6 addresses of the local PC and the remote IPv6 web server are only actually visible in the IPv6 layer itself.

Figure 7-1 shows the construction of the ICMPv6 echo frame. *As comparison, a raw IPv6 frame would simply not contain the IPv4 part*.

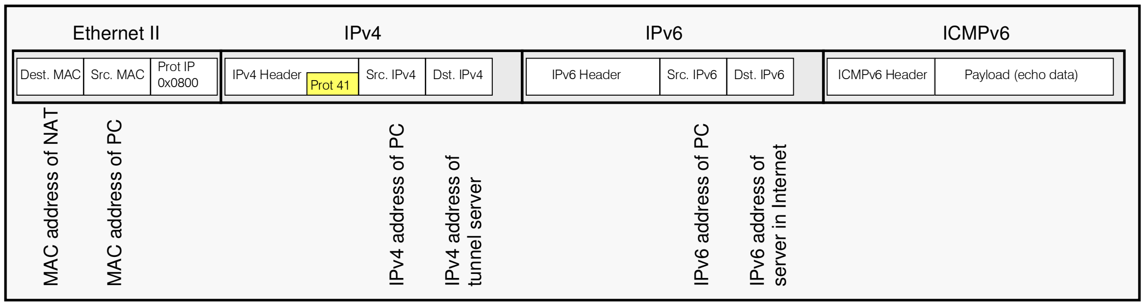


Figure 7-1 Simplified ICMPv6 frame showing the IPv6 layer tunnelled in IPv4 protocol 41

Most likely there will be no answer to the ping request, so the question is what happened to it? We know that the remote server is on-line because it could be successfully pinged using its IPv4 address and the 6in4 tunnel is sending the frame to the tunnel server's IPv6 address (an attempt to ping its IPv4 address [`216.66.42.2`] should also verify that the server is on-line). Assuming no other failures in the Internet, the chances are good that the ping response is in fact arriving back at our NAT as a 6in4 IPv4 frame but the next question is how the NAT actually reacts to this? NATs generally match outgoing frames with UDP or TCP protocol but will often not know what to do with an IPv4 protocol 41 frame and will simply discard it. The only time that it will not discard it is if a DMZ (Demilitarized Zone) has been configured to accept all non-matched reception. It is this DMZ that will allow us to get the tunnelling process to at least be able to operate behind such a NAT.

Before configuring a DMZ it is important to point out that a DMZ allows a local IPv4 address to receive all unmatched traffic arriving at the NATs global IPv4 address. This has the effect of exposing this address to the Internet without any firewall protection – it is thus not advisable to put the local PC in the DMZ apart for perhaps a quick test to confirm that the ping responses do then successfully arrive and that the IPv6 operation is also generally successful.

Another method of verifying the theory without having to actually expose the local PC directly to the Internet is to configure a non-existent local IPv4 address in the DMZ. In our example case there is no PC on address `192.168.0.12` so this could be used.

To configure the DMZ one needs first to log in to the NAT, which usually contains a password protected web server. The user's guide to the NAT may need to be consulted for exact details. Once logged into the NAT it will almost certainly have a DMZ configuration which requires enabling and also configuring with the local IPv4 address. Once the change is applied all unknown traffic from the Internet will be routed to this address.

Even if the DMZ IPv4 address has been set to a non-existent address in the local network (this is also harmless since no real PC is exposed to the traffic) the ping responses can be verified. This is due to the fact that each reception (the ping response arriving at the NAT) will cause the NAT to try to resolve the local MAC address of the non-existent device. This is in the form of an ARP broadcast to the DMZ IPv4 address. Following the ICMPv6 echo request there will thus be an ARP broadcast looking for the non-existent address. The broadcast will be seen after the round-trip delay which will probably be a little longer that the round trip delay of the IPv4 ping due to the additional tunnels in the patch. When this ARP resolve is consistently seen in response to the IPv6 ping one can conclude that it is really due to the ping response.

By configuring the PC with the IPv6 tunnel in the DMZ it should finally be possible to browse to the web site www.kame.net and see the dancing turtle. As mentioned above, this configuration would not normally be retained for much longer than a quick test due to the risk involved in exposing the PC directly to Internet traffic. Although it has indeed worked we need a better solution. The solution may involve upgrading the NAT to one which offers forwarding of protocol 41 but this still may not achieve all goals, like allowing multiple users to work with the single NAT and allowing multiple local devices to be contactable from the Internet on their global IPv6 address.

The next section presently a technique that can be used to achieve these goals in case no advanced NAT is available or no such flexibility exists even in protocol 41 aware NATs.

8. Introducing the 6in4 Relay Agent

Configuring a DMZ to catch all received IPv4 frames not specifically handled by the NAT is probably the simplest method of ensuring that tunnelled IPv6 frames get through to the local network. Putting anything other than a server to be specifically accessed from the Internet is however a potential security risk. Consequently, it is not a particularly good idea to place a PC in the DMZ just to allow it to be used to browse to IPv6 web sites.

Imagine however putting a dedicated piece of equipment in to the DMZ which otherwise represents no risk to the local network. This could be a dedicated PC or a networking device performing the missing functions that are lacking in our standard NAT. It could also be a special piece of software operating on our own PC or any other in the local network.

The actual task of this relaying function is quite simple. It needs to receive all IPv4 frames that the NAT doesn't understand and check to see whether they are tunnelled IPv6 frames. If they are, it then needs to look at the IPv6 address that is tunnelled in the frame to see whether it matches with a global IPv6 address in the local network. If not both are true, it simply discards the frame. If they both match it can do one of the following:

1. Deliver the 6in4 frame to the destination in the local network
2. Strip off the IPv4 header and deliver the raw IPv6 frame in the local network

Both of these options are in fact quite simple to achieve as is shown in figures 8-1 and 8-2. The more restrictive part of the exercise is the fact that the device needs to maintain a list of the IPv6 addresses and their matching IPv4 addresses in the local network. In a larger network this may require some dynamic technique to make it practical but in smaller local networks, where it is possible to set fixed IPv4 and global IPv6 addresses to the PCs - or other IP-enabled devices - it represents a real solution to quickly start taking advantage of IPv6 operation.

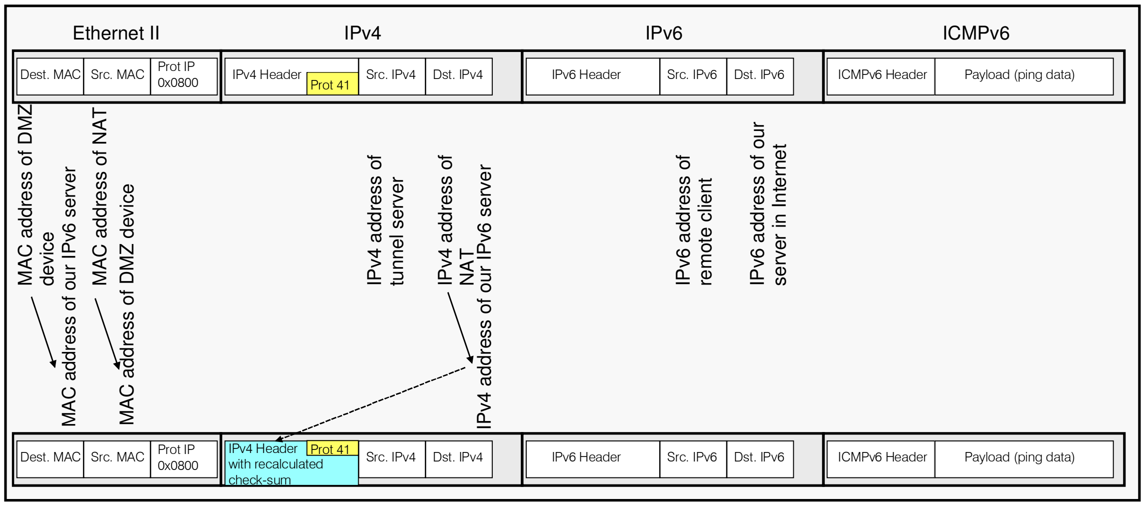

Ethernet II
IPv4
IPv6
ICMPv6
Dest. MAC
Src. MAC
Prot IP 0x0800
IPv4 Header
Prot 41
Src. IPv4
Dst. IPv4
IPv6 Header
Src. IPv6
Dst. IPv6
ICMPv6 Header
Payload (ping data)
MAC address of DMZ device
MAC address of our IPv6 server
MAC address of NAT
MAC address of DMZ device
IPv4 address of tunnel server
IPv4 address of NAT
IPv4 address of our IPv6 server
IPv6 address of remote client
IPv6 address of our server in Internet
IPv4 Header with recalculated check-sum

Figure 8-1 Simplified ICMPv6 frame showing received frame being relayed to the local destination

Note from figure 8-1 that the modification of the IPv4 layer requires a recalculation of the IPv4 check-sum before the frame is sent on to its final destination. Furthermore, the modification of the source MAC address is not absolutely necessary but it would otherwise look as though the NAT were sending first the received frame to the DMZ address and then to the local destination – with the MAC address corresponding to the relaying device the relationship is clearer.

Since tunnelled IPv6 frames are often only expected from a single tunnel server in the Internet a further check of validity could be to verify the source address in the IPv4 header as criteria to accept the frame.

By further checking the port number of UDP and TCP payload firewall functions can be added. For example, only TCP port 80 could be relayed if only standard HTTP traffic were to be allowed.

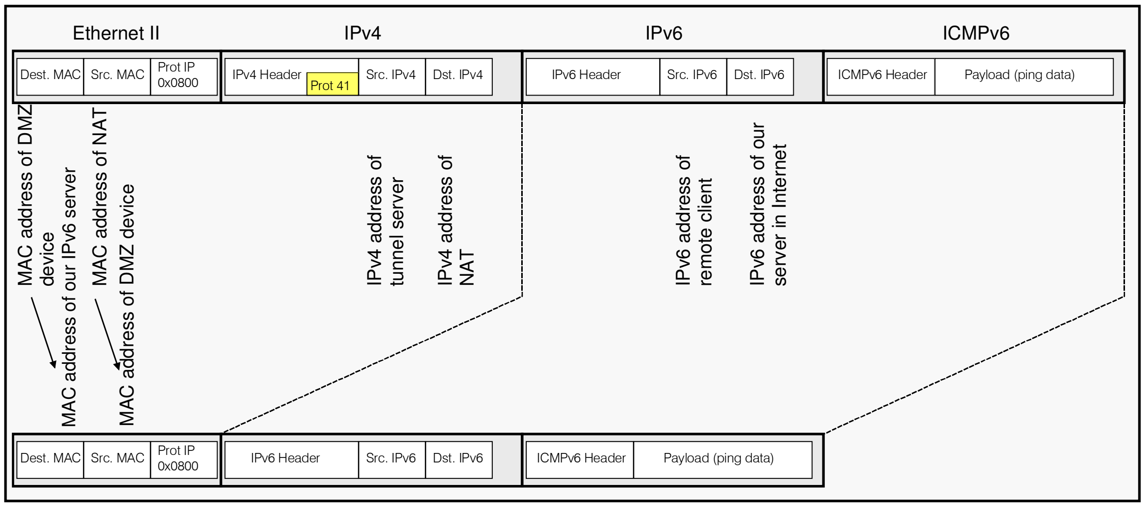


Figure 8-2 Simplified ICMPv6 frame showing received frame being relayed to the local destination as raw IPv6 frame

As shown by figure 8-2 it is in fact even simpler for the relaying device to extract the raw IPv6 content and send it to the local destination. In this case no check sum needs to be recalculated and the process is thus faster. However, although a destination which can receive IPv6 frames as raw or tunnelled frames would accept both cases, the Windows Vista case restricts receptions to the tunnel – it will not accept frames addressed directly to the IPv6 address. This means that whether the final deliver is made as a tunnelled IPv6 frame or as a raw IPv6 is best left as a configuration parameter to ensure that all cases are possible.

The following discussion turns its attention to how the µTasker simulator can be used as this special piece of software and the same code then be compiled and loaded to any of a number of standard development/evaluation boards to effectively create an alternative networking device to take over this roll.

First of all the reader should be aware that the µTasker project includes a simulator which uses Microsoft Visual Studio to run embedded code for several popular single-chip processors on a Windows PC, whereby the Ethernet operation of the embedded code is hooked to the PC's NIC (Ethernet adapter) by using the WinPcap library. WinPcap is also used by the well known network sniffer Wireshark and so will already be installed when Wireshark is installed on the PC. When the simulator is running it can receive real Ethernet frames on the local network and can also sent frames that its software generates. The simulator can also run as an executable, meaning that it doesn't need to be started within the Visual Studio environment but as a standard Windows program.

A simulated device can have a different MAC address and different IP address to that of the PC that it is running on. The Windows TCP/IP stack is not used in the process but instead the µTasker embedded TCP/IP stack is used to handle all raw Ethernet data. This means that by putting the simulator into the DMZ of the local network it can receive all frames which the NAT would otherwise discard. Since the IP Mac and IP address of the Windows PC that the simulator is running on it different this also doesn't represent any risk to the PC itself, which is not in the DMZ and is still protected by the firewall functions of the NAT.

Adding a 6in4 relay to the network is thus possible by running an executable of a µTasker project containing the discussed functionality. The simulator looks like a standard device on in the local network to the PC that it is running on and to others in the network. Since the µTasker project has also a standard web server running on its IPv4 address the configuration of the relaying table can also be made via web browser from any local PC as well as from PCs in the Internet if this is allowed. In addition to the relaying function, the simulator can also have its own global IPv6 address (as well as link local IPv6 address) so that it can act in parallel as an IPv6 server located directly in the DMZ.

Owners of development/evaluation boards can run the same code on real hardware to perform the same function so that no PC is required to operate all the time to ensure the operation.

The named goals can thus be achieved by using this technique:

1) As long as the IPv6 addresses of devices behind the NAT are correctly configured in the relay agent all of these addresses can be contacted from the Internet on their own IPv6 address. One single IPv4 address can thus be shared by a number of IPv6 addresses.
   Unlike using virtual servers on a single IPv4 NAT to share it with multiple servers, all of the IPv6 addresses can operate on the same port number (eg. TCP port 80 for HTTP). This avoids potential problems of having to put web servers on non-standard ports that are then blocked by firewalls in remote networks, making this technique unreliable in practice.
2) Multiple clients can share the single NAT and tunnel since the relay agent uses the tunnelled IPv6 information to deliver returned frames.

The following section discusses adding the µTasker 6in4 Relay Agent to a practical network.

9. Installing and Configuring the Relay Agent

The Relay Agent is an optional service in the µTasker project. It has two functions:

- first of all it is used to pass tunnelled 6in4 frames to the local IPv6 stack when the IPv6 address matches the IPv6 global address. This is used when the local node is working with tunnelled IPv6 in order to be able to receive this type of data. This is enabled when `USE_IPV6INV4` is set.
- The second function is to relay to other devices in the network as described in the previous section. This is enabled when `USE_IPV6INV4_RELAY_DESTINATIONS` is enabled and not equal to 0. This value also defines number of destinations that can be relayed to. Each entry (global IPv6 and IPv4 addresses plus MAC address of the node) has to be configured in the table to suit the local network. Although it would be possible to resolve the MAC address using ARP this is made fixed in order to simplify the realisation and avoid having to otherwise store data intermediately while the address is resolved.

When there is a table of destination addresses that can be relayed to the user enters these by calling

```
fnSetIPv6in4Destinations(IPV6INV4_RELAY_DESTINATION *ptrTable);
```

The location of the tables is flexible and the content can be modified as required. The operation can also be disabled by calling this with a zero pointer.

The µTasker project uses the parameter block to store a single entry and so the following call is used to enter it.

```
fnSetIPv6in4Destinations(
    &temp_pars->temp_parameters.relay_destination[0]);
                    // enter ipv6 in ipv4 table for relaying use
```

10. IPv6 Address Space

It is sometimes useful to step back from the details a little and reflect on the big picture. This section does this by taking a look at the big picture to see that this is indeed really huge!

During registration for a 6in4 tunnel a block of IPv6 addresses was reserved for our use. This block is in fact a sub-block of the range that 'belongs' to the tunnel broker. A look up of any IPv6 address in the block will result in the following range being displayed as belonging to Hurricane Electric, Inc.

```
NetRange:   2001:0470:0000:0000:0000:0000:0000:0000 -
            2001:0470:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF
CIDR:       2001:0470:0000:0000:0000:0000:0000:0000/32
```

This gives us some clues about how it is all being managed. First of all it is clear that this range is a /32 block, which has 2^96 unique IPv6 addresses. That is 7'922'816'251'426'433'759'354'950'336 or just under 8 octillion addresses.

We have also seen that we were allocated a big chunk of 18 quintillion (with option for 64'000x larger block size too), whereby it would be possible for Hurricane Electric to allocate 4'000 million tunnels of the standard size (one tunnel for each possible global IPv4 address).

Whenever a client in the Internet sends a connection request to an IPv6 address in our block it is first routed in the Internet to our tunnel server – this is a catch-all for all addresses in our block (a sub-net of Hurricane Electric's block) and the terminating point for frames in the IPv6 Internet. The tunnel server does the extra work of getting it that little extra distance to us, enclosed in an IPv4 frame as discussed in the sections on 6in4 tunnelling.

The numbers involved are however not always that easy to appreciate. They are big, that is for sure, but just how big they are may best be appreciated when one starts wondering how many servers one could set up in their basement to make use of the fact that one has access to so many unique IPv6 addresses. Some people may be disappointed that not all IPv6 addresses are actually available since there are indeed some groups set aside for special functions and others for future applications, but, even if only 1% were actually allocated it would allow the surface of the world to be literally covered with servers. Calculations show that on each square µm space in your basement (and every other µm^2 of the world's surface) you could install servers with unique IPv6 addresses. In fact this is not one server for each µm^2 of space but about 6 billion servers per square µm of space. That may help to clarify just what we are dealing with.

11. IPv6 Addresses

Whereas a device has only one global IPv4 address for its Ethernet interface, IPv6 devices can have multiple addresses. Several unicast addresses are defined in RFC 3513 as follows:

`0:0:0:0:0:0:0:0 [::/128]` is the unspecified address and must never be used

`0:0:0:0:0:0:0:1 [::1/128]` is the loopback address (this is not used in the µTasker project)

`fe80:x:x:x:x:x:x:x [fe80::/10]` is the link-local address which every node has and is used only in the local network and doesn't normally have to be configured. It is usually derived from the MAC address of the Ethernet interface (this is the technique used in the µTasker project) by generating the address as follows:

> It is assumed that every network card has a unique MAC-48 address and so this address is used as a part of the link-local address so that each device in a local network will also have a unique link-local address.
>
> The MAC-48 is made up of two parts; the first 24 bits are the OUI (Organizationally Unique Identifier); the following 24 bits are the vendor supplied unique part.
>
> A modified EUI-64 format for the lower 64 bits is used whereby the OUI of the MAC-48 are used as the first 24 most significant bits (and the 7th bit is set to '1' because this is used in the MAC address to specify that it is locally administered) and the vendor supplied unique part is used as least significant bits. In between the two the address values are padded with `b1111111111111110`.
>
> For example, a node with MAC-`48 00-11-22-33-44-55` will generate the link-local IPv6 address `fe80::0211:22ff:fe33:4455`.
>
> It is to be noted that IPv6 uses ICMPv6 neighbour discovery to find the node, which is similar to ARP as used by IPv4. This however uses Ethernet multicast rather than Ethernet board cast meaning that the IPv6 capable node needs to work with Ethernet multicast reception in which case the destination MAC address is `33:33:ff:xx:xx:xx` and the values `xx:xx:xx` are the last three bytes of the IPv6 address.

12. Operating a TCP Server over IPv6

The HTTP server is a popular TCP server and is used in the following discussion as reference, although any server on any TCP port is equivalent.

A server listens on its TCP port and will receive connection requests (SYS) from TCP clients. An HTTP server operating together with a dual-stack will be able to accept connections from TCP over IPv4 and TCP over IPv6. The server application usually doesn't need to know which type of connection is being used; it just needs to know on which socket the connection was established on so that it can then handle all communication on this socket appropriately.

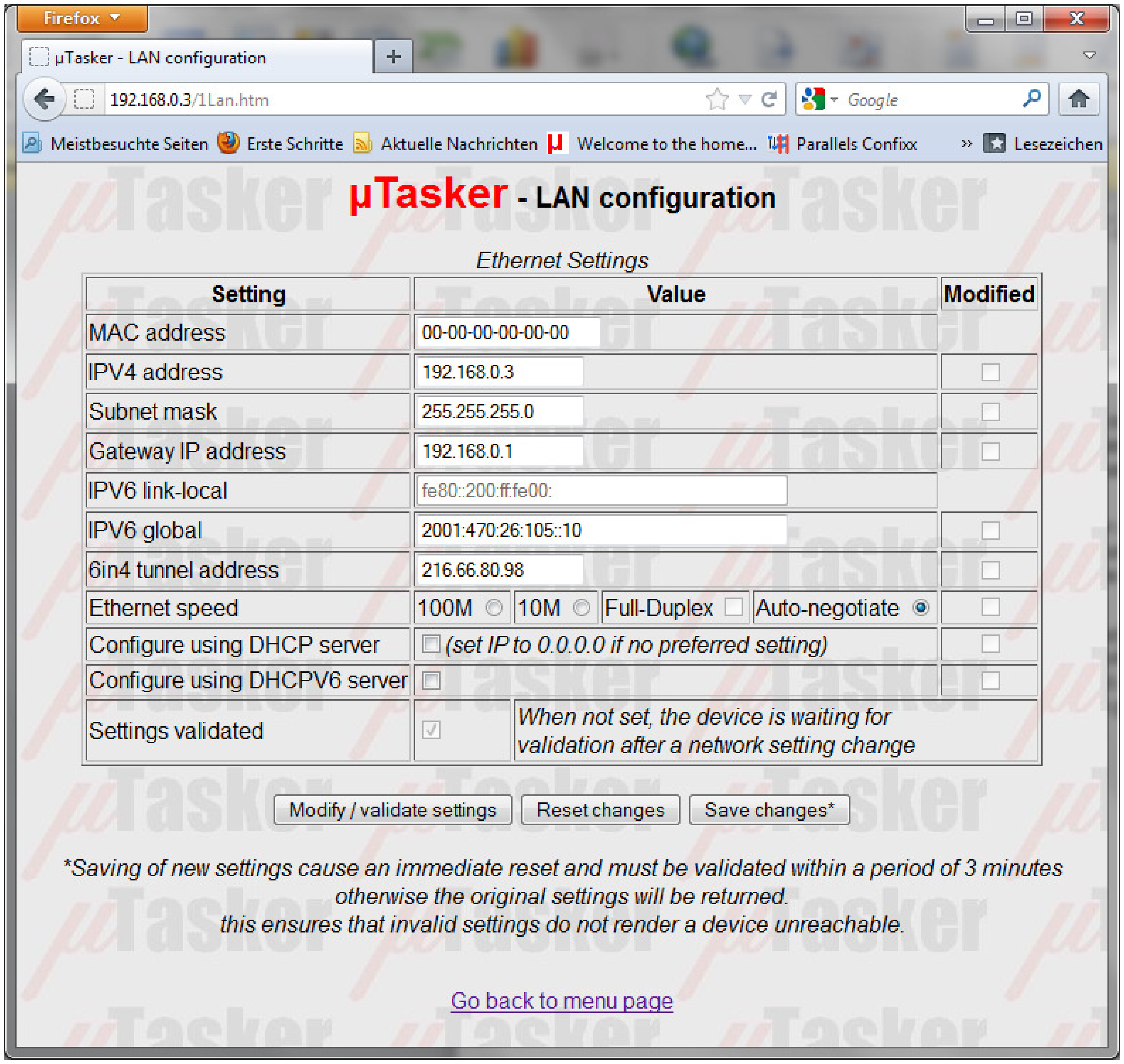
The TCP header and payload can however arrive on top of the following three header formats:

- Ethernet II plus IPv4 header
- Ethernet II plus IPv6 header
- Ethernet II plus IPv4 plus IPv6 headers (4in6 tunnelling when the IPv4 header is of type 0x41)

The Ethernet reception parses the headers and delivers only the TCP and payload to the TCP protocol layer. The socket however notes which headers were used by the connection so that it also knows how to return protocol data or payload data which the TCP layer or the HTTP application sends back.

The following shows examples of testing an HTTP server using a web browser on its IPv6 link-local address and its global IPv6 address with 6in4 tunnelling enabled. If tunnelling is not enabled the global IPv6 operation will be equivalent to the link-local case, which uses pure IPv6.

Since IPv6 addresses are long and this example uses such addresses directly it is useful to first establish a connection with the web server on its IPv4 address. The µTasker web server tends to have a LAN configuration page that can be used as source for the IPv6 addresses. The following diagram shows a typical LAN configuration page and the IPv6 addresses being used:



µTasker - LAN configuration

Ethernet Settings

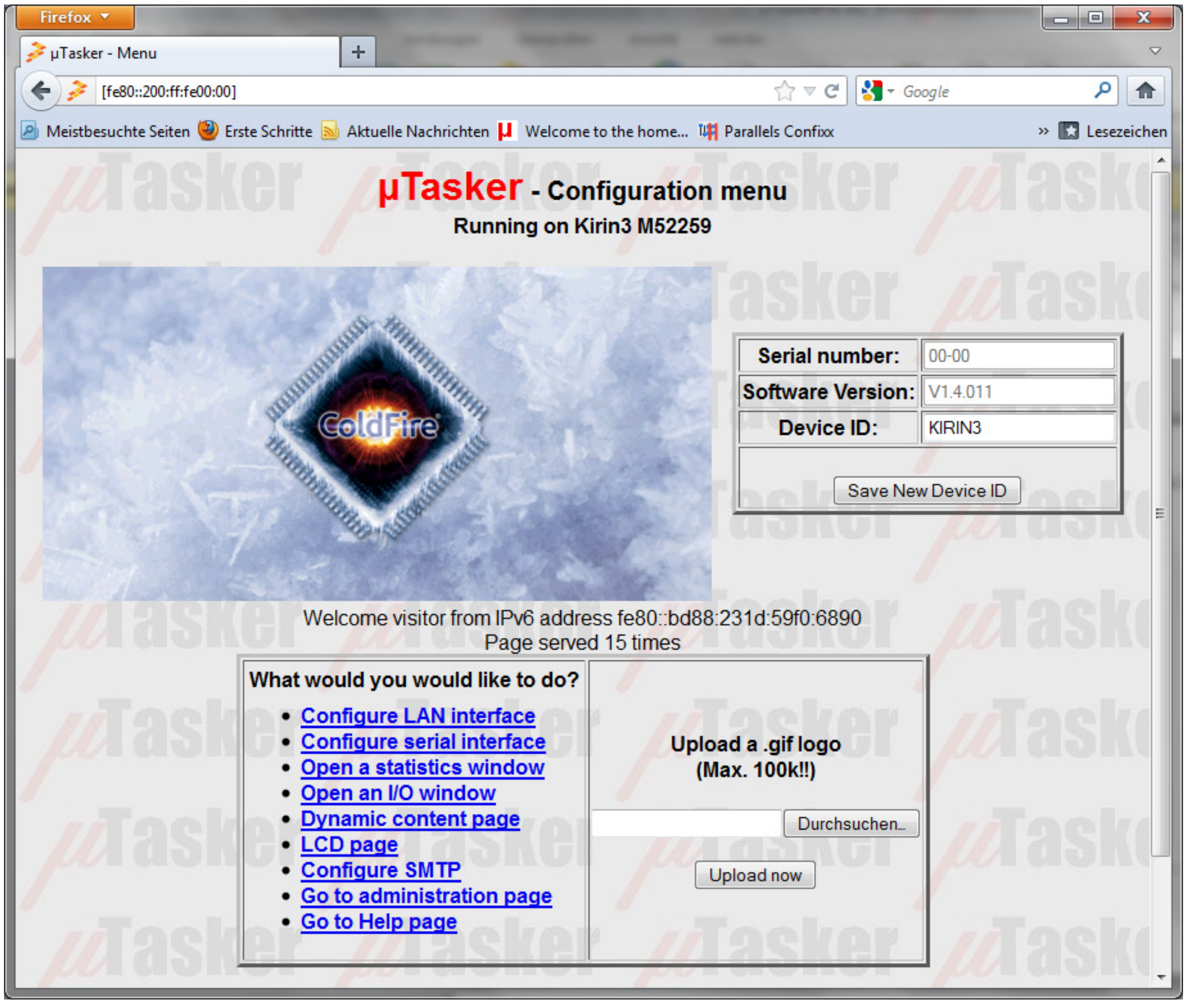
Setting	Value				Modified
MAC address	00-00-00-00-00-00				
IPV4 address	192.168.0.3				☐
Subnet mask	255.255.255.0				☐
Gateway IP address	192.168.0.1				☐
IPV6 link-local	fe80::200:ff:fe00:				
IPV6 global	2001:470:26:105::10				☐
6in4 tunnel address	216.66.80.98				☐
Ethernet speed	100M ○	10M ○	Full-Duplex ☐	Auto-negotiate ●	☐
Configure using DHCP server	☐ *(set IP to 0.0.0.0 if no preferred setting)*				☐
Configure using DHCPV6 server	☐				☐
Settings validated	✓	*When not set, the device is waiting for validation after a network setting change*			

**Saving of new settings cause an immediate reset and must be validated within a period of 3 minutes otherwise the original settings will be returned. this ensures that invalid settings do not render a device unreachable.*

Go back to menu page

To establish a connection to the link-local address (pure IPv6 connection in the local network) the IPv6 address can be entered in the browser URL field using square brackets:

```
http://[fe80::200:ff:fe00:00]
```



µTasker - Configuration menu

Running on Kirin3 M52259

Serial number:	00-00
Software Version:	V1.4.011
Device ID:	KIRIN3

Welcome visitor from IPv6 address fe80::bd88:231d:59f0:6890
Page served 15 times

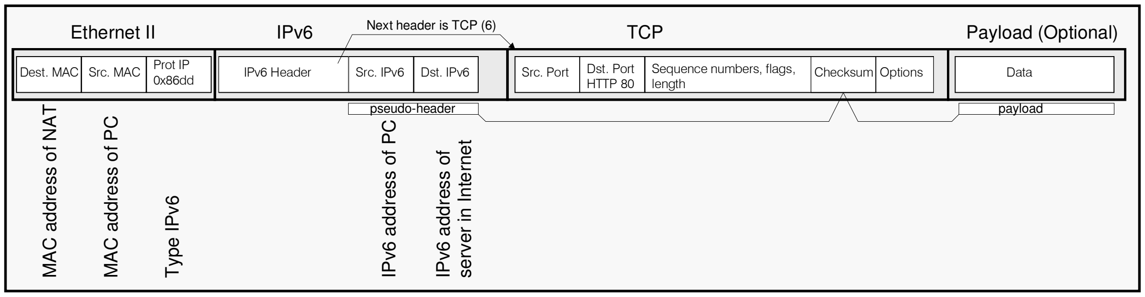
What would you would like to do?

- Configure LAN interface
- Configure serial interface
- Open a statistics window
- Open an I/O window
- Dynamic content page
- LCD page
- Configure SMTP
- Go to administration page
- Go to Help page

Upload a .gif logo (Max. 100k!!)

Notice that the web interface can distinguish whether the connection is via IPv4 or IPv6 and so displays the IPv6 address of the client rather than its IPv4 address. The clients IPv6 address is also a link-local address in this case.

This case is also very simple for the dual-stack receiver to handle due to the fact that the Ethernet type indicates directly IPv6 (0x86dd):



An important detail is the fact that the TCP checksum is calculated over the payload data plus a pseudo-header, made up of the IPv6 source and destination addresses. In the case of IPv4 the pseudo-header used are the shorter IPv4 source and destination addresses. This shows that the TCP layer needs to know the length of the IP addresses when receiving so that it can perform the correct reception check. When transmitting data it also needs to know the IP addresses and their lengths so that the correct transmission checksum can be inserted.

Note that the MAC addresses used in the local network are resolved using ICMPv6 neighbour advertisement. This takes place when the IPv6 address of the destination is not yet in the neighbour cache and has an equivalent function to ARP as used by IPv4.

The stack also knows that all nodes are local due to the fact that the highest 64 bits of the IPv6 addresses correspond to the link-local mask `fe80::/64`.

The HTTP server thus sends responses directly to the local client using IPv6 frames.

The second example shows the operation when the HTTP server is contacted on the global IPv6 address. This shows 6in4 tunnel in operation and the data passing via a tunnel broker in the Internet. For details of the frame contents see the section about 6in4 relay agents.

This particular case shows the local web browser connecting with the web server on the global IPv6 address, even though the web server is local. The device running the web server is also acting as a relay agent to the PC running the web browser. The frame s sent between nodes can be explained as follows:

A – since the web browser wants to contact an IPv6 address which is not local to the network it will use pack the IPv6 header and TCP data (in the case of a connection request this will be a simple TCP SYS) into a IPv4 frame destined to its tunnel in the IPv4 internal (`216.66.42.2`). The first frame is passed to the gateway so that it can be sent into the IPv4 Internet.

B – The gateway sends the frame into the IPv4 Internet where it is destined for the tunnel address (`216.66.42.2`).

C – The tunnel will extract the IPv6 part of the frame and send it into the IPv6 Internet. Since, however, the destination belongs to its own sub-net it will tunnel it back.

D – The frame is passed back to the IPv4 address of the gateway.

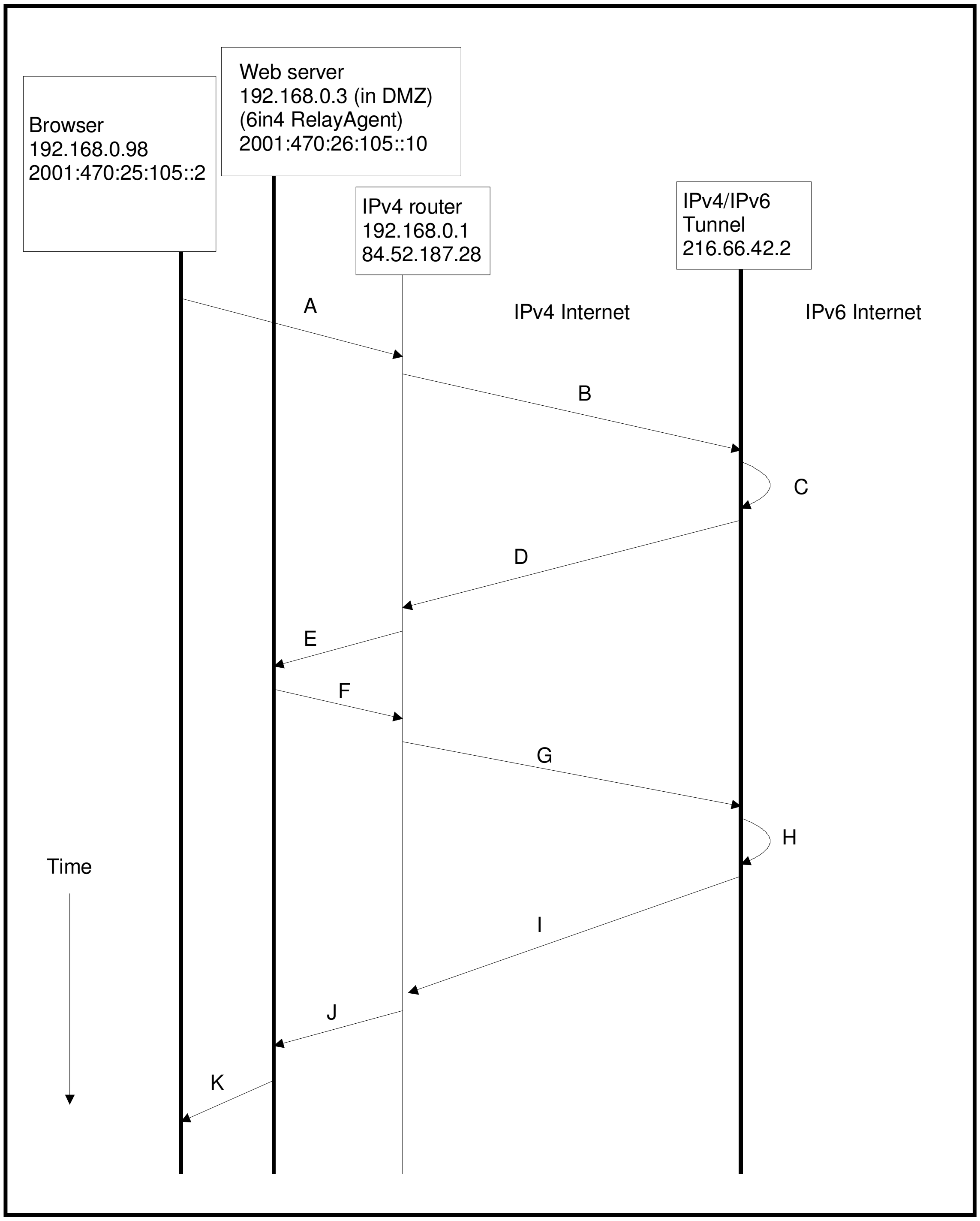
E – Since the IPv4 gateway doesn’t understand 6in4 protocol it passes the frame to the local IPv4 address 192.168.0.3 because this is configured to catch all unknown data (DMZ). The device understands 6in4 protocol and so sees that its own global IPv6 address is the Ipv6 destination address. It thus passes the TCP protocol and any data that it is carrying to the web server on TCP port 80

F –.The web server either accepts the connection (or returns data if this is a part of the communication). The IPv6 response is packed into an IPv4 tunnelled frame and returned to the IPv4 address where it arrived from (that is the tunnel). Again the frame is sent to the gateway since the address is not in the local network.

G, H, I – The following three steps are equivalent to B,C,D

J – The local address 192.168-0.3 is again used to catch unknown protocol frames but this time the device sees that the IPv6 destination address is not its own global IPv6 address. Since relaying is enabled it identifies that it know the destination and so relays it to the local IPv4 address 192.168.0.98

K – The frame is passed to the PC running the web browser so that its tunnel can extract the IPv6 frame and deliver it to its web browser.



Note that the web browser an web server are located in the same network in this example. If they are at different locations and their IPv6 addresses not in the same tunnel sub-net the IPv6 would deliver the IPv6 frames accordingly.

13. Operating a TCP Client over IPv6

The main difference between server and client operation is that the server listens for connections from a client and a client is active in establishing the TCP connection.

In the case of the server operation, as described in the previous section, the TCP layer receives information about whether the TCP connection was established using IPv6 or not – all subsequent transmission uses the corresponding IPv6/IPv4 mode.

Once a client is connected all transmission via its connected TCP socket will be handled according to the IPv6/IPv4 mode flag used by the socket in the same manner as the server.

The client, or the application using the client mode of operation, must therefore be able to choose whether the connection is to be made over IPv4 or IPv6 each time a connection is established. For the sake of compatibility any existing client code establishes an IPv4 connection by default, in which case the function

```
extern USOCKET fnTCP_Connect(USOCKET TCP_socket,
                             unsigned char *RemoteIP,
                             unsigned short usRemotePort,
                             unsigned short usOurPort,
                             unsigned long usMaxWindow_flags);
```

is used.

The final parameter (`usMaxWindow_flags`) is extended when IPv6 is enabled (otherwise it is `unsigned short`) to allow passing a maximum windows length (maximum 65’535 bytes) plus TCP connection flags. When the flag `TCP_CONNECT_IPV6` is added to this parameter (`0x00010000`) the connection takes places as IPv6 connection rather than IPv4 connection, whereby the `RemoteIP` must also be a pointer to an IPv6 address.

An FTP data connection operating in passive mode is an example of a TCP client. A TCP server may thus also require client capabilities as is pointed out in the following section discussion extensions required for IPv6 operation.

14. Extensions for IPv6

It was noted in the previous section that TCP in a dual-stack solution needs to be aware of the IP type that it is operating over due to the fact that it needs to calculate check sums including a pseudo-header made up of either the IPv4 or IPv6 IP addresses in the corresponding IP header.

However also higher layer protocols may require some awareness to be able to operate correctly.

In the case of HTTP a feature can be made of this awareness by displaying different web content depending on whether the user is connecting over IPv4 or IPv6 – *remember the fact that the Kane turtle only dances if you do connect via IPv6*. Also the µTasker project displays either the IPv4 or IPv6 address of the user on its start page. The application can identify the type of TCP connection used by reception data from the state of the following flag:

```
if (present_tcp->ucTCPInternalFlags & TCP_OVER_IPV6) {
    // data received over IPv6 connection
}
```

HTTP is otherwise compatible whether it is operating over IPv4 or over IPv6.

TELNET servers don't need to be aware of the type of IP used – their operation is fully compatible.

FTP servers however do need to be aware of the IP type used due to the fact that the data connection needs to be established with either an IP server/client (depending on whether the mode is active or passive). RFC2428 details two new commands that need to be supported by the FTP server when IPv6 operation is to be used:

- `EPRT` – this is the extended port which is to be used for the data connection when operating in active mode (extended PORT command). It informs of the IPv6 address and port number. *The active data connection is an example of client mode operation and so uses the technique explained in the previous section*.
- `EPSV` – this is the command to use passive mode for data transfers (extended PASV command). The server responds to this request for a port number by returning the 229 response and the port number in a format as specified by the RFC.

15. Conclusion

This document has discussed practical method to make use of IPv6 capabilities using the TCP/IP Dual stack in the µTasker project.

It has concentrated on using 6in4 tunnelling as a practical method of enabling operation in legacy IPv4 networks since it is expected that these will still be predominant in the years to come. The concept of the relay agent was introduced and examples given of operating a web server on a global IPv6 address even when this needs to be behind a standard IPv6 gateway.

These techniques allows immediate benefits of IPv6 technology to be utilised, whereby it is theoretically possible to operate more IPv6 devices with globally unique IPv6 addresses behind a single IPv6 gateway than there are IPv4 addresses in the World Wide Web (in fact many millions times more!!).

Modifications:
- V0.01 7.5.2010 – provisional version for the documentation page
- V0.02 11.5.2010 – added IPv6 tunnelling and 6in4 operation description
- V0.03 18.5.2010 – added tunnelling broker and tunnelling tests
- V0.04 20.5.2010 – added figure 7-1 and IPv6 address space section
- V0.05 30.5.2010 – added introduction to 6in4 relay agent
- V0.06 08.03.2012 – added appendix with IPv6 transmission notes. Added section on IPv6 addresses
- V1.00 31.03.2012 – first release version including IPv6 Web Server operating example
- V1.01 02.04.2012 – added IPv6 client operation and extensions for FTP server

Appendix A – IPv6 Transmission Software Details

This appendix discusses the impact of IPv6 and tunnelling on the software operation in the µTasker project.

As reference the IPv4 case is first considered, whereby it is important to understand that the operation was designed to accommodate two distinct modes of TCP operation. The first is the buffered TCP mode which is described in the TELNET document http://www.utasker.com/docs/uTasker/uTaskerTELNET.PDF This is the classic TCP/application approach where the application passes data to the TCP socket without needing to know any details about its operating details. The TCP socket buffers data until it can be sent, handles repetitions if data is lost, and sends the payload in chunks. This mode is shown in figure A-1. Although this represents the simplest operation for the application it requires an intermediate buffer and is therefore not the most efficient approach in terms of memory requirements considering that each possible socket needs its own buffer.

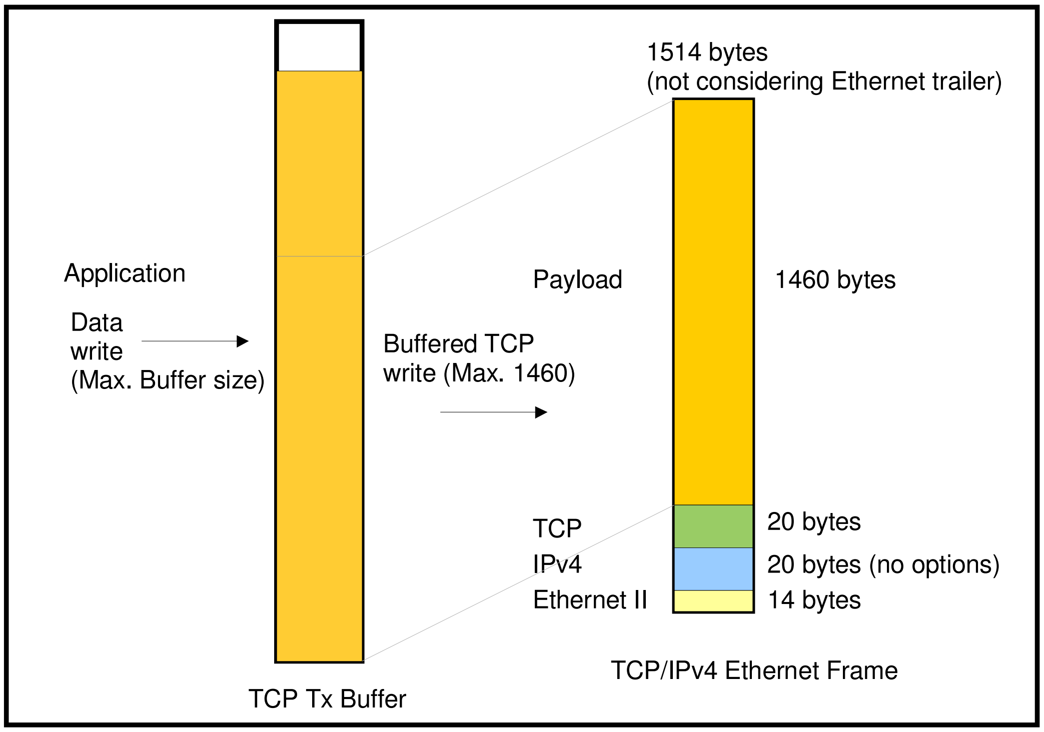


Figure A-1 Buffered TCP TCP/IPv4 Ethernet Frame

Figure A-2 shows the simple socket approach where the application has to work together with the TCP layer. It is used by the µTasker HTTP server to allow data to be served efficiently in multiple socket environments (supporting limited TCP windowing to achieve fast throughput as well as techniques to generate content as used for displaying variables and generating dynamic content) without the need for large buffers. Even processors with very limited RAM can utilise this technique to serve multiple users at high speed.

Since the µTasker concentrates on techniques to optimise capabilities with minimum memory footprint the second method is a very important one which can achieve important advantages in small footprint environments.

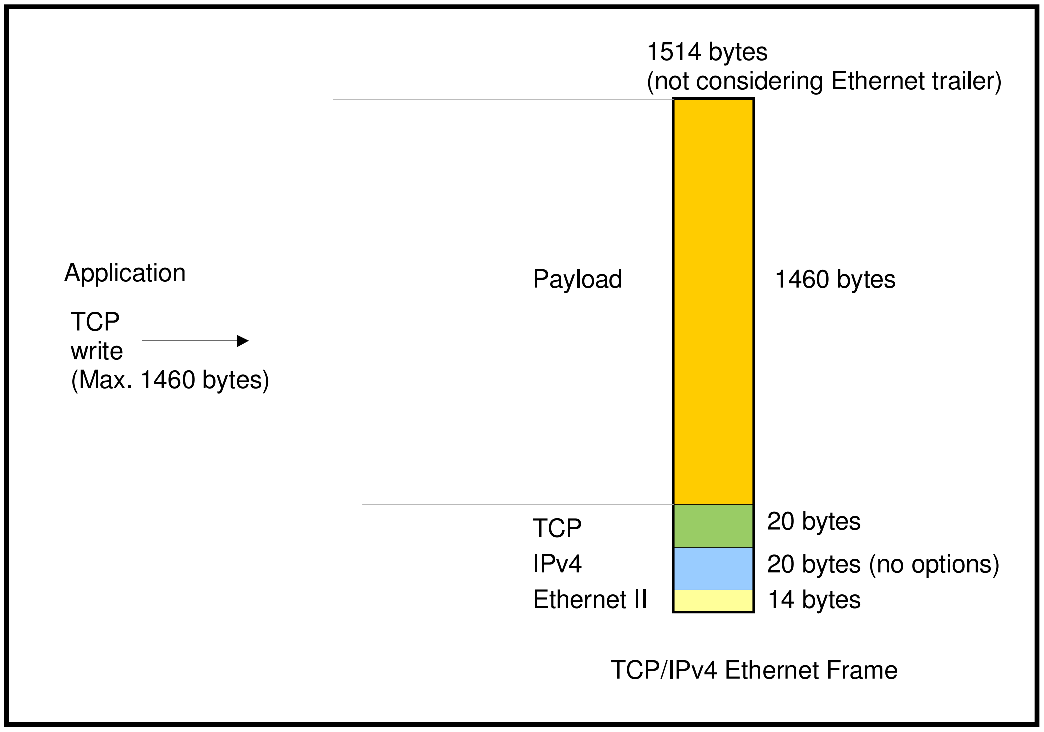


Figure A-2 Simple-Socket TCP TCP/IPv4 Ethernet Frame

Figures A-3 and A-4 show the effect of IPv6 without and with tunnelling.

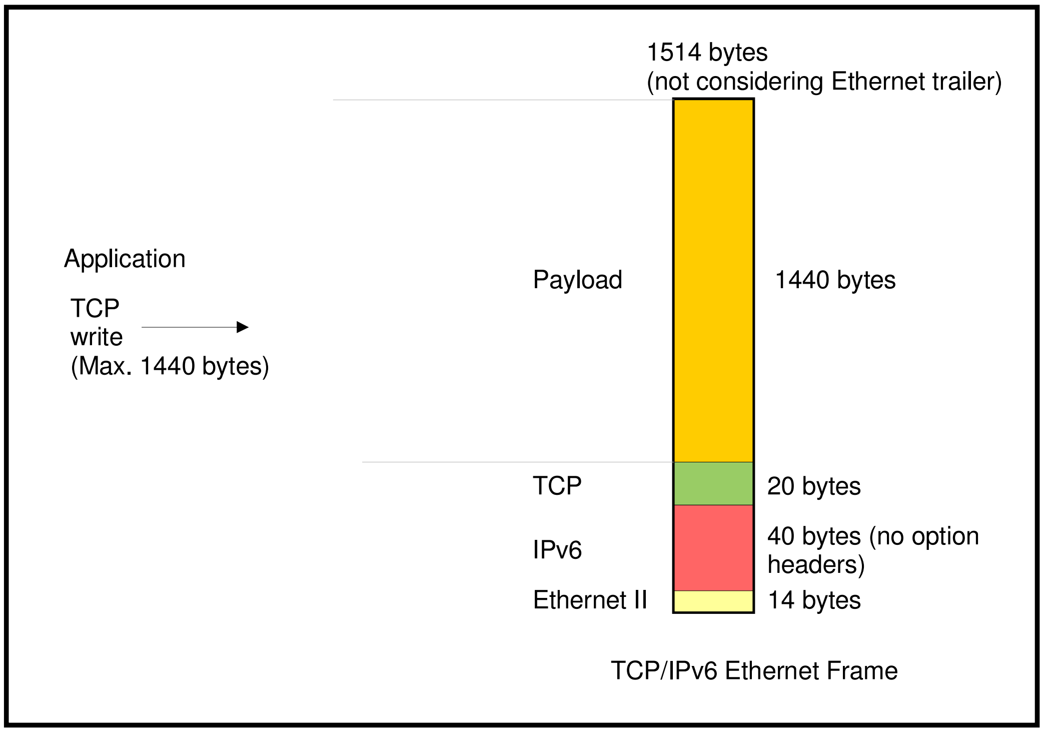


Figure A-3 TCP TCP/IPv6 Ethernet Frame (with no option headers)

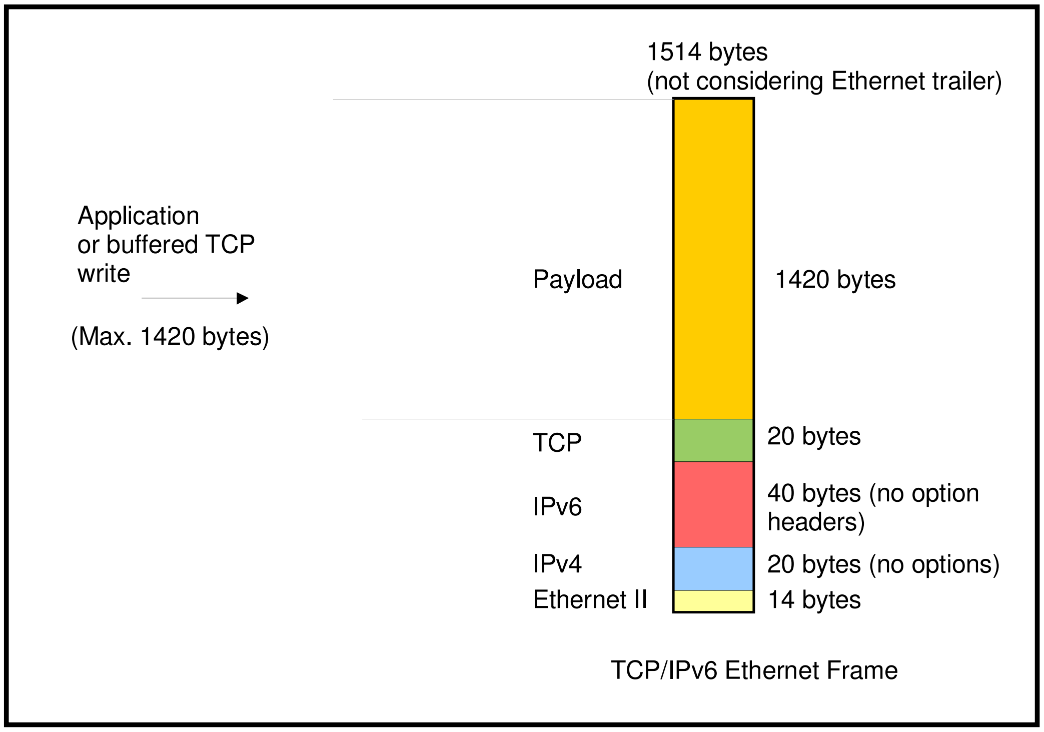


Figure A-4 TCP TCP/IPv6 6in4 Ethernet Frame (with no option headers)

The following details become clear:

- The single IPv6 IP header causes the IP part of the frame to increase by 20 bytes in comparison to theIPv4 case with no IP options. The amount of payload that can be carried in a single Ethernet frame thus decreases by 20 bytes.
- When 6to4 tunnelling is in use the size of the payload is also reduced by a further 20 bytes by IPv4 header size.
- When working in buffered TCP mode the application doesn't need to be aware of whether IPv4, IPv6 or 6to4 tunnelling is in operation since the TCP socket adds as much payload as possible to each transmitted TCP frame; the application simply passes data to the buffer.
- When simple socket mode of operation is used the application (including HTTP and other µTasker protocol implementations that use this method) need to know how much payload space is available so that the correct maximum amount can be passed. This is especially important to the HTTP server which may be generating content; if less payload can be sent than content was generated it would cause difficulties with such content generation techniques.
- A simple solution is to generally decrease the maximum payload size from 1460 bytes to 1420 bytes so that the limit is respected in all cases. Since this size reduction represents only a small impact this technique is reasonable in the simplest case.

IPv4 headers are shown without options which can cause it to increase from 20 bytes to a possible maximum of 60 bytes in size. These options are however very rarely used and not all are supported by routers. Since they are practically redundant they have never been supported by the µTasker project.

IPv6 headers can contain optional headers. These optional headers include authentication and encryption headers and are thus considered as important parts of IPv6. If optional headers are to be supported it is thus seen that the payload decreases by their respective size and it is also seen that the application using simple sockets must know the maximum payload size each time that it writes data because this may change from one packet to another.

At the time of writing activation of IPv6 requires the maximum payload size to be restricted to maximum 1440 bytes (from maximum 1460 with IPv4). The activation of 6in4 tunnelling requires the reduction to the maximum to 1420 bytes. This doesn't require the application to be informed of the maximum size. IPv6 option headers are not supported. Once IPv6 option header support becomes necessary, a method to dynamically adjust the maximum payload size by the application interface will also be added.

The application also doesn't need to know whether it is working with an IPv4, IPv6 or tunnelled IPv6 socket connection. This information is contained at the socket layer so that it knows whether to send the payload and TCP to the IPv4 or IPv6 transmission routine. In case of 6in4 tunnelling the IPv6 transmission routine will also automatically call the IPv4 transmission.