

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 13, 2013

B. Davie, Ed.
J. Gross
VMware, Inc.
March 12, 2013

**A Stateless Transport Tunneling Protocol for Network Virtualization
(STT)
draft-davie-stt-03**

Abstract

Network Virtualization places unique requirements on tunneling protocols. This draft describes STT (Stateless Transport Tunneling), a tunnel encapsulation that enables overlay networks to be built in virtualized networks. STT is particularly useful when some tunnel endpoints are in end-systems, as it utilizes the capabilities of the network interface card to improve performance.

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 13, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4](#).e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Requirements Language	4
1.2.	Terminology	4
1.3.	Reference Model	5
2.	Design Rationale	5
2.1.	Segmentation Offload	6
2.2.	Metadata	7
2.3.	Context Information	7
2.4.	Alignment	8
2.5.	Equal Cost Multipath	8
2.6.	Efficient Software Processing	8
3.	Frame Formats	8
3.1.	STT Frame Format	9
3.1.1.	Handling non-IP payloads	11
3.2.	Usage of TCP Header by STT	12
3.3.	Encapsulation of STT Segments in IP	13
3.3.1.	Diffserv and ECN-Marking	14
3.3.2.	Packet Loss	14
3.4.	Broadcast and Multicast	15
4.	Interoperability Issues	15
5.	IANA Considerations	16
6.	Security Considerations	16
7.	Contributors	16
8.	Acknowledgements	17
9.	References	17
9.1.	Normative References	17
9.2.	Informative References	18
	Authors' Addresses	18

1. Introduction

Network Virtualization places unique requirements on tunneling protocols. The utility of tunneling in virtualized data centers has been described elsewhere; see, for example [\[I-D.narten-nvo3-overlay-problem-statement\]](#), [\[VL2\]](#), [\[I-D.mahalingam-dutt-dcops-vxlan\]](#), [\[I-D.sridharan-virtualization-nvgre\]](#). Tunneling allows a virtual overlay topology to be constructed on top of the physical data center network, and provides benefits such as:

- o Ability to manage overlapping addresses between multiple tenants
- o Decoupling of the virtual topology provided by the tunnels from the physical topology of the network
- o Support for virtual machine mobility independent of the physical network
- o Support for essentially unlimited numbers of virtual networks (in contrast to VLANs, for example)
- o Decoupling of the network service provided to servers from the technology used in the physical network (e.g. providing an L2 service over an L3 fabric)
- o Isolating the physical network from the addressing of the virtual networks, thus avoiding issues such as MAC table size in physical switches.

This draft describes STT (Stateless Transport Tunneling), a tunnel encapsulation that enables overlay networks to be built in virtualized data center networks, providing the benefits outlined above. STT is particularly useful when some tunnel endpoints are in end-systems, as it utilizes the capabilities of standard network interface cards to improve performance. STT is an IP-based encapsulation and utilizes a TCP-like header inside the IP header. It is, however, stateless, i.e., there is no TCP connection state of any kind associated with the tunnel. The TCP-like header is used for pragmatic reasons, to leverage the capabilities of existing network interface cards, but should not be interpreted as implying any sort of connection state between endpoints.

STT is typically used to carry Ethernet frames between tunnel endpoints. These frames may be considerably larger than the MTU of the physical network - up to 64KB. Fields in the tunnel header are used to allow these large frames to be segmented at the entrance to the tunnel according to the MTU of the physical network and

subsequently reassembled at the far end of the tunnel.

1.1. Requirements Language

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 [RFC 2119](#) [[RFC2119](#)].

1.2. Terminology

The following terms are used in this document:

Stateless Transport Tunneling (STT). The tunneling mechanism defined in this document. The name derives from the fact that the tunnel header resembles the TCP/IP headers (hence "transport" tunneling) while "stateless" refers to the fact that none of the normal TCP state (connection state, send and receive windows, congestion state etc.) is associated with the tunnel (as would be required if an actual TCP connection were used for tunneling).

STT Frame. The unit of data that is passed into the tunnel prior to segmentation and encapsulation. This frame typically consists of an Ethernet frame and an STT Frame header. These frames may be up to 64KB in size.

STT Segment. The unit of data that is transmitted on the underlay network over which the tunnel operates. An STT segment has headers that are syntactically the same as the TCP/IP headers, and typically contains part of an STT frame as the payload. These segments must fit within the MTU of the physical network.

Context ID. A 64-bit field in the STT frame header that conveys information about the disposition of the STT frame between the tunnel endpoints. One example use of the Context ID is to direct delivery of the STT frame payload to the appropriate virtual network or virtual machine.

MSS. Maximum Segment Size. The maximum number of bytes that can be sent in one TCP segment [[RFC0793](#)].

NIC. Network Interface Card.

TSO. TCP Segmentation Offload. A function provided by many commercial NICs that allows large data units to be passed to the NIC, the NIC being responsible for creating MSS-sized segments with correct TCP/IP headers.

LR0. Large Receive Offload. The receive-side equivalent function of

TSO, in which multiple TCP segments are coalesced into larger data units.

VM. Virtual Machine.

1.3. Reference Model

Our conceptual model for a virtualized network is shown in Figure 1. STT tunnels extend in this figure from one virtual switch to another, providing a virtual link between the switches over some arbitrary underlay. More generally, STT tunnels operate between a pair of tunnel endpoints; these endpoints may be virtual switches, physical switches, or some other device (e.g. an appliance). The STT tunnel provides a virtual point-to-point Ethernet link between the endpoints. Frames are handed to the tunnel by some entity (e.g. a VM that is connected to a virtual switch in this picture) and first encapsulated with an STT Frame header. STT Frames may then be fragmented in the NIC, and are encapsulated with a tunnel header (the STT segment header) for transmission over the underlay. Note that other models are possible, e.g., where one or both tunnel endpoints are implemented in a physical switch. In such cases the tunnel endpoint may forward packets to and from another link (physical or virtual) rather than to a VM.

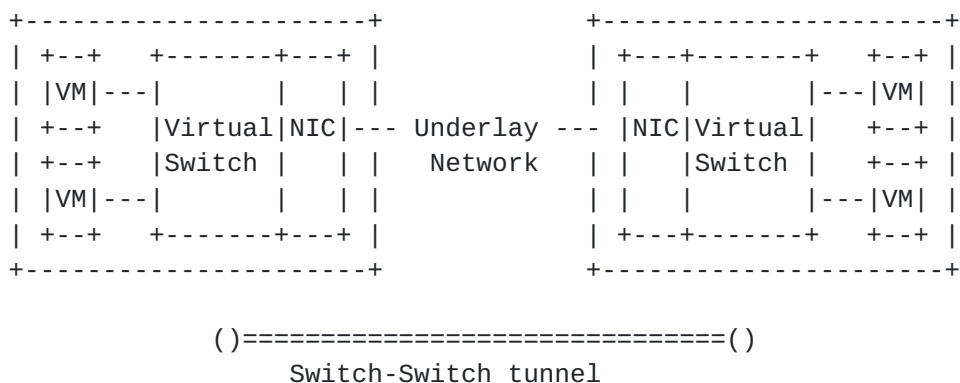


Figure 1: STT Reference Model

2. Design Rationale

We take as given the need for some form of tunneling to support the virtualization of the network as described in [Section 1](#). One might reasonably ask whether some existing tunneling protocol such as GRE[RFC2784] or L2TPv3[RFC3931] might suffice. In fact,

[I-D.sridharan-virtualization-nvgre] does just that, using GRE. The primary motivation for STT as opposed to one of the existing tunneling methods is to improve the performance of data transfers from hosts that implement tunnel endpoints. We expand on this rationale below.

2.1. Segmentation Offload

A large percentage of network interface cards (NICs) in use today are able to perform TCP segmentation offload (TSO). When a NIC supports TSO, the host hands a large (greater than 1 TCP MSS) frame of data to the NIC along with a set of metadata which includes, among other things, the desired MSS, and various fields needed to complete the TCP header. The NIC fragments the frame into MSS-sized segments, performs the TCP Checksum operation, and applies the appropriate headers (TCP, IP and MAC) to each segment.

On the receive side, some NICs support the reassembly of TCP segments, a function referred to as large receive offload (LRO). In this case, NICs attempt to reassemble TCP segments and pass larger aggregates of data to the host. (Since TCP's service model is a byte stream, there is no higher level frame for the NIC to reassemble, but it can pass chunks of the stream larger than one MSS to the host. Full reassembly of STT frames is handled in the host.) The benefits to the host include fewer per-packet operations and larger data transfers between host and NIC, which amortizes the per-transfer cost (such as interrupt processing) more efficiently. These gains can translate into significant performance gains for data transfer from the host to the network.

STT is explicitly designed to leverage the TSO capabilities of currently available NICs. While one might think of segmentation as a generic function, the majority of NICs are designed specifically to support TCP segmentation offload, as the details of the segmentation function are highly dependent on the specifics of TCP. In order to leverage such capability, therefore, the STT segment header is syntactically identical to a valid TCP header. However, we use some of the fields in the TCP header (specifically, sequence number and ACK number) to support the objectives of STT. The details are described in [Section 3.2](#). In essence, we need the same set of information that IP datagrams carry when IP fragmentation takes place: a unique identifier for the frame that has been fragmented, an offset into that frame for the current fragment, and the length of the frame to be reassembled. We fit these fields into the TCP header fields traditionally used for the SEQ and ACK numbers. STT segments are transmitted as IP datagrams using the TCP protocol number (6). The primary means to recognize STT segments is the destination port number. We discuss the interoperability impact of these design

choices in [Section 4](#).

The net effect of using TSO is that the frame size that is sent by endpoints in the virtualized network can be much larger than the MTU of the underlying physical network. The primary benefit of this is a significant performance gain when large amounts of data are being transferred between nodes in the virtual network. A secondary effect is that the header of the STT frame is amortized across a larger amount of data, reducing the need to shrink the STT frame header to minimum size.

Note that, while segmentation offload is the primary NIC function that STT takes advantage of, other NIC offload functions such as checksum calculation can also be leveraged.

[2.2. Metadata](#)

When a frame is delivered to the NIC that supports TSO for segmentation and transmission, a certain amount of metadata is typically passed along with it. This includes the MSS and potentially a VLAN tag to be applied to the transmitted packets.

In some virtualized network deployments, an STT frame may traverse a tunnel, be received and reassembled at an STT endpoint, and then be sent on another physical interface. In such cases, the tunnel terminating endpoint may need to pass metadata to a NIC to enable transmission of frames on the physical link. For this reason, appropriate metadata is carried in the STT frame header.

[2.3. Context Information](#)

When an STT Frame is received by a tunnel endpoint, it needs to be directed to the appropriate entity in the virtualized network to which it belongs. For this reason, a Context ID is required in the STT frame header. Some other encapsulations (e.g. [\[I-D.mahalingam-dutt-dcops-vxlan\]](#), [\[I-D.sridharan-virtualization-nvgre\]](#)) use an explicit tenant network identifier or virtual network identifier. The Context Identifier can be thought of as a generalized form of virtual network identifier. Using a larger and more general identifier allows for a broader range of service models and allows ample room for future expansion. There is little downside to using a larger field here because it is amortized across the entire STT Frame rather than being present in each packet.

2.4. Alignment

Software implementations of tunnel endpoints benefit from 32-bit alignment of the data to be manipulated. Because the Ethernet header is not a multiple of 32-bits (it is 14 bytes), 2 bytes of padding are added to the STT header, causing the payload beyond the encapsulated Ethernet header, which typically includes the IP header of the encapsulated frame, to be 32-bit aligned.

2.5. Equal Cost Multipath

It is essential that traffic passing through the physical network can be efficiently distributed across multiple paths. Standard equal cost multipath (ECMP) techniques involve hashing on address and port numbers in the outer protocol headers. There are two main issues to address with ECMP. First, it is important that, when a set of packets belong to a single flow (e.g. a TCP connection in the virtual network), all those packets should follow the same path. Second, all paths should be used efficiently, i.e. there needs to be sufficient entropy among the different flows to ensure they get distributed evenly across multiple paths.

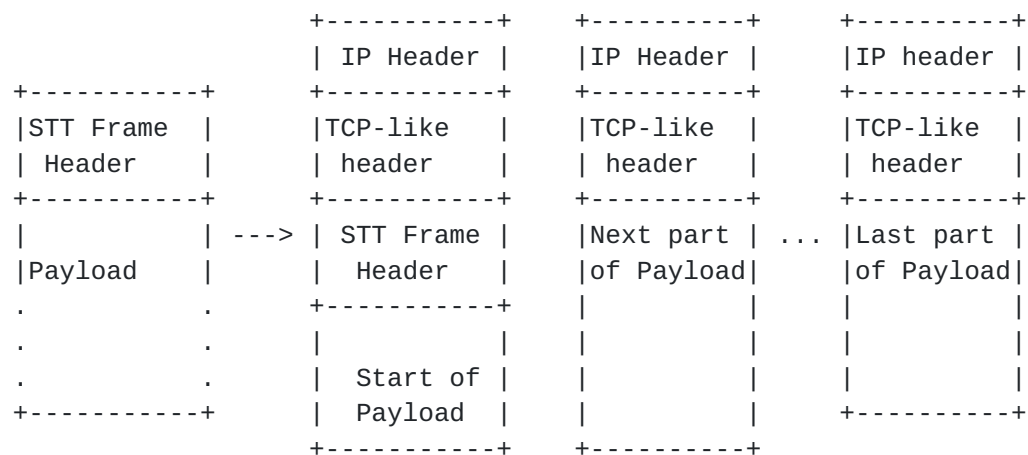
STT achieves the first goal by ensuring that the source and destination ports and addresses in the outer header are all the same for a single flow. The second goal is achieved by generating the source port using a random hash of fields in the headers of the inner packets, e.g. the ports and addresses of the virtual flow's packets. We provide more details on the usage of port numbers in [Section 3.2](#).

2.6. Efficient Software Processing

The design of STT is largely motivated by the desire to tunnel packets efficiently between virtual switches running in software. In addition to the points noted above, this leads to some design optimizations to simplify processing of packets, such as the use of an "L4 offset" field in the STT header to enable the payload to be located quickly without extensive header parsing.

3. Frame Formats

STT encapsulates data payloads of up to 64KB (limited by the length field in the STT header, described below). Those frames are then segmented (depending on the MTU of the underlying physical network) and the resulting segments are encapsulated in a standard TCP header, which in turn is encapsulated by an IP header and finally a MAC header. This is illustrated in Figure 2.



Original data
frame is encappd
with STT Header

STT Frame is segmented and transmitted as
a set of TCP segments (MAC
headers not shown)

Figure 2: STT Frame Fragments and Encapsulation

The details of the STT Frame header and the usage of the TCP-like header are described in detail below. The TCP segments shown in Figure 2 are of course further encapsulated as IP datagrams, and may be sent as either IPv4 or IPv6. The resulting IP datagrams are then transmitted in the appropriate MAC level frame (e.g. Ethernet, not shown in the figure) for the underlying physical network over which the tunnels are established.

3.1. STT Frame Format

Figure 3 illustrates the header of an STT frame before it is segmented.

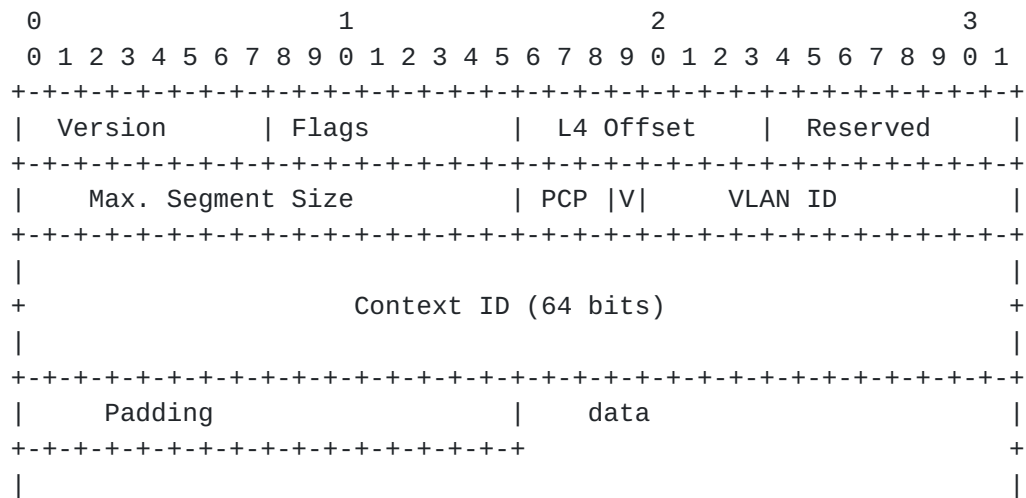


Figure 3: STT Frame Format

The STT frame header contains the following fields:

- o Version - currently 0.
- o Flags - describes encapsulated packet, see below.
- o L4 offset - offset in bytes from the end of the STT Frame header to the start of the encapsulated layer 4 (TCP/UDP) header.
- o Reserved field - MUST be zero on transmission and ignored on receipt.
- o Max Segment Size - the TCP MSS that should be used by a tunnel endpoint that is transmitting this frame onto another network.
- o PCP - the 3-bit Priority Code Point field that should be applied to this packet by an STT tunnel endpoint on transmission to another network (see [Section 2.2](#)).
- o V - a one bit flag that, if set, indicates the presence of a valid VLAN ID in the following field and valid PCP in the preceding field.
- o VLAN ID - 12-bit VLAN tag that should be applied to this packet by an STT tunnel endpoint on transmission to another network (see [Section 2.2](#)).
- o Context ID - 64 bits of context information, described in detail in [Section 2.3](#).

- o Padding - 16 bits as described above.

The flags field contains:

- o 0: Checksum verified. Set if the checksum of the encapsulated packet has been verified by the sender.
- o 1: Checksum partial. Set if the checksum in the encapsulated packet has been computed only over the TCP/IP header. This bit MUST be set if TSO is used by the sender. Note that bit 0 and bit 1 cannot both be set in the same header.
- o 2: IP version. Set if the encapsulated packet is IPv4, not set if the packet is IPv6. See below for discussion of non-IP payloads.
- o 3: TCP payload. Set if the encapsulated packet is TCP.
- o 4-7: Unused, MUST be zero on transmission and ignored on receipt.

As noted above, several of these fields are present primarily to enable efficient processing of the packet when it received at a tunnel endpoint. (For example, it's entirely possible to determine if the packet is IPv4 or IPv6 by looking at the Ethernet header - it's just more efficient not to have to do so.)

The payload of the STT frame is an untagged Ethernet frame.

3.1.1. Handling non-IP payloads

Note that the STT header does not have a general "protocol" field to allow the efficient processing of arbitrary payloads. The current version is designed to provide a virtual Ethernet link, and hence efficiently supports only Ethernet frames as the payload. The Ethernet header itself contains a protocol field, which then identifies the higher layer protocol, so it is straightforward to accommodate non-IP traffic.

It will be noted that the STT Frame header does contain fields that are intended to assist in efficient processing of IPv4 and IPv6 packets. These fields MUST be set to zero and ignored on receipt for non-IP payloads.

The use of STT to carry payloads other than Ethernet is theoretically possible but is beyond the scope of this document.

3.2. Usage of TCP Header by STT

Figure 4 illustrates the usage of the TCP header STT. This figure is essentially identical to that in [RFC0793] with the exception that we denote with an asterisk (*) two fields that are used by STT to convey something other than the information that is conveyed by TCP. Syntactically, STT segments look identical to TCP segments. However, STT tunnel endpoints treat the Sequence number and Acknowledgment number differently than TCP endpoints treat those fields. Furthermore, as noted above, there is no TCP state machine associated with an STT tunnel.

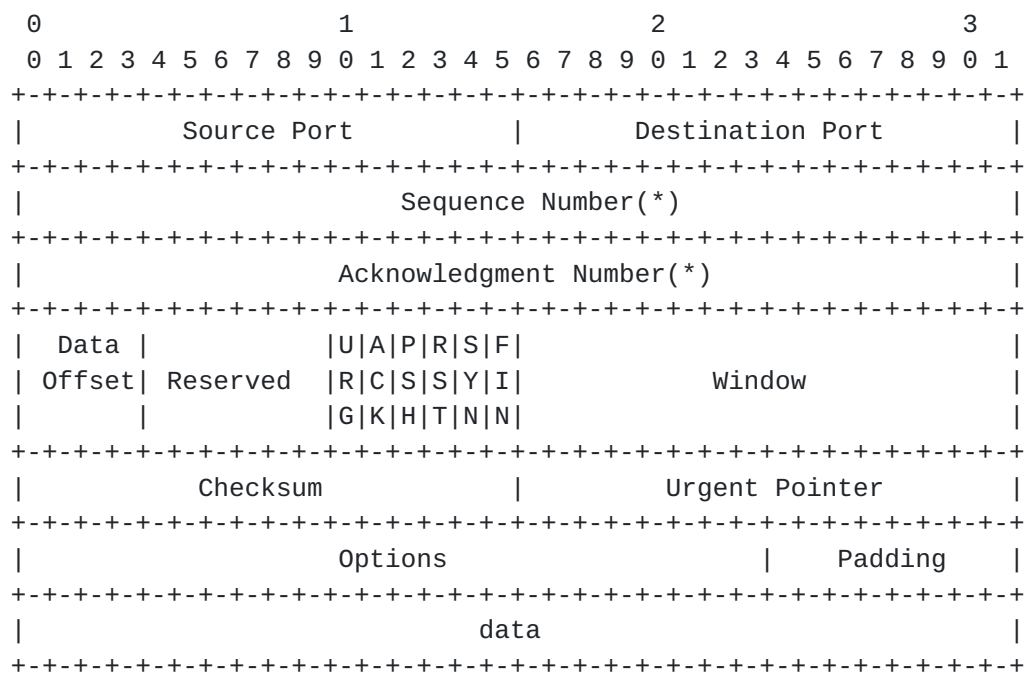


Figure 4: STT Segment Format

The Destination port is to be requested from IANA, in the user range (1024-49151).

In order to allow correct reassembly of the STT frame, the source port **MUST** be constant for all segments of a single STT frame.

As noted above ([Section 2.5](#)) the source port **SHOULD** be the same for all frames that belong to a single flow in the virtual network, e.g. a single TCP connection.

Also, to encourage efficient distribution of traffic among multiple paths when ECMP is used, the method to calculate the source port should provide a random distribution of source port numbers. An

example mechanism would be a random hash on ports and addresses of the TCP headers of the flow in the virtual network.

It is RECOMMENDED to use a source port number from the ephemeral range defined by IANA (49152-65535).

The Sequence number and Acknowledgment number fields are re-purposed in a way that does not confuse NICs that expect them to be used in the conventional manner. The ACK field is used as a packet identifier for the purposes of fragmentation, equivalent in function to the Identification field of IPv4 or the IPv6 Fragment header: it MUST be constant for all STT segments of a given frame, and different from any value used recently for other STT frames sent over this tunnel.

The upper 16 bits of the the SEQ field are used to convey the length of the STT frame in bytes. The lower 16 bits of the SEQ field are used to convey the offset (in bytes) of the current fragment within the larger STT frame.

Reassembly of the fragments may be done partially by NICs that perform LRO, since the sequence numbers of frames will increment appropriately. That is, the upper 16 bits don't change, and the lower 16 bits increment by N for every N byte segment that is transmitted, just as would be the case if an actual sequence number were being sent. Note that the size limit of an STT frame ensures that sequence numbers cannot wrap while sending the segments of a single STT frame.

All the fields after ACK have their conventional meaning, although nothing will be done with the Window or Urgent pointer values. Those fields SHOULD be zero on transmit and ignored on receipt. It is RECOMMENDED that the PSH (Push) flag be set when transmitting the last segment of a frame in order to cause data to be delivered by the NIC without waiting for other fragments. The ACK flag SHOULD be set to ensure that a receiving NIC passes the ACK field to the host to assist in reassembly. All other flags SHOULD be zero on transmit and ignored on receipt.

3.3. Encapsulation of STT Segments in IP

From the perspective of IP, an STT segment is just like any other TCP segment. The protocol number (IPv4) or Next Header (IPv6) has the value 6, as for regular TCP. The resulting IP datagram is then encapsulated in the appropriate L2 header (e.g. Ethernet) for transmission on the physical medium.

3.3.1. Diffserv and ECN-Marking

When traffic is encapsulated in a tunnel header, there are numerous options as to how the Diffserv Code-Point (DSCP) and ECN markings are set in the outer header and propagated to the inner header on decapsulation.

[RFC2983] defines two modes for mapping the DSCP markings from inner to outer headers and vice versa. The Uniform model copies the inner DSCP marking to the outer header on tunnel ingress, and copies that outer header value back to the inner header at tunnel egress. The Pipe model sets the DSCP value to some value based on local policy at ingress and does not modify the inner header on egress. Both models SHOULD be supported by STT endpoints. However, there is an additional complexity with the uniform model for STT, because a single IP datagram that is transmitted over the tunnel appears as multiple IP datagrams on the wire. Thus it is not guaranteed that all segments of the STT frame will have the same DSCP at egress. If uniform model behavior is configured, it is RECOMMENDED that the DSCP of the first segment of the STT frame be used to set the DSCP value of the IP header in the decapsulated STT frame.

[RFC6040] describes the correct ECN behavior for any type of IP in IP tunnel, and this behavior SHOULD be followed for STT tunnels. As with the Uniform Diffserv tunnel model, the fact that one inner IP datagram is segmented into multiple outer datagrams makes the situation slightly more complex. It is RECOMMENDED that if any segment of the received STT frame has the CE (congestion experienced) bit set in its IP header, then the CE bit SHOULD be set in the IP header of the decapsulated STT frame.

3.3.2. Packet Loss

Individual IP datagrams may be dropped (most often due to congestion) and, since there is no acknowledgment or reliable delivery of these datagrams, there is the potential to corrupt an entire STT Frame due to the loss of a single IP datagram. Fortunately, there are solutions to this problem in the case where the higher layer protocol running over STT is TCP. An STT receiving endpoint running in an end-system, as shown in Figure 1 for example, is not required to deliver complete STT frames to the TCP stack in the receiving VM. A partial frame payload can be delivered and the receiving TCP stack can deal with the missing bytes just as it would if running directly over a physical network. That is, TCP in the VM can send ACKs for the contiguous bytes received to trigger retransmission of the missing bytes by the sender. This is similar to the operation of LRO in current NICs. There are some subtleties to making this work correctly in the STT context, and it does depend on the STT endpoint

being aware of the higher layer protocols consuming data in the VM to which it is connected. The main point of this discussion is that, in the common deployments of STT running in a virtual switch, the potential harm of losing individual packets is not as serious as it might first appear.

3.4. Broadcast and Multicast

It is possible to establish point-to-multipoint STT tunnels by using an IP multicast address as the destination address of the tunnel. These may be used for broadcast or multicast traffic if the underlying physical network supports IP multicast. Control mechanisms for setting up such multicast groups are beyond the scope of this document. It is worth repeating that, despite the syntactic resemblance between the STT segment header and the TCP header, there is no TCP state machine associated with an STT tunnel, so the traditional issues of combining multicast with TCP (or reliable transports more generally) do not arise.

4. Interoperability Issues

It will be noted that an STT packet on the wire appears exactly the same as a TCP packet, but that processing of an STT packet on reception is entirely different from TCP - no three-way handshake to establish a connection, no ACKs, retransmission, etc. Hence, an STT tunnel endpoint clearly needs to be configured to behave in the correct manner rather than to perform standard TCP processing on the packet. The primary way to recognize an STT segment is the destination port number in the TCP header. In the event that an STT packet is inadvertently delivered to a device that is not configured to behave as an STT tunnel endpoint, no TCP connection will be established and STT packets will be dropped.

In the event that STT packets pass through middle boxes that process TCP, it is likely that (in the near term at least) they will be dropped, as there will be no TCP connection state established. This is clearly undesirable, but it is a general issue with any form of tunneling - the nature of many middle boxes is that they will not permit tunnels to pass through them. Hence the best solution is simply to avoid deploying middle boxes at locations where STT tunnels (or other forms of tunnels for network virtualization) will need to pass through them. This will not, however, always be feasible, especially when virtualized networks extend among multiple data centers. Other solutions include configuring the middle boxes to permit TCP packets to pass through when the port number matches the port assigned for STT.

In the longer term, we might reasonably expect that middle boxes would be able to recognize STT traffic, and to terminate and originate STT tunnels if necessary (e.g. to perform functions that require the STT payload to be inspected such as statefull firewalling).

It is also of course possible to provide all the functionality of STT using a different IP protocol number (or next header value in IPv6). This approach makes sense in the long run but will typically not enable current NIC hardware to be leveraged for TSO and LRO functions.

It is also possible to run STT traffic over other forms of tunnel (GRE, IPSEC, etc.) in which case they the STT traffic can pass through appropriately configured middle boxes.

5. IANA Considerations

A TCP port in the user range (1024- 49151) will be requested from IANA.

6. Security Considerations

In the physical network, STT packets are simply IP datagrams, and do not introduce new security issues. Most standard IP security mechanisms (such as IPSEC encryption or authentication) can be implemented on STT packets if desired. As noted above, however, tunneling generally interacts poorly with middle boxes, and STT is no exception. Devices such as firewalls are likely to drop STT traffic unless the capability to recognize STT packets is implemented, or unless the STT traffic is itself run over some sort of tunnel that the firewall is configured to permit. Intrusion detection systems would similarly need to be enhanced to be able to look inside STT packets.

It should also be noted that while STT packets resemble TCP segments, the lack of a TCP state machine means that TCP-related security issues (e.g. SYN-flooding) do not apply. Similarly, some of the benefits of the TCP state machine (e.g. the ability to discard packets with unexpected sequence numbers) are also absent for STT traffic.

7. Contributors

The following individuals contributed to this document:

Brad McConnell
Rackspace
5000 Walzem Road
San Antonio, TX 78218
Email: bmcconne@rackspace.com

JC Martin
eBay
2145 Hamilton Ave.
San Jose, CA 95125
Email: jcmartin@ebaysf.com

Iben Rodriguez
eBay
2477 Woodland Ave
San Jose, CA 95128
Email: Iben.rodriguez@gmail.com

Ilango Ganga
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA - 95054
Email: ilango.s.ganga@intel.com

Igor Gashinsky
Yahoo!
111 West 40th Street
New York, NY 10018
Email: igor@yahoo-inc.com

8. Acknowledgements

We thank Martin Casado for inspiring this work and making all the introductions, and to Ben Pfaff for his explanations of the implementation. Thanks also to Pierre Ettori, Yukio Ogawa and Koichiro Seto for their helpful comments.

9. References

9.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, [RFC 793](#), September 1981.

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

9.2. Informative References

- [I-D.mahalingam-dutt-dcops-vxlan]
Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", [draft-mahalingam-dutt-dcops-vxlan-03](#) (work in progress), February 2013.
- [I-D.narten-nvo3-overlay-problem-statement]
Narten, T., Black, D., Dutt, D., Fang, L., Gray, E., Kreeger, L., Napierala, M., and M. Sridhavan, "Problem Statement: Overlays for Network Virtualization", [draft-narten-nvo3-overlay-problem-statement-04](#) (work in progress), August 2012.
- [I-D.sridharan-virtualization-nvgre]
Sridharan, M., Greenberg, A., Venkataramaiah, N., Wang, Y., Duda, K., Ganga, I., Lin, G., Pearson, M., Thaler, P., and C. Tumuluri, "NVGRE: Network Virtualization using Generic Routing Encapsulation", [draft-sridharan-virtualization-nvgre-02](#) (work in progress), February 2013.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", [RFC 2784](#), March 2000.
- [RFC2983] Black, D., "Differentiated Services and Tunnels", [RFC 2983](#), October 2000.
- [RFC3931] Lau, J., Townsley, M., and I. Goyret, "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", [RFC 3931](#), March 2005.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", [RFC 6040](#), November 2010.
- [VL2] Greenberg et al, "VL2: A Scalable and Flexible Data Center Network", 2009.
Proc. ACM SIGCOMM 2009

Authors' Addresses

Bruce Davie (editor)
VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
USA

Email: bdavie@vmware.com

Jesse Gross
VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
USA

Email: jgross@vmware.com

