Tunneling Through Inner Space

Bob Briscoe*

31 Oct 2014

Abstract

It is common but perhaps misguided to provide extensibility for a layer X header in the layer X header.

> **Strawman principle:** In a middlebox world, it is both more principled and more pragmatic to extend the layer X header within layer $X+1$ (i.e. within the payload encapsulated by the layer X header).[1,2]

For instance, extensions to the IP header should not be located in a chain of extensions to the IP header; they should be in an area of the payload that IP encapsulates (e.g. where you would expect UDP or TCP or an IPv6 destination option). Similarly, extensions to TCP should not be located in the TCP Options within the TCP header, they should be located within the TCP Data (i.e. in the space encapsulated by TCP that is intended for the app-layer).

I'm not yet even convinced myself that this approach warrants promotion to the status of a principle. In the four examples this paper explores, it happened to be possible to make the approach work with some ingenious hackery, but it would not be easy in general.

Therefore the message cannot be that *existing* protocols should be extended like this (because they were not designed to be extended in this way, so it will not always possible). The message is that we can at least try. And, in *future*, protocols would be more extensible in a middlebox world if they were designed so that they could be extended within their own encapsulation.

Terminology

Muddlebox: A middlebox in a muddle—designed to make a quick buck. Saying "protocol extensibility rules" to one of these vendors will earn you a patronising smile.

Meddlebox: A meddling middlebox—designed to seek out potential attacks that do not conform to the stereotype known to be safe. Saying "protocol extensibility rules" to one of these vendors will earn you a patronising snort.

This paper is about extensions, not options. Good protocol design should cater for options and extensions separately. When a protocol is first designed, it is certainly good practice to separate out optional parts—modular structure is good discipline and it makes the core functions efficient. However, extensions are different—by definition, the original design did not cater for them, so they can break the mould.

1 Rationale for the Strawman Principle

Some reasons why the strawman is principled, as well as pragmatic:

- Implementations of layer X that have not implemented or are not interested in the extension to layer X need not be bothered with walking over a load of extensions they do not know or care about.
- Implementations of the extension can be coded to know where to look for the extensions they implement.
- A legacy muddlebox that intervenes at layer X will pass the payload (layer $X+1$) transparently. Therefore extending layer X within its payload should traverse muddleboxes.
- An endpoint (or meddlebox) that needs to intervene in the new layer-X extension can be upgraded to know the location of the extension in layer $X+1$.
- The strawman principle is not a manifesto for extending layer X at layer $X+1, X+2, \dots$ and ever-deeper. This would only happen in the unlikely case where the initial protocol design (with its options) was not fit for purpose,

*bob.briscoe@bt.com, BT Research & Technology, B54/77, Adastral Park, Martlesham Heath, Ipswich, IP5 3RE, UK

[1]Given the Internet architecture denies the existence of L5-6, assume for now that $4+1=7$

[2]I first heard this principle articulated by Rob Hancock in Feb 2010 at a plenary meeting of the Trilogy project.

leading certain layer-$X+1$ extensions to become essential for common use. Then extending these extensions could push into the next layer down.

- Extending layer X should not be confused with attacking at layer X. If an attack on layer X is encapsulated in layer $X+1$, meddleboxes will be reprogrammed to block it. Whereas, if a useful extension to layer X were encapsulated in layer $X+1m$, meddleboxes would not be reprogrammed to block it.
- An extension to layer X intended for cooperation with layer-X middleboxes can be added at layer X or $X+1$, depending respectively whether it will work with legacy middleboxes or not.
- If the endpoints of layer X don't want layer-X middleboxes to intervene in their layer-X extension, they can authenticate and/or encrypt layer $X+1$ and layer-X will still work.

2 Example Tricks & Hacks

Below are some examples of where the strawman principle has already been applied (sometimes unconsciously) to make protocol extensions traverse middleboxes. Each is an example of the ingenious tricks that have been necessary, because the strawman principle had not been articulated when the Internet's protocols were designed.

2.1 Minion

Minion [ICG13] encapsulates a transport protocol within the TCP wire protocol, but provides a richer set of application services:

- multiplexing of multiple messages (or message streams) on a single connection;
- interleaving of multiplexed messages (to eliminate head-of-line blocking);
- message cancellation;
- request/reply support;
- ordered and unordered messages;
- chained messages;
- multiple priority levels with byte-granularity preemption;
- datagram transport layer security (DTLS).

On the wire Minion is indistinguishable from a TCP connection. Therefore it traverses many forms of middlebox.

The Minion transport can be implemented wholly in user space, using only the socket API to TCP in the kernel. Ideally though, it needs one simple kernel extension (in the form of a new socket option). This makes the receiver's TCP stack deliver out-of-order data directly to the application, so that Minion can manage its own message ordering, including interleaving, etc.

TCP's segment boundaries cannot be relied on as the boundaries for Minion messages, because middleboxes might resegment a TCP stream.[3] Therefore Minion uses Recursively Embeddable Consistent Overhead Byte Stuffing (RECOBS) to introduce its own marker bytes at the start and end of messages, and to encode the rest of the stream to avoid these byte values.

Minion does not signal its presence as an explicit transport protocol number in the IP header. It is built into an application that uses a specific ephemeral port, so the app implicitly knows that both ends implement the Minion transport. It would therefore be hard to provide the Minion application services on a well-known port.[4]

Some middleboxes 'normalise' a connection, not only so that it complies with the TCP wire protocol[5], but also so that it complies with TCP's semantics. Therefore Minion had to compromise by keeping TCP's fully reliable retransmission behaviour even though the service it provides to the app allows partial reliability (e.g. cancelling a message if it would arrive too late). Minion only continues to retransmit some messages to keep middleboxes happy—it can still deliver messages out of order, and it can always ignore some retransmissions.

2.2 Inner Space

Inner Space for TCP Options [Bri14] is still in the design stage. It provides more space for TCP options (and *extensions*) than the limited 40B available in the TCP header. Some combinations of recent TCP options already no longer fit alongside the commonly used options, largely because they

[3]Where segment boundaries are concerned, segmentation offload hardware in the NIC of the host can be categorised as a middlebox for our purposes. Nonetheless, real 'boxes' that split TCP connections also alter TCP's e2e segmentation.

[4]Unless the app started with TCP and negotiated to use Minion within the application—quite aside from the problem of having to mimic the app-layer protocol that some middleboxes expect on certain well-known ports, e.g. http on port 80.

[5]Probably as TCP was in the 1990s when the text book that the developer still has from college was written

include cryptographic material that is necessarily fairly large and incompressible (e.g. TCP Fast Option, TCP Authentication Option, tcpcrypt, Multipath TCP).

Inner Space has a lot of similarities with Minion, but with the purpose of locating extra TCP extensions within the TCP data, so that they enjoy more space and traverse middleboxes. It takes care to appear identical to TCP on the wire. Rather than use a TCP option to signal the presence of Inner Space within the TCP Data, it uses a magic number at the start of the byte stream in each direction (estimated to cause a false positive on one connection globally in more than 40 years). Therefore, Inner Space restores TCP's extensibility—not only does it traverse most middleboxes, but extensions do not have to waste handshaking latency checking for option-stripping middleboxes (e.g. for tcpcrypt or mptcp).

The Inner Space protocol within the TCP data is solely a framing protocol that identifies where the segment boundaries were when they were sent, and the extent of the Inner TCP option space on each. The protocol[6] recognises that TCP actually needs two sorts of extensions:

Immediate Those processed in order of reception;

Ordered Those processed in the order they were sent (e.g. those that apply to the byte stream at the point they were inserted, such as re-key commands);

To process extensions in order of reception, the TCP stack on the receiver has to be able to extract these options even if there are still gaps in the earlier byte-stream. To find segment boundaries even if middleboxes have resegmented the stream, it uses a similar but improved scheme to Minion's; called zero overhead message boundary insertion (ZOMBI).

Inner Space is implemented in the TCP stack and it aims to apply to all existing applications that use TCP and all existing TCP options as well as future ones (the TCP client uses an optimistic handshake to detect whether the server supports Inner Space). Therefore, unlike Minion, Inner Space encounters middleboxes that have a stereotypical view of the application layer, not just TCP. On some well-known ports, especially port 80, middleboxes expect the TCP payload to look like HTTP, e.g. a Web filter will parse the HTTP to find the URL.

Fortunately, this does not require the TCP stack to hold knowledge of all well-known app protocols. Instead, Inner Space includes the optional facility to preserve the start of the application layer payload in the first two segments of a connection (it defers the framing header to the end of the first segment). Experiments will determine whether this is sufficient to traverse such deep packet inspection (DPI) boxes.

[6]Not -02 as cited, but the next revision, which will be out by the time of the SEMI workshop

Like Minion, Inner Space has to walk on egg-shells to appear to comply with middleboxes' stereotypical view of TCP:

- Some TCP options acknowledge data (e.g. SACK or MPTCP's Data ACK) and others are applied to pure ACKs (e.g. tcpcrypt's MAC which covers the TCP header). Therefore, Inner Space takes care that all 'Immediate' options do not require any receive window even though they are carried within the TCP Data (they are processed on reception so they are never buffered anyway). Otherwise deadlock could occur if receive buffer were required in one direction to acknowledge data in the other [RPB+12].
- These 'Immediate' options do not require reliable delivery, but they are included with the TCP sequence space solely to pander to middleboxes that would otherwise 'correct' the seq and ack numbers. This means that options within ACKs that otherwise contain no TCP payload consume sequence space (e.g. the MAC option of tcpcrypt). In turn, this means that Inner Space has to suppress ACKing ACKs that contain extensions but no actual payload, otherwise the resulting ACK storm would continue for ever. If some subsequent actual payload is ACKed, it will cover this unACK'd data cumulatively—effectively extremely delayed ACKs.

2.3 ConEx Destination Option

Congestion Exposure (ConEx) is a signal at the IP layer from the sender to certain policy devices in the network (e.g. traffic management functions). ConEx policy devices are therefore middleboxes. After much debate, it was decided to locate ConEx signals [KKU14] within an IPv6 destination option, even though the destination doesn't even need to read it or even be ConEx-aware (nonetheless, the ConEx signal is immutable e2e and can be covered by the e2e authentication of the IPv6 AH header).

The rationale was that ConEx signals needed to be visible at network trust boundaries, where an IP header should always be visible (borders gateways, multiservice edge nodes, gateway GPRS support nodes, etc.). However, it would have been wrong to use a hop-by-hop option, otherwise every IP hop would have had to walk over the option, only to discover it had not been configured to recognise it (ConEx recognition would only be enabled on policy devices).

Locating ConEx signals in a header intended for the next layer up was the best compromise, because ConEx policy devices could be programmed to look for it, even though the destination option was not originally intended for middleboxes.

ConEx highlighted the need for a 'middlebox extension header' in IPv6, to complete the family of hop-by-hop and destination options. A middlebox header would need to include facilities for controlling propagation by tunnel endpoints, so that new middlebox options (like ConEx) will always be copied to the outer IP header over a tunnel, to assure accessibility to middleboxes. However, the design of IPv6 makes the introduction of such a new family of headers fraught with deployment difficulties—there are no controls over how tunnel endpoints treat unrecognised extensions.

2.4 Generic UDP Tunneling (GUT)

Generic UDP Tunneling (GUT [MVB10]) was designed and implemented to allow new transport protocols to be tunnelled over UDP. On the wire it looks like a UDP datagram, but inside it potentially contains another transport protocol. A process would run on a well-known GUT port, and simple decapsulate the contents of the UDP header and re-present it to the stack as if it had just arrived as the IP payload—thus circumventing any middleboxes on the path.

GUT in itself is an example of the strawman principle of extending the transport layer within the payload of the transport layer (UDP). Nonetheless, the really interesting aspect of GUT is that it even includes a facility to extend IP; UDP and GUT headers. The GUT header (encapsulated by UDP) contains a Next Header field that recognises the same extension number space as an IPv6 header. Therefore it can encapsulate new IP extensions or new transport protocols that would not otherwise traverse middleboxes.

GUT was consciously designed using the strawman principle of this paper. The idea being that an IP-aware middlebox or host does not need to see a field in the IP header to be able to find an extension to IP. If a middlebox or host has the code installed to understand an extension, that code tells it where the extension is to be found. The legacy wire protocol does not have to tell it.

3 Guidelines for the Future

All the above tricks have been necessary because the Internet's protocols were designed so that an extension to layer X had to be declared in the layer X header. In retrospect, would it not have been so much easier if the original Internet protocols had followed the strawman principle for protocol extensibility? Below are some strawman guidelines bsed on this principle, for how protocols ought to be designed for extensibility in future, to provide a structure for both middlebox traversal and middlebox co-operation:

- Distinguish between options and extensions.
- Provide space in layer $X + 1$ that can be jumped over by layer-$X + 1$ implementations, so that it can be used to extend layer-X without corrupting the payload used by layer $X + 1$.

These guidelines cover *wire protocol* extension. Further thought is required to develop guidelines for extensibility of protocol *semantics*.

References

[Bri14] Bob Briscoe. Inner Space for TCP Options. Internet Draft draft-briscoe-tcpm-inner-space-01, Internet Engineering Task Force, October 2014. (Work in Progress).

[ICG13] Janardhan Iyengar, Stuart Cheshire, and Josh Graessley. Minion - Wire Protocol. Internet Draft draft-iyengar-minion-protocol-01, Internet Engineering Task Force, July 2013. (Work in Progress).

[KKU14] Suresh Krishnan, Mirja Kuehlewind, and Carlos Ralli Ucendo. IPv6 Destination Option for ConEx. Internet Draft draft-ietf-conex-destopt-07, Internet Engineering Task Force, October 2014. (Work in progress).

[MVB10] Jukka Manner, Nuutti Varis, and Bob Briscoe. Generic UDP Tunnelling (GUT). Internet Draft draft-manner-tsvwg-gut-02, Internet Engineering Task Force, July 2010. (Expired).

[RPB+12] Costin Raiciu, Christophe Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, April 2012.