Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 4, 2015

J. Hildebrand
Cisco Systems
B. Trammell
ETH Zurich

March 03, 2015

Substrate Protocol for User Datagrams (SPUD) Prototype
draft-hildebrand-spud-prototype-02

Abstract

SPUD is a prototype for grouping UDP packets together in a "tube", also allowing network devices on the path between endpoints to participate explicitly in the tube outside the end-to-end context.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 4, 2015.

1. Introduction

The goal of SPUD (Substrate Protocol for User Datagrams) is to provide a mechanism for grouping UDP packets together into a "tube" with a defined beginning and end in time. Devices on the network path between the endpoints speaking SPUD may communicate explicitly with the endpoints outside the context of the end-to-end conversation.

The SPUD protocol is a prototype, intended to promote further discussion of potential use cases within the framework of a concrete approach. To move forward, ideas explored in this protocol might be implemented inside another protocol such as DTLS.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

2. Requirements, Assumptions and Rationale

The prototype described in this document is designed to provide an encapsulation for transport protocols which allows minimal and selective exposure of transport semantics, and other transport- and higher-layer information; and explicit discovery of selected information about devices along the path by the transport and higher layers.

The encryption of transport- and higher-layer content encapsulated within SPUD is not mandatory; however, the eventual intention is that explicit communication between endpoints and the path can largely replace the implicit endpoint-to-path communication presently derived by middleboxes through deep packet inspection (DPI).

SPUD is not a transport protocol; rather, we envision it as the lowest layer of a "transport construction kit". Using SPUD as a common encapsulation, such that new transports have a common appearance to middleboxes, applications, platforms, and operating systems can provide a variety of transport protocols or transport protocol modules. This construction kit is out of scope for this prototype, and left to future work, though we note it could be an alternate implementation of an eventual TAPS interface.

The design is based on the following requirements and assumptions:

- Transport semantics and many properties of communication that endpoints may want to expose to middleboxes are bound to flows or groups of flows. SPUD must therefore provide a basic facility for associating packets together (into what we call a "tube" for lack of a better term).

- SPUD and transports above SPUD must be implementable without requiring kernel replacements or modules on the endpoints, and without having special privilege (root or "jailbreak") on the endpoints. Eventually, we envision that SPUD will be implemented in operating system kernels as part of the IP stack. However, we also assume that there will be a (very) long transition to this state, and SPUD must be useful and deployable during this transition. In addition, userspace implementations of SPUD can be used for rapid deployment of SPUD itself and new transport protocols over SPUD, e.g. in web browsers.

- SPUD must operate in the present Internet. In order to ensure deployment, it must also be useful as an encapsulation between endpoints even before the deployment of middleboxes that understand it.

- SPUD must be low-overhead, specifically requiring very little effort to recognize that a packet is a SPUD packet and to determine the tube it is associated with.

- SPUD must impose minimal restrictions on the transport protocols it encapsulates. SPUD must work in multipath, multicast, and mobile environments.

- SPUD must provide incentives for development and deployment by multiple communities. These communities and incentives will be defined through the prototyping process.

3. Lifetime of a tube

A tube is a grouping of packets between two endpoints on the network. Tubes are started by the "initiator" expressing an interest in comminicating with the "responder". A tube may be closed by either endpoint.

A tube may be in one of the following states:

unknown no information is currently known about the tube. All tubes implicitly start in the unknown state.

opening the initiator has requested a tube that the responder has not yet acknowledged.

running the tube is set up and will allow data to flow

resuming an out-of-sequence SPUD packet has been received for this tube. Policy will need to be developed describing how (or if) this state can be exploited for quicker tube resumption by higher-level protocols.

This leads to the following state transitions (see Section 4.3 for details on the commands that cause transitions):

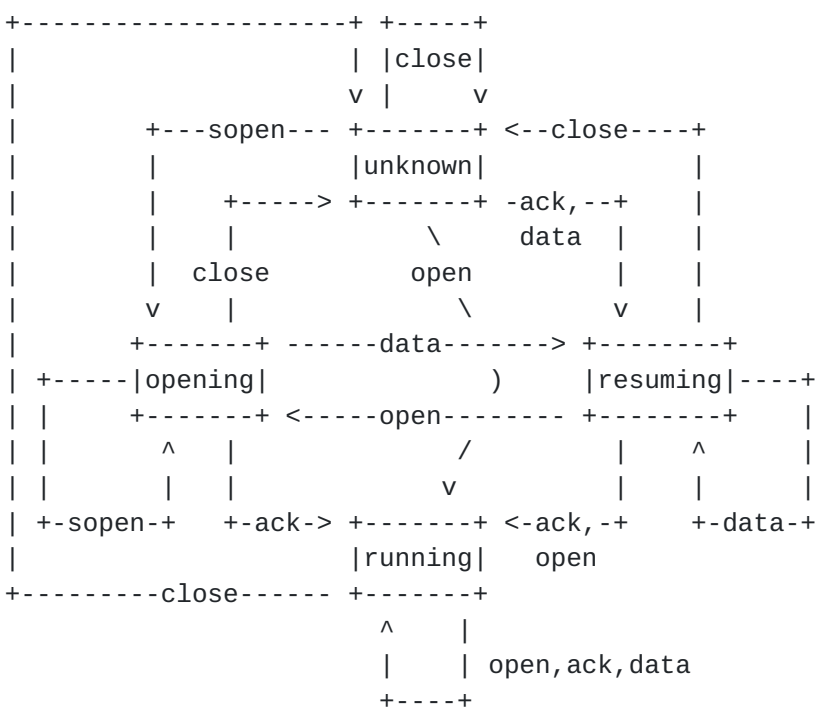

Figure 1: State transitions

All of the state transitions happen when a command is received, except for the "sopen" transition which occurs when an open command is sent.

4. Packet layout

SPUD packets are sent inside UDP packets, with the SPUD header directly after the UDP header.

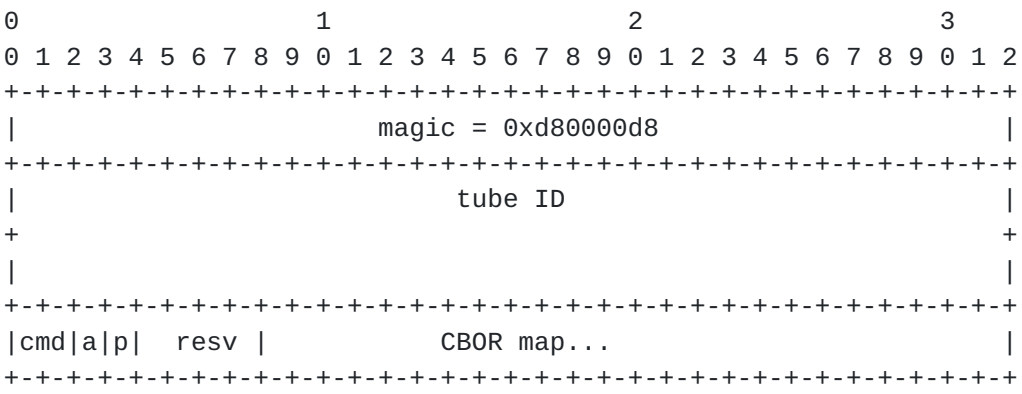

Figure 2: SPUD packets

The fields in the packet are:

- 32-bit constant magic number (see Section 4.1)
- 64 bits defining the id of this tube
- 2 bits of command (see Section 4.3)
- 1 bit marking this packet as an application declaration (adec)
- 1 bit marking this packet as a path declaration (pdec)
- 4 reserved bits that MUST be set to 0 for this version of the protocol
- If more bytes are present, they contain a CBOR map

4.1. Detecting usage

The first 32 bits of every SPUD packet is the constant bit pattern d80000d8 (hex), or 1101 1000 0000 0000 1101 1000 (binary). This pattern was selected to be invalid UTF-8, UTF-16 (both big- and little-endian), and UTF-32 (both big- and little-endian). The intent is to ensure that text-based non-SPUD protocols would not use this pattern by mistake. A survey of other protocols will be done to see if this pattern occurs often in existing traffic.

The intent of this magic number is not to provide conclusive evidence that SPUD is being used in this packet, but instead to allow a very fast (i.e., trivially implementable in hardware) way to decide that SPUD is not in use on packets that do not include the magic number.

4.2. TUBE ID

The 64-bit tube ID uniquely identifies a given tube. All commands (see Section 4.3) are scoped to a single tube.

[EDITOR'S NOTE: Does a Tube ID have to be bound to a single source address or not? This would be for mobility, not multipath.]

4.3. Commands

The next 2 bits of a SPUD packet encode a command:

Data (00) Normal data in a running tube

Open (01) A request to begin a tube

Close (10) A request to end a tube

Ack (11) An acknowledgement to an open request

4.4. Declaration bits

The adec bit is set when the application is making a declaration to the path. The pdec bit is set when the path is making a declaration to the application.

4.5. Reserved bits

The final required four bits of SPUD packet MUST all be set to zero in this version of the protocol. These bits could be used for extensions in future versions.

4.6. Additional information

The information after the SPUD header (if it exists) is a CBOR [RFC7049] map (major type 5). Each key in the map may be an integer (major type 0 or 1) or a text string (major type 3). Integer keys are reserved for standardized protocols, with a registry defining their meaning. This convention can save several bytes per packet, since small integers only take a single byte in the CBOR encoding, and a single-character string takes at least two bytes (more when useful-length strings are used).

The only integer keys reserved by this version of the document are:

0 (anything) Application Data. Any CBOR data type, used as application-specific data. Often this will be a byte string (major type 2), particularly for protocols that encrypt data.

The 0 key MUST NOT be used when the adec or pdec bit is set. Path elements MUST NOT inspect or modify the contents of the 0 key.

The overhead for always using CBOR is therefore effectively three or more bytes: 0xA1 (map with one element), 0x00 (integer 0 as the key), and 0x41 (byte string containing one byte). [EDITOR'S NOTE: It may be that the simplicity and extensibility of this approach is worth the three bytes of overhead.]

5. Initiating a tube

To begin a tube, the initiator sends a SPUD packet with the "open" command (bits 01).

Future versions of this specification may contain CBOR in the open packet. One example might be requesting proof of implementation from the receiving endpoint,

6. Acknowledging tube creation

To acknowledge the creation of a tube, the responder sends a SPUD packet with the "ack" command (bits 11). The current thought is that the security provided by the TCP three-way handshake would be left to transport protocols inside of SPUD. Further exploration of this prototype will help decide how much of this handshake needs to be made visible to path elements that _only_ process SPUD.

Future versions of this specification may contain CBOR in the ack packet. One example might be answering an implementation proof request from the initiator.

7. Closing a tube

To close a tube, either side sends a packet with the "close" command (bits 10). Whenever a path element sees a close packet for a tube, it MAY drop all stored state for that tube. Further exploration of this prototype will determine when close packets are sent, what CBOR they contain, and how they interact with transport protocols inside of SPUD.

What is likely at this time is that SPUD close packets MAY contain error information in the following CBOR keys (and associated values):

"error" (map, major type 5) a map from text string (major type 3) to text string. The keys are [RFC5646] language tags, and the values are strings that can be presented to a user that understands that language. The key "*" can be used as the default.

"url" (text string, major type 3) a URL identifying some information about the path or its relationship with the tube. The URL represents some path condition, and retrieval of content at the URL should include a human-readable description.

8. Path declarations

SPUD can be used for path declarations: information delivered to the endpoints from devices along the path. Path declarations can be thought of as enhanced ICMP for transports using SPUD, allowing information about the condition or state of the path or the tube to be communicated directly to a sender.

Path declarations may be sent in either direction (toward the initiator or responder) at any time. The scope of a path declaration is the tube (identified by tube ID) to which it is associated. Devices along the path cannot make declarations to endpoints without a tube to associate them with. Path declarations are sent to one endpoint in a SPUD conversation by the path device sending SPUD packets with the source IP address and UDP port from the other endpoint in the conversation. These "spoofed" packets are required to allow existing network elements that pass traffic for a given 5-tuple to continue to work. To ensure that the context for these declarations is correct, path declaration packets MUST have the pdec bit set. Path declarations MUST use the "data" command (bits 00).

Path declarations do not imply specific required actions on the part of receivers. Any path declaration MAY be ignored by a receiving application. When using a path declaration as input to an algorithm, the application will make decisions about the trustworthiness of the declaration before using the data in the declaration.

The data associated with a path declaration may always have the following keys (and associated values), regardless of what other information is included:

"ipaddr" (byte string, major type 2) the IPv4 address or IPv6 address of the sender, as a string of 4 or 16 bytes in network order. This is necessary as the source IP address of the packet is spoofed

"cookie" (byte string, major type 2) data that identifies the sending path element unambiguously

"url" (text string, major type 3) a URL identifying some information about the path or its relationship with the tube. The URL represents some path condition, and retrieval of content at the URL should include a human-readable description.

"warning" (map, major type 5) a map from text string (major type 3) to text string. The keys are [RFC5646] language tags, and the values are strings that can be presented to a user that understands that language. The key "*" can be used as the default.

The SPUD mechanism is defined to be completely extensible in terms of the types of path declarations that can be made. However, in order for this mechanism to be of use, endpoints and devices along the path must share a relatively limited vocabulary of path declarations. The following subsections briefly explore declarations we believe may be useful, and which will be further developed on the background of concrete use cases to be defined as part of the SPUD effort.

Terms in this vocabulary considered universally useful may be added to the SPUD path declaration map keys, which in this case would then be defined as an IANA registry.

8.1. ICMP

ICMP [RFC4443] (e.g.) messages are sometimes blocked by path elements attempting to provide security. Even when they are delivered to the host, many ICMP messages are not made available to applications through portable socket interfaces. As such, a path element might decide to copy the ICMP message into a path declaration, using the following key/value pairs:

"icmp" (byte string, major type 2) the full ICMP payload. This is intended to allow ICMP messages (which may be blocked by the path, or not made available to the receiving application) to be bound to a tube. Note that sending a path declaration ICMP message is not a substitute for sending a required ICMP or ICMPv6 message.

"icmp-type" (unsigned, major type 0) the ICMP type

"icmp-code" (unsigned, major type 0) the ICMP code

Other information from particular ICMP codes may be parsed out into key/value pairs.

8.2. Address translation

SPUD-aware path elements that perform Network Address Translation MUST send a path declaration describing the translation that was done, using the following key/value pairs:

"translated-external-address" (byte string, major type 2) The translated external IPv4 address or IPv6 address for this endpoint, as a string of 4 or 16 bytes in network order

"translated-external-port" (unsigned, major type 0) The translated external UDP port number for this endpoint

"internal-address" (byte string, major type 2) The pre-translation (internal) IPv4 address or IPv6 address for this endpoint, as a string of 4 or 16 bytes in network order

"internal-port" (unsigned, major type 0) The pre-translation (internal) UDP port number for this endpoint

The internal addresses are useful when multiple address translations take place on the same path.

8.3. Tube lifetime

SPUD-aware path elements that are maintaining state MAY drop state using inactivity timers, however if they use a timer they MUST send a path declaration in both directions with the length of that timer, using the following key/value pairs:

"inactivity-timer" (unsigned, major type 0) The length of the inactivity timer (in microseconds). A value of 0 means no timeout is being enforced by this path element, which might be useful if the timeout changes over the lifetime of a tube.

8.4. Path element identity

Path elements can describe themselves using the following key/value pairs:

"description" (text string, major type 3) the name of the software, hardware, product, etc. that generated the declaration

"version" (text string, major type 3) the version of the software, hardware, product, etc. that generated the declaration

"caps" (byte string, major type 2) a hash of the capabilities of the software, hardware, product, etc. that generated the declaration [TO BE DESCRIBED]

"ttl" (unisigned integer, major type 0) IP time to live / IPv6 Hop Limit of associated device [EDITOR'S NOTE: more detail is required on how this is calculated]

8.5. Maximum Datagram Size

A path element may tell the endpoint the maximum size of a datagram it is willing or able to forward for a tube, to augment various path MTU discovery mechanisms. This declaration uses the following key/value pairs:

"mtu" (unsigned, major type 0) the maximum transmission unit (in bytes)

8.6. Rate Limit

A path element may tell the endpoint the maximum data rate (in octets or packets) that it is willing or able to forward for a tube. As all path declarations are advisory, the device along the path must not rely on the endpoint to set its sending rate at or below the declared rate limit, and reduction of rate is not a guarantee to the endpoint of zero queueing delay. This mechanism is intended for "gross" rate limitation, i.e. to declare that the output interface is connected to a limited or congested link, not as a substitute for loss-based or explicit congestion notification on the RTT timescale. This declaration uses the following key/value pairs:

"max-byte-rate" (unsigned, major type 0) the maximum bandwidth (in bytes per second)

"max-packet-rate" (unsigned, major type 0) the maximum bandwidth (in packets per second)

8.7. Latency Advisory

A path element may tell the endpoint the latency attributable to traversing that path element. This mechanism is intended for "gross" latency advisories, for instance to declare the output interface is connected to a satellite or [RFC1149] link. This declaration uses the following key/value pairs:

"latency" (unsigned, major type 0) the latency (in microseconds)

8.8. Prohibition Report

A path element which refuses to forward a packet may declare why the packet was not forwarded, similar to the various Destination Unreachable codes of ICMP.

[EDITOR'S NOTE: Further thought will be given to how these reports interact with the ICMP support from Section 8.1.]

9. Declaration reflection

In some cases, a device along the path may wish to send a path declaration but may not be able to send packets ont he reverse path. It may ask the endpoint in the forward direction to reflect a SPUD packet back along the reverse path in this case.

[EDITOR'S NOTE: Bob Briscoe raised this issue during the SEMI workshop, which has largely to do with tunnels. It is not clear to the authors yet how a point along the path would know that it must reflect a declaration, but this approach is included for completeness.]

A reflected declaration is a SPUD packet with both the pdec and adec flags set, and contains the same content as a path declaration would. However the packet has the same source address and port and destination address and port as the SPUD packet which triggered it.

When a SPUD endpoint receives a declaration reflection, it SHOULD reflect it: swapping the source and destination addresses IP addresses and ports. The reflecting endpoint MUST unset the adec bit, sending the packet it as if it were a path declaration.

[EDITOR's NOTE: this facility will need careful security analysis before it makes it into any final specification.]

10. Application declarations

Applications may also use the SPUD mechanism to describe the traffic in the tube to the application on the other side, and/or to any point along the path. As with path declarations, the scope of an application declaration is the tube (identified by tube ID) to which it is associated.

An application declaration is a SPUD packet with the adec flag set, and contains an application declaration formatted in CBOR in its payload. As with path declarations, an application declaration is a CBOR map, which may always have the following keys:

- cookie (byte string, major type 2): an identifier for this application declaration, used to address a particular path element

Unless the cookie matches one sent by the path element for this tube, every device along the path MUST forward application declarations on towards the destination endpoint.

The definition of an application declaration vocabulary is left as future work; we note only at this point that the mechanism supports such declarations.

11. CBOR Profile

Moving forward, we will likely specify a subset of CBOR that can be used in SPUD, including the avoidance of floating point numbers, indefinite-length arrays, and indefinite-length maps. This will allow a significantly less complicated CBOR implementation to be used, which would be particularly nice on constrained devices.

12. Security Considerations

This gives endpoints the ability to expose information about conversations to elements on path. As such, there are going to be very strict security requirements about what can be exposed, how it can be exposed, etc. This prototype DOES NOT tackle these issues yet.

The goal is to ensure that this layer is better than TCP from a security perspective. The prototype is clearly not yet to that point.

13. IANA Considerations

If this protocol progresses beyond prototype in some way, a registry will be needed for well-known CBOR map keys.

14. Acknowledgements

Thanks to Ted Hardie for suggesting the change from "Session" to "Substrate" in the title, and to Joel Halpern for suggesting the change from "session" to "tube" in the protocol description.

15. References

15.1. Normative References

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

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

15.2. Informative References

[RFC1149] Waitzman, D., "Standard for the transmission of IP datagrams on avian carriers", RFC 1149, April 1990.

Authors' Addresses

Joe Hildebrand
Cisco Systems

Email: jhildebr@cisco.com

Brian Trammell
ETH Zurich

Email: ietf@trammell.ch