IPv6 Operations
Internet-Draft
Expires: January 5, 2005

J. Massar
Unfix / SixXS

July 7, 2004

AYIYA: Anything In Anything
draft-massar-v6ops-ayiya-02

Status of this Memo

Copyright Notice

Abstract

This document defines a tunneling protocol that can be encapsulated in any other protocol. This protocol uses authentication tokens, allowing multiple identities to exist on the same endpoint and thus also to create tunnels from/to the same NAT and making it possible to automatically change the endpoint of the tunnel on the fly. This protocol is intended as an alternative to the proto-41 protocol in use for tunneling IPv6 over IPv4 packets over the Internet but can also be applied in multihoming and mobility solutions. Due to the authentication this protocol is especially useful for dynamic non-24/7 endnodes which are located behind NATs and want to use an IPv6

Tunnel Broker, for instance. The protocol can carry any payload and thus is not limited to only IPv6 over IPv4 but can also be used for IPv4 over IPv6 and many other combinations of protocols.

Table of Contents

1. Requirements notation 3
2. Introduction 3
3. AYIYA Packet Format 4
3.1 Identity Length (IDLen) 5
3.2 Identity Type (IDType) 5
3.3 Signature Length (SigLen) 6
3.4 Hashing Method (HshMeth) 6
3.5 Authentication Method (AutMeth) 7
3.6 Operation Code (OpCode) 7
3.6.1 No Operation 7
3.6.2 Forward 7
3.6.3 Echo's 8
3.6.4 MOTD 8
3.6.5 Queries 9
3.7 Next Header 11
3.8 Epoch Time 12
4. AYIYA Heartbeat 14
5. Signing the packet 14
5.1 Hashing the packet 15
5.2 Signing with a Shared Secret 15
5.3 Signing with a Public/Private Key 15
6. Identity information in DNS 16
7. Acknowledgements 16
8. Security Considerations 16
9. Scenarios 18
9.1 Using AYIYA for IPv6 Tunnel Brokers 18
9.2 Tunneling to multiple endhosts behind a NAT 19
9.3 Multihoming using AYIYA 20
9.4 Mobility using AYIYA 23
10. IANA Considerations 23
11. References 23
Author's Address 25
Intellectual Property and Copyright Statements 26

1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

Many users are currently located behind NATs which prohibit the usage of proto-41 IPv6 in IPv4 tunnels [RFC3056] unless they manually reconfigure their NAT setup which in some cases is impossible as the NAT cannot be configured to forward proto-41 ([RFC1933]) to a specific host. There might also be cases when multiple endpoints are behind the same NAT, when multiple NATs are used or when the user has no control at all over the NAT setup. This is an undesired situation as it limits the deployment of IPv6 [RFC3513], which was meant to solve the problem of the disturbance in end to end communications caused by NATs, which where created because of limited address space in the first place.

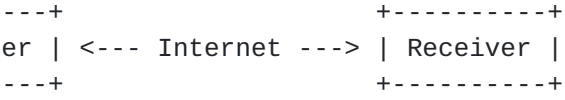
This problem can be solved easily by tunneling the IPv6 packets over either UDP [RFC0768], TCP [RFC0793] or even SCTP [RFC2960]. Taking into consideration that multiple separate endpoints could be behind the same NAT and/or that the public endpoint can change on the fly, there is also a need to identify the endpoint that certain packets are coming from and endpoints need to be able to change e.g. source addresses of the transporting protocol on the fly while still being identifiable as the same endpoint. The protocol described in this document is independent of the transport and payload's protocol. An example could be IPv6-in-UDP-in-IPv4, which is a typical setup that can be used by IPv6 Tunnel Brokers [RFC3053].

This document does not describe how to determine the identity, signature type or the inner and outer protocols. These should be negotiated manually or automatically by e.g. using TSP or a relevant protocol which is capable of describing the configuration parameters of AYIYA tunnels. Separate documents for the configuration protocols supporting AYIYA should include the details on how this is done.

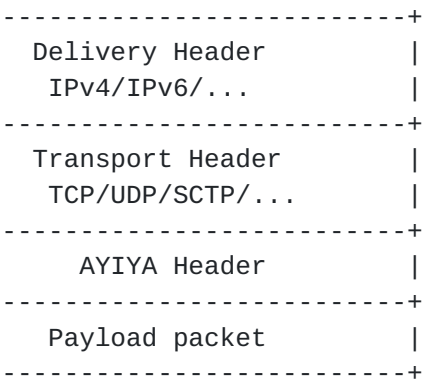
Additionally this document describes how AYIYA could be used in both a multihoming and in a mobility scenario.

3. AYIYA Packet Format

The AYIYA protocol is put inside the payload of another protocol. This can be either UDP [RFC0768], TCP [RFC0793] or SCTP [RFC2960] which are the currently defined transport protocols, future transport protocols could also be used. The transport protocol can be run over both IPv4 or IPv6 or any other future protocol. AYIYA can also be directly put in the payload of IPv4 or IPv6, just like UDP, TCP or SCTP. This allows tunneling of IP over IP with a minimum overhead. When deciding in which protocol to encapsulate AYIYA one must keep into thought that some firewalls and/or NAT gateways won't work with certain combinations. Schematically, this will look like the following diagram.

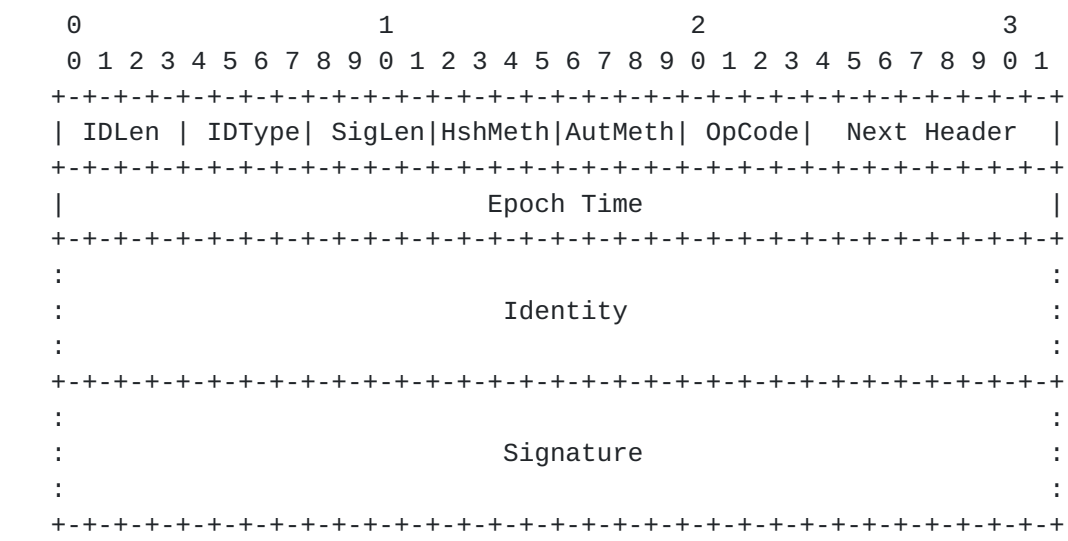


A complete on the wire packet will typically have the format depicted in the following diagram.



In the case where one encapsulates AYIYA directly inside IP the Transport Header won't be present.

The AYIYA protocol header has the following format.



All fields are in network byte order (Big Endian). The base AYIYA header without an identity or signature is 8 bytes.

3.1 Identity Length (IDLen)

The IDLen (Identity Length) Field defines the length of the Identity Field using a power of 2 in octets.

e.g. an IDLen of 4 is 2^4 = 16 bytes.

3.2 Identity Type (IDType)

The Identity Type specifies what kind of Identity is included in the header. The Identity field is used by the receiver to determine which sender sent the packet, this is done as there is an assumption that the source endpoint, the source IP address or the source port, may change arbitrarily which will be the case when the sender is behind a NAT, using DHCP, PPP or using IPv6 privacy extensions [RFC3041] and thus has a changing address. Even though the endpoint is expected to change, the Identity will remain the same unless negotiated otherwise. Currently defined Identity types are:

- 0x0 None
- 0x1 Integer
- 0x2 ASCII string

Types 0x3 till 0xf are reserved for future usage.

The type of identity used by an AYIYA tunnel is negotiated either manually or automatically outside of this protocol, these fields are

included to allow verification of the type and also to allow multiple types to be used by one receiver at the same time. ASCII strings are NULL padded when they do not fill the complete identity field. The types are multifunctional, e.g. type 0x01 could contain an IPv4 address when the length of the identity is 0x2 or could contain an IPv6 address when the length is 0x4. A string could contain DNS names. The exact content of the Identity Field is defined by the users of this protocol and are out of scope of this document.

If the Identity Type is None, the Identity Field is absent from the packet. The Identity Length Field must be 0 in this case. The Signature Field, if present, will then directly follow the Epoch Time Field. In case the Signature Field is not present the payload will directly follow the Epoch Time field.

3.3 Signature Length (SigLen)

The SigLen describes the length of the hash and is specified in octets divided by four. e.g. a SigLen of 3 means the signature is 12 bytes long, a SigLen of 15 means that the Signature is 60 bytes long. The protocol thus allows for a maximum signature of 480 bits.

3.4 Hashing Method (HshMeth)

The HshMeth (Hashing Method) bits describe the type of the Hashing Method used to create the signature of the packet. By hashing the complete packet we can verify that the packet did not change during transit between sender and receiver. Currently defined Hashing Methods are:

- 0x0 No hash
- 0x1 MD5
- 0x2 SHA1

As there are known collisions for MD5 [RFC1321] it is advised to use SHA1 [RFC3174] as a default Hashing Method. Hashing Methods 0x3 to 0xf are reserved for future usage.

When the Hashing Method is 0x0, AuthMeth and SigLen MUST also be set to 0 and the packet doesn't include a Signature, the payload, defined by the Next Header, then directly follows the Identity field, which may also be absent depending on its type.

3.5 Authentication Method (AutMeth)

AuthMeth (Authentication Method) describes the type of the Authentication Method. By authenticating the packet we can verify that the sender really originated from the sender, of course assuming that the Authentication Method has not been compromised. The AYIYA protocol doesn't have any options for encryption. Encryption can be done in the payload. The currently defined Authentication Methods are:

- 0x0 No authentication
- 0x1 Hash using a Shared Secret
- 0x2 Hash using a public/private key method

Authentication Methods 0x3 to 0xf are reserved for future usage.

In the case where an implementation does not support or expect the received Identity or Signature Type (e.g. because it was configured for a different type) it MUST silently discard the packet. The user may be notified of this event.

3.6 Operation Code (OpCode)

The Operation Code can request special operation on the packet. Currently the following Operation Codes are defined.

- 0x0 No Operation / Heartbeat
- 0x1 Forward
- 0x2 Echo Request
- 0x3 Echo Request and Forward
- 0x4 Echo Response
- 0x5 MOTD
- 0x6 Query Request
- 0x7 Query Response

Values 0x8 till 0xf are reserved for future extensions.

3.6.1 No Operation

The No Operation OpCode allows the packet to be used for updating the latest received time, see the Heartbeat section explaining the rationale why this packet is also dubbed a Heartbeat Packet. The Next Header field must be set to 59 (No Next Header).

3.6.2 Forward

The Forward OpCode specifies that this is a normal packet of which the payload is to be forwarded.

3.6.3 Echo's

The "Echo Request" OpCode requests that the payload is echoed back to the sender, the OpCode of the returned packet must then be set to "Echo Response". The payload of the Echo Request packet MUST NEVER be forwarded. When the OpCode is set to "Echo Request and Forward" then the packet must be echoed back to the sender and also forwarded as a normal packet. This allows the Heartbeat functionality, as discussed in the next chapter, to be integrated into the normal packet stream. It can also be used to ensure delivery of the packet to the other end of the tunnel. An Echo operation causes the payload to be returned unaltered. The Next Header field must be set to 59 (No Next Header) for both the Echo and Echo Response Operation Codes. When the OpCode is "Echo Request and Forward" the Next Header field must be set to the value describing the payload to be forwarded.

3.6.4 MOTD

The MOTD (Message Of The Day) OpCode allows the sender to send a message to the receiver. The receiver SHOULD then display this message to the user. This facility can be used to notify the user of software updates, changes in infrastructure and many more events. Typically a server side of the AYIYA connection will send its MOTD to the receiver when it connects and the server has not seen the client for some time. The Next Header field must be set to 59 (No Next Header). The payload of the MOTD Operation Code consists of a multiline ASCII message in the following format as per ABNF [RFC2234].

```
message   = epochtime LF title LF [URL] LF [text]

epochtime = *DIGIT

title     = *( %x2D | %x2E | %x30-39 | %x41-5A | %x61-7A )

URL       = [ "http://" | "https://" ] *(%x23 | %x25 | %x26 |
            %x2B | %x2E-3A | %x3F-5A | %x61-7A)

text      = *( LF | %x20-7E )
```

The epochtime, the time in seconds since "00:00:00 1970-01-01 UTC", specifies the date of this message, when the server sends the same message it should also send the same epochtime, this allows the receiver to ignore the message when the user has already read it when it is sent more than once. The string of which epochtime is made up corresponds to the output of the UNIX "date +%s" command. A client could convert the epochtime to a human readable string and present that to the user. Title should contain a short descriptive title of

the content of the message. URL can contain a URL to further or even all information, keeping text empty. Text can contain the message directly. At least URL or text must contain content, both may also be specified. The precedence is then given to the message in text with additional information to be found on URL.

3.6.5 Queries

The Query and Query Response Operation Codes allow the sender to ask a question to the receiver. The Next Header field is set to 59 (No Next Header). The payload consists of a 4 byte 32bit integer in network byte order which can be used as sequence number the client SHOULD change this number every query, the receiver will respond with the same sequence number so that the original sender can distinguish between the queries it sent. The rest of the payload contains the answer to the question the sender asked in ASCII. The length of the answer depends on the payload length. In cases where the receiver doesn't want to or can't answer it should return only the sequence number, the original sender can find out why this occurred by querying for the error code. Queries may be given in lower, upper or mixed case, they must be treated the same. The query functionality can amongst others be used for retrieving diagnostic information during troubleshooting situations. The following sections define the queries that are defined.

3.6.5.1 contact

The "contact" query requests the receiver to respond with contact information. The response could be a name of a person, a phone number, email address, NIC handle etc. Multiple lines can be used to specify the information. No format is defined to avoid automatic parsing.

3.6.5.2 hostname

The "hostname" query requests that the receiver respond with the hostname it perceives to have. This information can be used to verify when an endpoint changes its endpoint a lot if maybe there are not multiple hosts using the same authentication information. The side that notices the many changes could then issue a hostname query and compare the results and notify the users using a MOTD or even send a "shutdown" query to one of the clients.

3.6.5.3 lasterror

The "lasterror" query asks the receiver to return a human readable description of the last error that occurred. The response can consist of multiple lines by inserting ASCII newlines at appropriate places.

```
description = *[ LF | %x20-7E ]
```

3.6.5.4 outer-endpoint

The "outer-endpoint" query requests the receiver to respond with the endpoint address (IPv4 / IPv6) information on a single line of the following format:

```
endpoint = (IPv4address | IPv6address) [ SP (tcp-port | udp-port) ]

tcp-port = port

udp-port = port

port     = *DIGIT
```

e.g.:
2001:db8::1 3740

192.0.2.42

This can be useful to determine if the sender is the same as another host, like in the example case mentioned at the "hostname" query. It can also be useful to determine if the client is behind a NAT even though AYIYA can operate fully across NATs.

The IPv6address and IPv4address ABNF are defined in "APPENDIX B: ABNF Description of Text Representations" of the "IP Version 6 Addressing Architecture" [RFC3513].

3.6.5.5 shutdown

The "shutdown" query requests that the receiver side shuts down the tunnel and stops communicating with the server. The shutdown command MUST be preceded with a MOTD to notify the user with the reason of the shutdown request. The "shutdown" command must always be obeyed. A client thought might retry communications after at least 5 minutes after receiving and having obeyed the shutdown command.

3.6.5.6 version

"version" requests the version information of the client on the receiver side. The response should be in the following format as formatted per ABNF [RFC2234].

```
response       = ayiya-version " " client-version " " os-version

ayiya-version  = "AYIYA/draft-02"

client-version = version

os-version     = version

version        = name [ "/" release ]

name           = chars

release        = chars

chars          = *( %x2D | %x2E | %x30-39 | %x41-5A | %x61-7A )
```

ayiya-versions contains the version of the protocol which the client supports. Currently only draft-02 can be mentioned as draft-00 didn't include OpCodes and draft-01 didn't include the Query OpCode.

client-version contains the version of the client.

os-version contains the Operating System version.

Including the release version is optional for both client-version and os-version. This allows the querier to know partially what is running on the side of the receiver but avoids disclosing too much information which might be seen as an information leakage in some security contexts.

The following, individual lines give example responses:

```
AYIYA/draft-02 SixXS-AICCU/2004.07.04 Wendy/6.66-suus

AYIYA/draft-02 HSKT/2004.07.04 NikiOS

AYIYA/draft-02 SiXXS-AICCU NicoleOS
```

3.7 Next Header

The Next Header, like in IPv6, contains the protocol value of the

payload following the AYIYA Packet Header. There is no length field as that can be deduced from the protocol that is carrying this packet.

When there is no payload to be forwarded, the Next Header field MUST be set to 59 (No Next Header). When the Next Header field is set to No Next Header the payload can not be forwarded even if there is a payload included in the AYIYA packet.

3.8 Epoch Time

Epoch Time is the time in seconds since "00:00:00 1970-01-01 UTC". Both the sender and the receiver are advised to be synchronized using NTP [RFC2030] to make sure that their clocks do not differ too much even after traveling the intermediate networks between the sender and the receiver. The number of seconds since the above date is stored in a 32 bit unsigned integer in network byte order.

The Epoch Time is included to be able to guard against replay attacks. See the Security Considerations section for more details.

The Epoch Time will loop in 2038 when the 32 bit unsigned integer reaches its maximum value. This will cause that the difference between the two times is larger than the advised timeout time even though the difference is not that big. To avoid a service interruption, because the time in the packet is not inside the limits of the clock shift time, every implementation MUST handle times in the range (0-timeout)..0 specially and compensate the loop, e.g. by shifting away from the loop time. Typical C code which handles the verification of the epochtime is included as an example:

```
// epochtime = epochtime as received in the packet
// Don't forget to convert the byteorder using ntohl()
bool ayiya_checktime(time_t epochtime)
{
   // Number of seconds we allow the clock to be off
   #define CLOCK_OFF 120
   int i;

   // Get the current time
   time_t curr_time = time(NULL);

   // Is one of the times in the loop range?
   if ( (curr_time  >= -CLOCK_OFF) ||
        (epochtime >= -CLOCK_OFF))
   {
      // Shift the times out of the loop range
      i = (curr_time + (CLOCK_OFF*2)) -
          (epochtime + (CLOCK_OFF*2));
   }
   else i = curr_time - epochtime;

   // The clock may be faster, thus flip the sign
   if (i < 0) i = -i;

   // Compare the clock offset
   if (i > CLOCK_OFF)
   {
      // Time is off, silently drop the packet
      return false;
   }

   // Time is in the allowed range
   return true;
}
```

Theory for the above: for simplicity let's assume the loop is around 10000. Sender sends an epochtime of 9990, but the receiver's time is at 10 already, thus we apply the shift and the times become: (9990 +

240)%10000 = 230 and (10 + 240)%10000 = 250 the difference !(250 - 230) is 20, which is in the allowed clock_off range. If we didn't apply this compensation the difference would have been !(10 - 9990) = 9980 seconds which would mean the packet would have been dropped even though the time of the packet is valid.

4. AYIYA Heartbeat

As the receiver will disable the tunnel after it has not received a packet from the sender after a configured time the sender should send packets to the other side of the tunnel with the Next Header field set to 59 (No Next Header) but the payload may contain data which is private to the implementation. The implementation could include a sequence number in the payload like is common with ICMP Echo [RFC2463] packets. The receiver will reply on reception of this packet returning the exact payload the sender transmitted allowing the sender to compare the information and deduce latency information and other statistical information from it using the implementation specific data contained in the payload. This packet allows the sender to test the tunnel's functionality. If the signature is not correct, either because of the wrong shared secret, wrong hash, wrong identity or connectivity problems, the sender will not get a reply and could notify the user of this situation.

Senders should send these packets once per 60 seconds as the receiver is usually configured to disable the tunnel after it has received no packets for a timeout time of 120 seconds. An implementation could choose to not send the heartbeat packet when it has already sent a packet in the last 60 seconds thus avoiding a small overhead in transmission and processing of these extra heartbeat packets. Receivers MUST handle every correctly verified packet as the last received one.

A side effect of the Heartbeat Packet is that a NAT will update its mappings and keep the same source/destination ports in cases where AYIYA is encapsulated inside UDP, for instance.

An implementation could choose to not send any heartbeat packets, but this will cause the connectivity, provided by the tunnel, to be interrupted until the sender sends the next packet.

5. Signing the packet

When there is no Hashing there neither is no signing of the packet and the header won't include a Signature.

If a received packet contains flags that the packet contains a hash or that the packet contains an authentication then the receiver must

verify that the signature provided is correct by following the same procedure as taken by the sender and comparing the results of the signatures. When the signatures match the packet can be processed further. When the signatures do not match the receiver MUST silently ignore the packet and may notify the user.

5.1 Hashing the packet

When there is no Authentication we create the signature of the packet by initializing the signature of the packet to NULL. The rest of the fields and the payload should also be initialized as to be sent over the wire. The signature is then made over the complete packet using the Hashing Method defined by the type, thus over the entire AYIYA header and the payload together.

This method allows verification that the packet has not been incidentally mangled along its route to the receiver. It does not provide any security or authenticity that the packet has been forcefully mangled.

5.2 Signing with a Shared Secret

To create the signature of the packet, the Signature Field MUST be set to the signature of the shared secret, this signature is made using the same hashing method as the one specified in the Signature Type Field. The signature is then made over the complete packet, thus the AYIYA header and the payload. By hashing the shared secret we allow shared secrets of arbitrary lengths to be used. Which shared secret is used is out of scope of this document and this should be described by manual or automatic configuration documents which should describe the definition of the shared secret.

The result is stored in the Signature field, which contains the signature of the shared secret while hashing.

Implementations could pre-cache the hashed shared secret and would thus not require the knowledge of the real shared secret.

This method thus allows verification that the packet has not been modified along its path from sender to receiver and also allows verification that the sender or receiver, who should be the only parties knowing the shared secrets, where the originators of this packet. This all without sending the shared secret over the network.

5.3 Signing with a Public/Private Key

The Signature Field of the header is initialized to NULL. The rest of the fields and the payload should also be initialized as to be

sent over the wire. A hash defined by the HshMeth Field is then calculated over the complete packet. After that the Public / Private Key signature is calculated, the result of which is stored in the Signature Field.

6. Identity information in DNS

Some of the Identity Types could represent an IPv4 or IPv6 address or a hostname. Using normal (reverse) DNS lookup procedures the additional properties relating to these identities can then also be found out using DNS. These properties could be alternate endpoint addresses, pointers to home agents or public keys. This can be used for the multihoming and mobility scenarios to bootstrap the initial connection. When a receiver receives the first packet from a, up to then unknown identity, it could lookup the identities additional properties like its public key to be able to authenticate the received packet.

7. Acknowledgements

The protocol presented has formed during the existence of SixXS [SIXXS] to allow the users of the various Tunnel Servers provisioned by SixXS to have a dynamic non-static IPv4 endpoint which could even be located behind a NAT. This protocol is the natural successor of the combination of the proto-41 tunneling protocol and the SixXS Heartbeat protocol.

Thanks to Christian Strauf, Brian Carpenter and Pim van Pelt for valuable comments which improved this document and therefore the protocol a lot.

8. Security Considerations

The shared secret used MUST never be made publicly available to 3rd parties otherwise that 3rd party could sign a packet and automatically reconfigure the tunnel endpoint. This would enable a 3rd party to send traffic in both directions and thus posing as the actual user.

The inclusion of the Epoch Time along with the verification on the receiver side should guard against replay attacks. The receiver MUST ensure that the time difference between local clock and the epochtime never differ for more than 60 seconds. This allows for a tolerance of latency and time-shifts.

Note that the Epoch Time doesn't guard against resending of the same packet. A solution could be to add a sequence number in the AYIYA protocol but that would overcomplicate the receivers as they would

need to keep state and even re-order packets, which is something that is not wanted with a protocol that is built to allow packets to drop. Upper layer protocols should have protection mechanisms against this, e.g. TCP has its own sequence numbering. Not having a sequence number also allows packets to be lost during transmission. One could also choose to encapsulate AYIYA inside TCP to alleviate this issue but note well that while using TCP one inherits also the latency issues because of the reordering which makes TCP of suboptimal use for tunneling packets through.

Any packet that is not well formed or contains an invalid signature MUST be silently dropped, appropriate logging may be done of these issues but in that case a rate limit MUST be applied to not clutter the logs with these messages. Invalid signatures MUST be handled as possibly being spoofed, thus no packet MUST be sent back as these packets would then go to the spoofed source address.

As a side effect of this protocol, when a sender can not or does not send a packet in time, the tunnel is detected as defunct and the receiver will dispose of it. This could be the case when the sender's connectivity is interrupted. Disposition of the tunnel will also make sure that no packets will be forwarded over the tunnel to an endpoint which might not be expecting this kind of traffic as it is not the host that heartbeated the last time. This situation could arise for instance when DHCP changes the endpoint address or a host which dials into a PPP pool disconnects, after which the next dialin, by another host receives the former hosts endpoint address.

This document specifies a tunneling protocol which can circumvent administrative policies implied by a firewall. This firewall can prohibit the communication between sender and receiver. If such a policy is in place, then that is an administrative policy which should not be tried to be circumvented. Using tunneling in general opens up a new hole into a network which might be used for gaining access into that network.

When and one or both of the outer addresses is a [RFC3041] address, any process that receives the AYIYA packets can still make the relation to that single host as the Identity in the AYIYA packet doesn't change. The limited privacy effect of RFC3041 is thus removed in this case.

As mentioned in the "hostname" and "shutdown" sections a receiver may request a shutdown of the tunnel in cases where it suspects that multiple different hosts are using the same authentication data. This could be the case when multiple hosts are mistakenly configured using the same configuration. To limit the load on the receiver the receiver may thus request a shutdown after having notified the

clients of this problem using a MOTD.

9. Scenarios

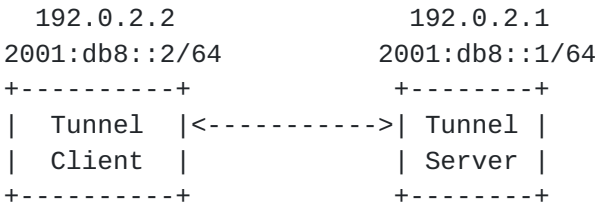
As AYIYA is a generic tunneling protocol it can be used in many different scenarios, amongst which the scenarios described in this section.

Note that TEST-NET [RFC3300] addresses used in the scenarios could never reach a Tunnel Server over the public Internet due to filtering of these documentation prefixes.

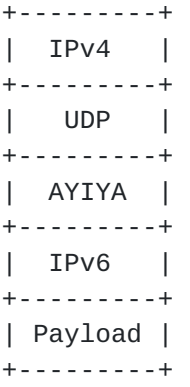
9.1 Using AYIYA for IPv6 Tunnel Brokers

The main scenario where AYIYA is intended to be used is for solving the problem where an IPv4 host is behind a NAT and wants to tunnel to a Tunnel Server [RFC3056]. As many NATs don't support forwarding protocol 41 or require manual configuration of the NAT, using AYIYA and encapsulating the AYIYA packet including the payload inside IPv4 UDP is a good solution. The AYIYA packet includes an identity, thus the endpoint address of the client does not need to be known and the tunnel can be brought and kept up at wish by the user when its client notifies the Tunnel Server of its existence by sending AYIYA packets.

This type of tunnel will generally use a Identity Type of 0x3, the Identity Field will contain the IPv6 address of the endpoint of the tunnel from the direction where the packet is coming from, the Signature Type will be 0x2 (SHA-1).



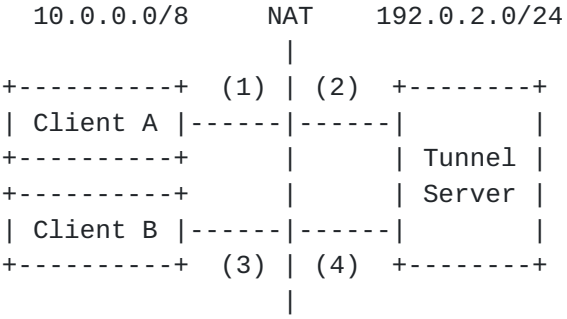
The packet send over the wire will have the following format:



This setup causes a per-packet overhead of: 20 (IPv4) + 8 (UDP) + 8+16+20 (AYIYA+Identity+Signature) = 72 bytes. This allows encapsulation of packets of 1428 bytes over Ethernet, which has a MTU of 1500 bytes. As the minimum MTU of IPv6 packets is 1280 bytes, any medium with at least an MTU of (1280 + 72 =) 1352 bytes can be used for AYIYA without having to fragment the packets.

9.2 Tunneling to multiple endhosts behind a NAT

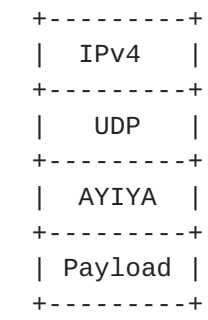
This scenario demonstrates a typical situation where this protocol will mainly be used: tunneling to multiple endhosts behind a NAT. In this scenario the Tunnel Server acts as a receiver in server mode which does not initiate any tunneling as it does not know the source endpoint of the clients, which might change at arbitrary time points. In this scenario the server is assumed to have a static endpoint. The server does not send heartbeats to check connectivity; it is up to the client to send the heartbeats at the agreed regular intervals making sure the server does not dispose of the tunnel. This setup allows both clients behind the NAT to change their private IPv4 addresses and also allows the NAT to change its public IPv4 or source port numbers. The server will notice the changes of source IP or port numbers and can reconfigure its tunnel to send to the specific host:port combination for which a mapping will exist at the NAT and the packet can go through the NAT.



(1) = src = 10.10.0.1:1234, dst = 192.0.2.42:3740
(2) = src = 192.0.2.5:4321, dst = 192.0.2.42:3740
(3) = src = 10.10.9.2:7890, dst = 192.0.2.42:3740
(4) = src = 192.0.2.5:5678, dst = 192.0.2.42:3740

AYIYA is capable of crossing any NAT because an AYIYA receiver uses the AYIYA port as the source port and the address that received the initial AYIYA packet from the client as a source address, Restricted Cone NATs, Port-Restricted Cone NATs and Symmetric NATs can be traversed. If the mapping would change the next packet coming from the client would update the host:port mapping on the receiver.

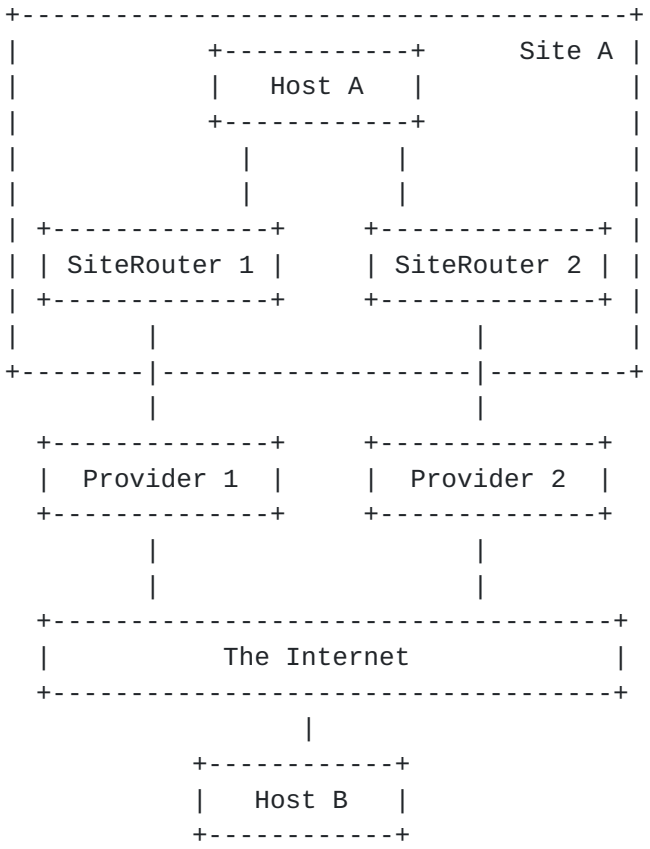
This scenario would typically encapsulate AYIYA and the payload inside IPv4 and UDP. Schematically this would look like:



9.3 Multihoming using AYIYA

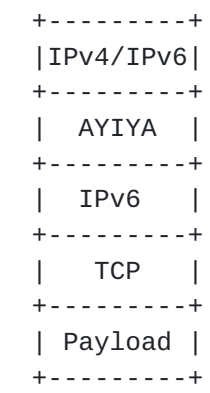
AYIYA can also be used as a tunneling protocol for solving multihoming problems. For instance the following packet could be crafted which encapsulates IPv6 inside IPv6. The encapsulated packet could contain a source/destination address which could be described as the identifiers of this multihoming protocol. The outer IPv6 addresses are the locators.

A typical Multihoming scenario. Site A is connected to the Internet using two independent upstream providers (Provider 1 and Provider 2). Every host inside Site A has two addresses, one from Provider 1 (2001:db8:1000::/48) and one from Provider 2 (2001:db8:2000::/48). Both upstream providers correctly filter egress traffic making sure that only source addresses assigned to Site A from their own address space is sent to the Internet, thus protecting against spoofing. The SiteRouters must thus make sure that only the correctly sourced packets are sent outward. Either the Host or the SiteRouters could support AYIYA, in the first case the SiteRouters MUST never do any additional AYIYA tunneling, this can be accomplished easily by checking that the IPv6 Next Header or IPv4 Protocol field doesn't contain the value mentioning that the next header is an AYIYA header. If this value is not set to be an AYIYA header, the SiteRouters MAY initiate AYIYA traffic to the remote host or site using their Site Identifiers. This allows host-host, host-site and also site-site multihoming.



The hosts communicating with each other using this setup would need to agree on which identities, hashing, authentication methods and shared secrets or private/public keys they use. This is out of scope for this document.

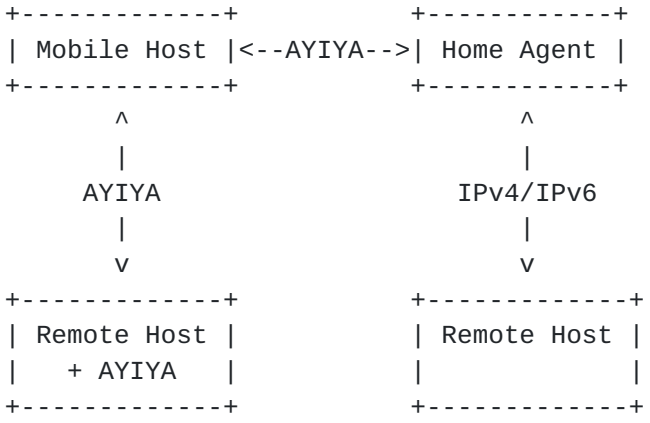
The following figure depicts an example IPv6 TCP packet which is encapsulated using AYIYA inside IPv4 or IPv6 which demonstrates its protocol independency. The Next Header field of the outer IPv6 packet directly contains an IPv6 Next Header value of IANA:TBD. The Next Header field of the AYIYA Header contains the value of 41 (IPv6). The locators used are IPv4 or IPv6 addresses, while the identifiers and the actual protocol addresses that are being multihomed are IPv6. This scenario could allow a host to decide to start communicating with another host over IPv4 when an IPv6 route is not available or doesn't have the required properties, based on latency for instance.



If the locator of a host changes, that host can directly send a heartbeat packet to the other host notifying that host of the change. The receiving host recognizes the new locator as a valid source as the signature can be verified and sets its outgoing packets to use this new endpoint. As the identifiers are encapsulated, existing connections or communications won't notice this change.

9.4 Mobility using AYIYA

AYIYA could be used in a mobility situation for tunneling its Home Address back to the Home Agent, thus acting as a normal tunnel situation and for the Remote Host it seems the communication is happening directly. In this case the remote host doesn't need to support AYIYA. When the Remote Host does support AYIYA, it could also directly setup a tunnel with the mobile host, circumventing that traffic is sent over the Home Agent. The Remote Host can determine if a host supports AYIYA by looking up properties in DNS and use a Public/Private Key algorithm to authenticate the packets without prior information, e.g. the keys, needing to be available. The following diagram illustrates this.



The exact mechanism for determining the public/private key and the identities used are out of scope for this document.

10. IANA Considerations

IANA will need to allocate a protocol number value for "AYIYA" allowing AYIYA packets to be directly encapsulated inside IPv4, IPv6 or possibly any other future protocols.

IANA will need to allocate a port number in the case where AYIYA is used over UDP [RFC0768], TCP [RFC0793] or SCTP [RFC2960] or any other protocol supporting port numbers. A port number request has been made through normal port allocation procedures requesting a system port.

11 References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC

793, September 1981.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC1933] Gilligan, R. and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers", RFC 1933, April 1996.

[RFC2030] Mills, D., "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI", RFC 2030, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC2463] Conta, A. and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 2463, December 1998.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[RFC3041] Narten, T. and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 3041, January 2001.

[RFC3053] Durand, A., Fasano, P., Guardini, I. and D. Lento, "IPv6 Tunnel Broker", RFC 3053, January 2001.

[RFC3056] Carpenter, B. and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, February 2001.

[RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[RFC3300] Reynolds, J., Braden, R., Ginoza, S. and A. De La Cruz, "Internet Official Protocol Standards", RFC 3300, November 2002.

[RFC3513] Hinden, R. and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003.

[SIXXS] Massar, J. and P. van Pelt, "SixXS - IPv6 Deployment & Tunnelbroker", <http://www.sixxs.net>.

Author's Address

Jeroen Massar
Unfix / SixXS
Hofpoldersingel 45
Gouda 2807 LW
NL

EMail: jeroen@unfix.org
URI: http://unfix.org/~jeroen/

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.