Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2016

M. Kuehlewind
B. Trammell
ETH Zurich

July 3, 2015

SPUD Use Cases
draft-kuehlewind-spud-use-cases-00

Abstract

The Substrate Protocol for User Datagrams (SPUD) BoF session at the IETF 92 meeting in Dallas in March 2015 identified the potential need for a UDP-based encapsulation protocol to allow explicit cooperation with middleboxes while using new, encrypted transport protocols. This document summarizes the use cases discuss at the BoF and thereby proposes a structure for the description of further use cases.

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 January 4, 2016.

Table of Contents

1. Introduction

This document describe use cases for a common Substrate Protocol for User Datagrams (SPUD) that could be used by an overlaying transport or application to explicitely expose information to middleboxes or request information from (SPUD-aware) middleboxes.

For each use case, we first describe a problem that can not be solved with current protocols, or only solved inefficiently. We then discuss which information should be exposed by which party to help the described problem. We also discuss potential mechanisms to use that exposed information at middleboxes and/or endpoints, in order to demonstrate the feasibility of using the exposed information to the given use case. The described mechanisms are not necessarily proposals for moving forward, nor do they necessarily represent the best approach for applying the exposed information, but should illustrate and motivate the applicability of the exposed information.

In this document we assume that there is no pre-existing trust relationship between the communication endpoints and any middlebox on the path. Therefore we must always assume that information that is exposed can be wrong or nobody will actually act based on the exposed information. However, for the described use cases there should still be a benefit, e.g if otherwise no information would be available.

Based on each mechanism, we discuss deployment incentives of each involved party. There must be clear incentives for each party to justify the proposed information exposure and at best an incremental deployment strategy. Finally, we discuss potential privacy concerns regarding the information to be exposed, as well as potential security issues of the proposed mechanisms.

2. Firewall Traversal

2.1. Problem Statement

Today UDP is often blocked by firewalls, or only enabled for a few well-known applications. However, this makes it hard to deploy new services on top of UDP.

For a long time UDP has not been used much for high volume traffic and therefore it was assumed that most UDP traffic is spam or attack traffic. This is not true anymore. The volume of (good) UDP traffic is growing, mostly due to voice and video (real-time) services, e.g. RTCWEB uses UDP for data and media, where TCP is not suitable anyway.

Even if firewall administrators are willing to implement new rules for UDP services, it is hard to track session state for UDP traffic. As UDP is unidirectional, it is unknown whether the receiver is willing to accept the connection. Further there is no way to figure how long state must be maintained once established. To efficiently establish state along the path we need an explicit contract, as is done implicitly with TCP today.

2.2. Information Exposure

To maintain state in the network, it must be possible to easily assign each packet to a session that is passing a certain network node. This state should be bound to something beyond the five-tuple to link packets together. In [I-D.trammell-spud-req] propose the use of identifiers "tubes". This allows for differential treatment of different packets within one five-tuple flow, presuming the application has control over segmentation and can provide requirements on a per-tube basis. Tube IDs must be hard to guess: a tube ID in addition to a five-tuple as an identifier, given significant entropy in the tube ID, provides an additional assurance that only devices along the path or devices cooperating with devices along the path can send packets that will be recognized by middleboxes and endpoints as valid.

Further, to maintain state, the sender must explicitly indicate the start and end of a tube to the path, while the receiver must confirm connection establishment. This, together with the first packet following the confirmation, provides a guarantee of return routability; i.e. that the sender is actually at the address it says it is. This impies all SPUD tubes must be bidirectional, or at least support a feedback channel for this confirmation. Even though UDP is not a bidirectional transport protocol, often services on top of UDP are bidirectional anyway. Even if not, we only require one packet to acknowledge a new connection. This is low overhead for this basic

security feature. This connection set-up should not impose any additional start-up latency, so the sender must be also able to send payload data in the first packet.

If a firewall blocks a SPUD packet, it can be beneficial for the sender to know why the packet was blocked. Therefore a SPUD-aware middlebox should be able to send error messages. Such an error message can either be sent directly to the sender itself, or alternatively to the receiver that can decide to forward the error message to a sender or not.

2.3. Mechanism

A firewall or middlebox can use the tube ID as an identifier for its session state information. If the tube ID is large enough it will be hard for a non-eavesdropping attacker to guess the ID.

If a firewall receives a SPUD message that signals the start of a connection, it can decide to establish new state for this tube. Alternatively, it can also forward the packet to the receiver and wait if the connection is wanted before establishing state. To not require forwarding of unknown payload, a firewall might want to forward the initial SPUD packet without payload and only send the full packet if the connection has be accepted by the receiver.

The firewall must still maintain a timer to delete the state of a tube if no packets were received for a while. However, if a end signal is received the firewall can remove the state information faster.

If a firewall receives a SPUD message which does not indicate the start of a new tube and no state is available for this tube, it may decide to block the traffic. This can happen if the state has already timed out or if the traffic was rerouted. In addition a firewall may send an error message to the sender or the receiver indicatng that no state information are available. If the sender receives such a message it can resend a start signal (potentially together with other tube state information) and continue its transmission.

2.4. Deployment Incentives

It is not expected that the provided SPUD information will enable all generic UDP-based services to safely pass firewalls , however, for new services that a firewall administrator is willing to allow, it makes state handling easier.

For application developers that actually would like to use a new transport services, there are today often only two choices; encapsulation over UDP or over TCP. SPUD already provides encapsulation over UDP as well as maintains (a few) additional information about the network state. This shim layer can support application developers to more easily implement new services.

2.5. Trust and Privacy

We proposed to limit the scope of the tube ID to the five-tuple. While this makes the tube ID useless for session mobility, it does mean that the valid ID space is sufficiently sparse to maintain the "hard to guess" property, and prevents tube IDs from being misused to track flows from the same endpoint across multiple addresses. This limitation may need further discussion.

By providing information on the connection start up, SPUD only exposes information that are often already given in the higher layer semantics. Thus it does not expose additional information, it only makes the information explicit and accessible without specific higher-layer/application-level knowledge.

3. State Lifetime Discovery

3.1. Problem Statement

Even if the transport protocol implements a close-down mechanism or SPUD explicitly provides an end of tube signal, a network device cannot assume that these signals are provided reliably. Therefore each network device that holds per-flow/per-tube state must implement a mechanism to remove the state if no traffic that is matching this state information has been observer for a while. Usually this is realized by maintaining a timeout since the last observed packet.

An endpoint that wants to keep a connection open even if it is not sending any data for a while might need to send heartbeat packets to keep state alive that potentially is store somewhere on the network path. However, the timeout period of the network device storing this information is unknow to the endpoint. Therefore it has to send heartbeat fairly rapidly, or might assume a default value of 150ms that is commonly used today.

3.2. Information Exposure

SPUD can be used to request the timeout used by a middlebox. As SPUD-enabled endpoint therefore sends a path-to-endpoint option that is initialized with an non-valid value (e.g. 0) and midpoints can update this information to the timeout value that is used to maintain

per-tube state. As multiple network devices might be on a path that maintain per-tube state, the timeout information should only be updated to the minimum value. A sender could also initial the timeout value to the minimum heartbeat frequency it will use or the maximum idle period (if known).

[Editor's note: Would it be necessary/useful to get a (separate) confirmation from each middlebox that has understood and read this SPUD information? Alternatively, it would maybe be useful signal the proposed heartbeat period separately, however that's also complicated because the endpoint might adapt it's heartbeat period based on the timeout information...]

3.3. Mechanism

If a network device that uses a timeout to remove per-tube state receives a SPUD timeout information request, it should expose its own timeout value if smaller than the one already given in the SPUD header. Alternatively, if a value is already given, it might decide to use the given value as timeout for the state information of this tube.

A SPUD sender can request the timeout used by network devices on path to maintain state. If a minimum heartbeat frequency is used or the maximum idle period is known, the sender might pre-set this value. If the pre-set value is not changed, the sender does not know if there is at least one SPUD-aware middlebox on the path that understands the time-out information. In any case a sender must always assume that there could be additional non-SPUD aware middlebox that has a smaller timeout. Therefore even if the proposed timeout is used for heartbeating, traffic can still be blocked due to removed state. This is also the case if a middlebox did not correctly indicate its timeout value, e.g. when the value is dynamically changed to a smaller value if more state needs to be maintained. However, usually the number of middleboxes on the path that hold per-flow/tube state is low. Therefore the chance that the received feedback indicates the right timeout value is high.

[Editor's note: Do we need a SPUD message that can be initialized by the middlebox to let the endpoint know that the time has changed?]

A SPUD endpoint receiving a SPUD header with timeout information should reflect this information to the sender with the next packet that it will be sent (or after a short timeout). Therefore this information should be requested with the first packet, that should immediately trigger the receiver to at least send one packet. In addition SPUD-aware nodes on the backward path are able to also signal their timeout.

[Editor's note: Is it necessary to have an explicit SPUD heartbeat packet, that should also be reflected by the receiver to keep state on the backwards path alive..? And then request timeouts for the forward and backward path separately?]

3.4. Deployment Incentives

Initially, if not widely deployed, there will be not much benefit to using this extension. However, an endpoint can never be sure that all middleboxes on the path that maintain state information based on a timeout will expose this information (correctly). An endpoint must always be prepared that traffic can be blocked (after an idle period) and the connection must be restarted. This is the same today if heartbeats are used. Therefore, SPUD will not help to simplify the implementation but it will also no make it much more complicated as only the heartbeat interval might be changed.

However, under the assumption that there are usually only a small number of middbleboxes on one network path that hold (per-tube) state information, it is likely that if information is exposed by a middlebox, this information is correct and can be used.

The more SPUD gets deployed, the more often endpoints will be able to set the heartbeat interval correctly. This will reduce the number of unnecessary reconnects that cause additional latency. Further, an endpoint might be able to request a higher timeout by pre-setting the value.

Network nodes that understand the SPUD timeout information and expose their timeouts are able to handle timeouts more flexibly, e.g. announcing lower timeout values if space is sparse. Further if an endpoint announces a low pre-set value because the endpoint knows that it will only have short idle periods, the timeout interval could be reduced.

3.5. Trust, Privacy and Security

[Editor's note: no trust needed here as discussed above... right? And I currently don't see privacy issues here...?']

[Editor's note: Make sure this is not a vector for simplified state exhaustion attacks...? Don't think it's worse than TCP...? Any other attacks?]

4. Low-Latency Service

4.1. Problem Statement

Networks are often optimized for low loss rates and high throughput by providing large buffers that can absorb traffic spikes or rate variations and always holding enough data to keep the link full. This is beneficial for applications like high-priority bulk transfer, where only the total transfer time is of interest. (High volume) interactive application, such as video calls, however, have very different requirements. Usually these application can tolerate high(er) loss rates, as they anyway cannot wait for missing data to be retransmitted, while having hard latency requirements necessary to make their service work.

Large network buffers may induce high queuing delays due to greedy cross traffic using loss-based congestion control that periodically fills the buffer. In loss-based congestion control the sending rate is periodically increased until a loss is observed to probe for available bandwidth. Unfortunately, the queuing delay that is indices by this probing can downgrade the quality of experience for competing interactive applications or even make them simply unusable. Further, to co-exist with greedy flows that use loss-based congestion control, one has to react based on the same feedback signal (loss) and implement about the same aggressiveness than these competing flows.

4.2. Information Exposure

While large buffers that are able to absorb traffic spikes that are often induced by short bursts are beneficial for some applications, the queuing delay that might be induced by these large buffers is very harmful to other applications. We therefore propose an explicit indication of loss- vs. latency-sensitivity per SPUD tube. This indication does not prioritize one kind of traffic over the other: while loss-sensitive traffic might face larger buffer delay but lower loss rate, latency-sensitive traffic has to make exactly the opposite tradeoff.

Further, an application can indicate a maximum acceptable single-hop queueing delay per tube, expressed in milliseconds. While this mechanism does not guarantee that sent packets will experience less than the requested delay due to queueing delay, it can significantly reduce the amount of traffic uselessly sitting in queues, since at any given instance only a small number of queues along a path (usually only zero or one) will be full.

4.3. Mechanism

A middlebox may use the loss-/latency-sensitive signal to assign packet to the appropriate service if different services are implemented at this middlebox. Today's traffic, that does not indicate a low loss or low latency preference, would still be assigned to today's best-effort service, while a new low latency service would be introduced in addition.

The simplest implementation of such a low latency service (without disturbing existing traffic) is to manage traffic with the latency-sensitive flag set in a separate queue. This queue either, in itself, provides only a short buffer which induces a hard limit for the maximum (per-queue) delay or uses an AQM (such as PIE/ CoDel) that is configured to keep the queuing delay low.

In such a two-queue system the network provider must decides about bandwidth sharing between both services, and might or might not expose this information. Initially there will only be a few flows that indicate low latency preference. Therefore at the beginning this service might have a low maximum bandwidth share assigned in the scheduler. However, the sharing ratio should be adopted to the traffic load/number of flows in each service class over time. This can be done manually by a network administrator or in an automated way.

Applications and endpoints setting the latency sensitivity flag on a tube must be prepared to experience relatively higher loss rates on that tube, and might use techniques such as Forward Error Correction (FEC) to cope with these losses.

If in addition the maximum per-hop delay is indicated by the sender, a SPUD-aware router might drop any packet which would be placed in a queue that has more than the maximum single-hop delay at that point in time before queue admission. Thereby the overall congestion can be reduced early instead of withdrawing the packet at the receiver after it has blocked network resources for other traffic. Alternatively, a SPUD-aware node might only remove the payload and add a SPUD error message, to report what the problem is.

An endpoint indicating the maximum per-hop delay must be aware that is might face higher loss rates under congestion than competing traffic on the same bottleneck. Especially, packets might be dropped due to the maximium per-hop delay indication before any congestion notification is given to any other competing flows on the same bottleneck. This should considered in the congestion reaction as any loss should be consider as a sign for congestion.

4.4. Deployment Incentives

Application developers go to a great deal of effort to make latency-sensitive traffic work over today's Internet. However, if large delays are induced by the network, an application at the endpoint cannot do much. Therefore applications can benefit from further support by the network.

Network operators have already realized a need to better support low latency services. However, they want to avoid any service degradation for existing traffic as well as risking stability due to large configuration changes. Introducing an additional service for latency-sensitive traffic that can exist in parallel to today's network service (or potentially fully replace today's service at some point in future...) helps this problem.

4.5. Trust and Privacy

An application does not benefit from wronly indicating loss- or latency-sensitivity as it has to make a tradeoff between low loss and potential high delay or low delay and potential high loss. Therefore there is no incentive for lying. A simple classification of traffic in loss-sensitive and latency-sensitive does not expose privacy-critical information about the user's behavior.

5. Application-Limited Flows

5.1. Problem Statement

Today, there are a large number of flows that are mostly application-limited, where the application can adapt this limit to changing traffic conditions. An example is unicast streaming video where the coding rate can be adapted based on detected congestion or changing link characteristics. This adaptation is difficult, since cross-traffic (much of which uses TCP congestion control) will often probe for available bandwidth more aggressively than the application's control loop. Further complicating the situation is the fact that rate adaptation may have negative effects on the user's quality of experience, and should therefore be done infrequently.

5.2. Information Exposure

With SPUD, the sender can provide an explicit indication of the maximum data rate that the current encoding needs. This can provide useful information to the bottleneck to decide how to correctly treat the corresponding tube, e.g. setting a rate limit or scheduling weight if served from its own queue.

Further, a network node that imposes rate shaping could expose the rate limit to the sender if requested. This would help the sender to choose the right encoding and simplifies probing. If the rate limited is changed the network node might want to signal this change without being requested for it.

In addition, both the endpoint as well as a middlebox could announce sudden changes in bandwidth demand/offer. While for the endpoint it might be most important to indicate that the bandwidth demand has increased, a middlebox could indicate if more bandwidth is (currently) available. Note that this information should only be indicated if the network node was previously the bottleneck/the out-going link is fully loaded. Further, if the information that bandwidth is available is provided to multiple endpoints at the same time, there is a higher risk of overloading the network as all endpoints might increase their rate at the same time.

[Editor's note: Should a middlebox even indicate how much capacity is available.. or 1/n of the available capacity if indicated to n endpoints? But there might be a new bottleneck now...]

5.3. Mechanism

If the maximum sending rate of a flow is exposed this information could be used to make routing decision, if e.g. two paths are available that have different link capacity and average load characteristics.

Further, a network nodes, that receives an indication of the maximum rate limit for a certain tube, might decide to threat this flow in an own queue and prioritize this flow in order to keep the delay low as long as the indicated rate limit is not exceeded. This should only be done if there is sufficient capacity on the link (the average load over a previous time period has be low enough to serve an additional maximum traffic load as indicated by the rate limit) or the flow is known to have priority, e.g. based on additional out-of-band signaling. If the link, however, is currently congested, a middlebox might choose to ignore this information or indicate a lower rate limit.

If a network node indicates rate shaping, this information can be used by the sender to choose its current data/coding rate appropriately. However, a sender should still implement a mechanism to probe ifor available bandwidth to verify the provided information. As a certain rate limit is expected the sender should probe carefully around this rate.

A network node might further indicate a different/lower rate limit during the transmission. However, in this case, it might be easy for an attacker to send a wrong rate limit, therefore an endpoint should not change its data rate immediately, but might be prepared to see higher losses rates instead.

If a sender receives an indication that more bandwidth is available it should not just switch to a higher rate but probe carefully. Therefore it might step-wise increase its coding rate or first add additional FEC information which will increase the traffic rate on the link and at the same time provide additional protection as soon as the new capacity limit is reached.

A network node that receives an indication that a flow will increase its rate abruptly, might prioritize this flow for a certain (short) time to enable a smoother transition. [Editor's node: Need to figure out if high loss/delay when the coding rate is increased is actually a problem and if so further evaluate if short-term prioritization helps.]

5.4. Deployment Incentives

By indicating a maximum sending rate a network operator might be able to better handle/schedule the current traffic. Therefore the network operator might be willing to support these kind of flows explicitly by trying to serve the flow with the requested rate. This can benefit the service quality and increase the user's satisfaction with the provided network service.

If the maximum sending rate is known by the application, the application might be willing to expose this information if there is a chance that the network will try to support this flow by providing sufficient capacity.

Currently application have no good indication when to change their coding rate. Especially, increasing the rate is hard. Further, it should be avoided to change the rate (forth and back) too often. An indication if and how much bandwidth is available, is therefore helpful for the application and can simplify probing (even though there will still and always be an additional control loop needed to react to congestion and for probing).

5.5. Trust, Privacy and Security

[TBD] [Editor's note: is there an attack possible by indicating a low limit (from or to the application)? Note, that the application should not rely on this information and still probe for more capacity (if needed) and react to congestion!]

6. Service Multiplexing

6.1. Problem Statement

Many services rewuire multiple parallel transmissions to transfer different kinds of data which usually have a clear priority between each other. One example is WebRTC where the audio is most important and should be higher prioritized than the video, while control traffic might have the lowest priority. Further, some packets within one flow might be more important than others within the same flow/ tube, e.g. such as I-frames in video transmissions. However, today a network will treat all packets the same in case of congestion and might e.g. drop audio packets while video and control traffic are still transmitted.

6.2. Information Exposure

A SPUD sender may indicate a lower priority relative to another tube that is used in the same 5-tuple.

Similarly, a lower packet priority within one flow/tube could be indicated to give one packet a low priority than other packets with the same tube ID. This information can be used to preferentially drop less inportant packets e.g. carrying information that could be recovered by FEC or where missing data can be easily concealed.

Further, with a stronger integration of codec and transport technology SPUD could even indicate more even finer grained priority levels to provide automatic graceful degradation of service within the network itself.

[Editor's note: do we want to also provide per-packet information over spud? Or would all lower priority packets of one flow simply below to a different tube? In this case can we send a SPUD start message with more than on tube ID?]

6.3. Mechanism

Preferential dropping can be implemented by a router queue in case packets need to be dropped due to congestion. In this case the router might not drop the incoming packet but look for a packet with the same tube ID that is already in the queue and has a lower priority than to actual packet that should have been dropped. Note that a middlebox should only drop a different packet if there is currently a lower priority packet in the queue, because it otherwise does not know whether it will every see a lower priority packet for this flow. This could cause unfairness issues. Therefore a middlebox might need to hold additional state, e.g. keeping position

of the last low priority packet of each tube in a separate table. The chance that a low priority packet of the same or corresponding tube currently sits in the queue, is lower the smaller the buffer is. Therefore for low-latency, real-time services, there is a tradeoff.

Alternatively, the middlebox might queue the lower priority traffic in a different queue. Using a different queue might be suitable for lower flow priority but should not be used for lower priority packets within the same flow as this can also lead to other issues such as high reordering. Further, using a lower priority queue will not only give higher priority to the traffic belong to the same service/sender but also to all other competing flows. This is usually not the intention.

[Editor's note: Does it makes sense to, in addition, rate-limit the higher prirority flows to their current rate to make sure that the bottleneck is not further overloaded...?]

If a sender has indicated lower priority to certain tubes and only experiences losses/congestion for the lower priority tubes, the sender should still not increase its sending for the higher priority tube and might even consider to decrease the sending rate for the higher prioroty tubes as well. Potentially a (delay-based) mechanism for shared bottleneck detection should be used to ensure that all transmissions actually share the same bottleneck.

6.4. Deployment Incentives

[Editor's note: similar as above -> support of interactive services increases costumer satisfaction...]

6.5. Trust and Privacy

As only lower priority should be indicated, it is harder to use this information for an attack.

[Editor's note: Do not really see any trust or privacy concerns here...?]

7. Acknowledgements

This document grew in part out of discussions of initial use cases for middlebox cooperation at the IAB SEMI Workshop and the IETF 92 SPUD BoF; thanks to the participants.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

Security and privacy considerations for each use case are given in the corresponding subsection.

10. Informative References

[I-D.trammell-spud-req]
Trammell, B. and M. Kuehlewind, "Requirements for the design of a Substrate Protocol for User Datagrams (SPUD): draft-trammell-stackevo-spud-req-00 (To be published soon)", 2015.

Authors' Addresses

Mirja Kuehlewind
ETH Zurich
Zurich, Switzerland

Email: mirja.kuehlewind@tik.ee.ethz.ch

Brian Trammell
ETH Zurich
Zurich, Switzerland

Email: ietf@trammell.ch