

IPPM Considerations for the IPv6 PDM Destination Option

Nalini Elkins – Inside Products, Inc.

Mike Ackermann – BCBS Michigan

Rob Hamilton – Chemical Abstracts

Questions from IETF91

1. Does PDM have enough variables to actually diagnose problems?
2. Are all PDM fields necessary?
3. Why is the proposal for an IPv6 extension header rather than a TCP option? Only TCP is important.
4. Does PDM create too much overhead?
5. Will PDM work for complex apps not just simple applications with one send and one receive?

Enough Variables?

- Question:
 - Does PDM have enough variables to actually diagnose problems?
- Answer:
 - PDM works WITH other fields in headers
 - PDM ADDs data which is not easy to get
 - Technician will be looking at tool such as Wireshark (or add-on)

Too Many Fields?

- Question:
 - Are all PDM fields necessary?
- Answer:
 - Yes.
 - (Explained in answer to next question)

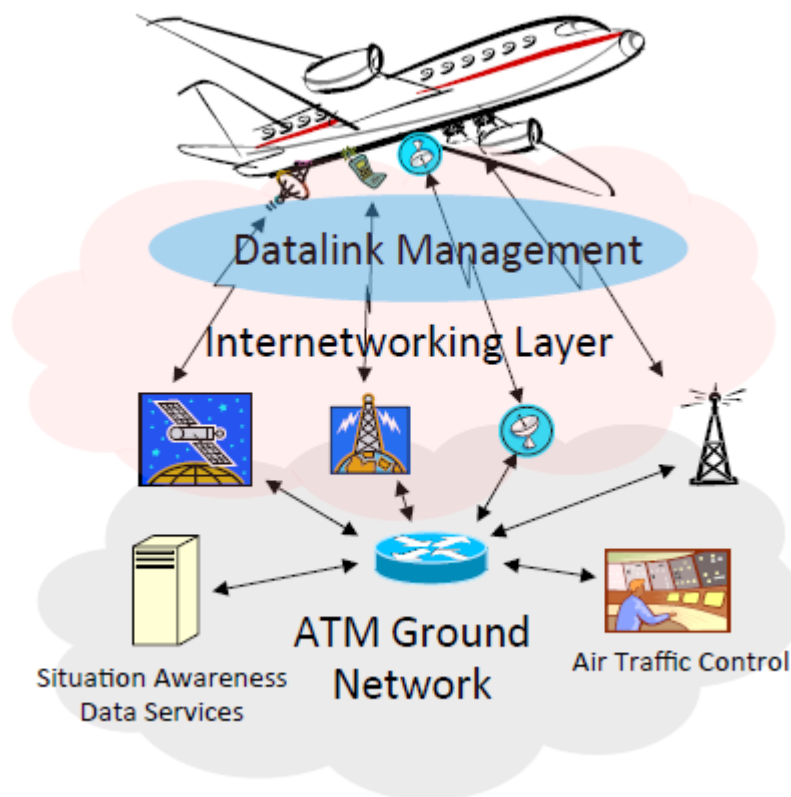
Why IPv6 Extension Header?

- Question:
 - Why is the proposal for an IPv6 extension header rather than a TCP option? Only TCP is important.
- Answer:
 - Large enterprises have traffic which is non-TCP which will benefit from PDM.
 - Non-TCP traffic includes:
 - IBM's Enterprise Extender, which allows its SNA Peer-to-Peer Networking (APPN) traffic flow over UDP links
 - Some WWW traffic flows as UDP packets
 - TFTP, SNMP, DNS, OSPF, etc.
 - Other/new upper layer protocols (e.g. the new Frame Control Protocol)
 - TCP applications will also benefit from PDM.

From Boeing

From IETF 91: IPv6GEO – GEO Information in IPv6 Packet Headers
<http://www.ietf.org/proceedings/91/slides/slides-91-6man-8.pdf>

- Aircraft have many links with varying cost, performance, availability profiles.
- Not all links available during all phases of flight.
- Not all links encode geo information at the link-layer
- Wide variety of applications – not all of which are geo-aware
- **IPv6 layer is only commonality**



Too Much Overhead?

- Total PDM size is only 16 bytes. 16 (optional) bytes should NOT be an issue on today's high speed networks.

Potential overhead

(additional time to deliver the response to the end user)

Packet Bytes in Packet	RTT	Bytes Per Milli	Bytes in PDM	New RTT	Overhead
1000	1000 milli	1	16	1016.000	16.000 milli
1000	100 milli	10	16	101.600	1.600 milli
1000	10 milli	100	16	10.160	.160 milli
1000	1 milli	1000	16	1.016	.016 milli

Actual RTTs

Packets going to multiple business partners

Packet Bytes in Packet	RTT	Bytes Per Milli	Bytes in PDM	New RTT	Overhead
1000	17 milli	58	16	17.360	.360 milli

Packets within a data center

(Notice that the scale is now in microseconds not milliseconds)

Packet Bytes in Packet	RTT	Bytes Per Micro	Bytes in PDM	New RTT	Overhead
1000	20 micro	50	16	20.320	320 micro

Note: Both examples are for large enterprises

Only for Simple Apps?

- Question
 - Will PDM work for complex apps not just simple applications with one send and one receive.
- Answer
 - Not at all.
 - Examples follow.

One-Way Flow

Packet	Sender	PSN This Packet	PSN Last Recvd	Delta Last Recvd	Delta Last Sent
1	Server	1	0	0	0
2	Server	2	0	0	5
3	Server	3	0	0	12
4	Server	4	0	0	20

In a one-way flow, only the Delta Last Sent will be seen as used. Recall, Delta Last Sent is the difference between the send of one packet from a device and the next. This is a measure of throughput for the sender - according to the sender's point of view. That is, it is a measure of how fast is the application itself (with stack time included) able to send packets.

How might this be useful? If one is having a performance issue at the client and sees that packet 2, for example, is sent after 5 microseconds from the server but takes much longer to arrive at the destination (deduced from other fields in the packet) then one may safely conclude that there are delays in the path other than at the server which may be causing the delivery issue of that packet. Such delays may include the network links, middle-boxes, etc.

Multiple Send Flow

Assume that two packets are sent with each send from the server.

Packet	Sender	PSN This Packet	PSN Last Recvd	Delta Last Recvd	Delta Last Sent
1	Server	1	0	0	0
2	Server	2	0	0	5
3	Client	1	1	20	0
4	Server	3	1	10	15

Notice that in packet 3, the client has a value of Delta Last Received of 20. Recall that Delta Last Received is the Send time of packet 3 - receive time of packet 2. So, what does one know now? In this case, Delta Last Received is the processing time for the Client to send the next packet.

How to interpret depends on what is actually being sent. Remember, PDM is not being used in isolation, but to supplement the fields found in other headers.

Examples

- Client is sending a standalone TCP ACK. One would find this by looking at the payload length in the IPv6 header and the TCP Acknowledgement field in the TCP header. So, in this case, the client is taking 20 units to send back the ACK. This may or may not be interesting.
- Client is sending data with the packet. Again, one would find this by looking at the payload length in the IPv6 header and the TCP Acknowledgement field in the TCP header. So, in this case, the client is taking 20 units to send back data. This may represent "User Think Time". Again, this may or may not be interesting, in isolation. But, if there is a performance problem receiving data at the server, then taken in conjunction with RTT or other packet timing information, this information may be quite interesting.

Benefit of PDM

- Of course, one also needs to look at the PSN Last Received field to make sure of the interpretation of this data. That is, to make sure that the Delta Last Received corresponds to the packet of interest.
- The benefits of PDM are that we have such information available in a uniform manner for all applications and all protocols without extensive changes required to applications.

Multiple Send with Errors

- Are the functions of PDM better suited to TCP or a TCP option? Let us take the case of how PDM may help in a case of TCP retransmissions in a way that TCP options or TCP ACK / SEQ would not.
- Assume that three packets are sent with each send from the server.
- From the server, this is what is seen:

Pkt	Sender	PSN This Pkt	PSN LastRecvd	Delta LastRecvd	Delta LastSent	TCP SEQ	Data Bytes
1	Server	1	0	0	0	123	100
2	Server	2	0	0	5	223	100
3	Server	3	0	0	5	333	100

At Client

- The client however, does not get all the packets. From the client, this is what is seen for the packets sent from the server.

Pkt	Sender	PSN This Pkt	PSN LastRecvd	Delta LastRecvd	Delta LastSent	TCP SEQ	Data Bytes
1	Server	1	0	0	0	123	100
2	Server	3	0	0	5	333	100

- Notice that the packet with PSN = 2 from the server is not received

Server Retransmits

- Let's assume that the server now retransmits the packet. (Obviously, a duplicate acknowledgment sequence for fast retransmit or a retransmit timeout would occur. To illustrate the point, these packets are being left out.)
- So, then if a TCP retransmission is done, then from the client, this is what is seen for the packets sent from the server.

Pkt	Sender	PSN This Pkt	PSN LastRecvd	Delta LastRecvd	Delta LastSent	TCP SEQ	Data Bytes
1	Server	4	0	0	30	223	100

- The server has resent the old packet 2 with TCP sequence number of 223. The retransmitted packet now has a PSN This Packet value of 4.
- The Delta Last Sent is 30. That is the time between sending the packet with PSN of 3 and this current packet.

Server Retransmits AGAIN

- Let's say that packet 4 STILL does not make it. Then, after some amount of time (RTO) then the packet with TCP sequence number of 223 is resent.
- From the client, this is what is seen for the packets sent from the server.

Pkt	Sender	PSN This Pkt	PSN LastRecvd	Delta LastRecvd	Delta LastSent	TCP SEQ	Data Bytes
1	Server	5	0	0	60	223	100

TCP SEQ Unhelpful

- If now, this packet makes it, one has a very good idea that packets exist which are being sent from the server as retransmissions and not making it to the client. This is because the PSN of the resent packet from the server is 5 rather than 4. If we had used TCP sequence number alone, we would never have seen this situation. Because the TCP sequence number in all situations is 223.
- This situation would be experienced by the user of the application (the human being actually sitting somewhere) as a "hangs" or long delay between packets. On large networks, to diagnose problems such as these where packets are lost somewhere on the network, one has to take multiple traces to find out exactly where.

Only Client Trace Needed

- The first thing is to start with doing a trace at the client and the server. So, we can see if the server sent a particular packet and the client received it. If the client did not receive it, then we start tracking back to trace points at the router right after the server and the router right before the client. Did they get these packets which the server has sent? This is a time consuming activity.
- With PDM, we can speed up the diagnostic time because we may be able to use only the trace taken at the client to see what the server is sending.

Appendix

- Presentation from IETF 91
- Explanation of PDM

We propose:

Requirement

- In basic IP transport
- Undisturbed by middle systems

Solution

- **Implementation** of existing extension header: Destination Options Header (DOH)
- Performance and Diagnostic Metrics (PDM) DOH

PDM

- Performance and Diagnostic Metrics Destination Option (PDM) contains the following fields: (by 5-tuple)
- PSNTP : Packet Sequence Number This Packet
- PSNLR : Packet Sequence Number Last Received
- DELTALR : Delta Last Received
- DELTALS : Delta Last Sent
- TIMEBASE : Base timer unit
- SCALEDL : Scale for Delta Last Received
- SCALEDS : Scale for Delta Last Sent

PDM Timing

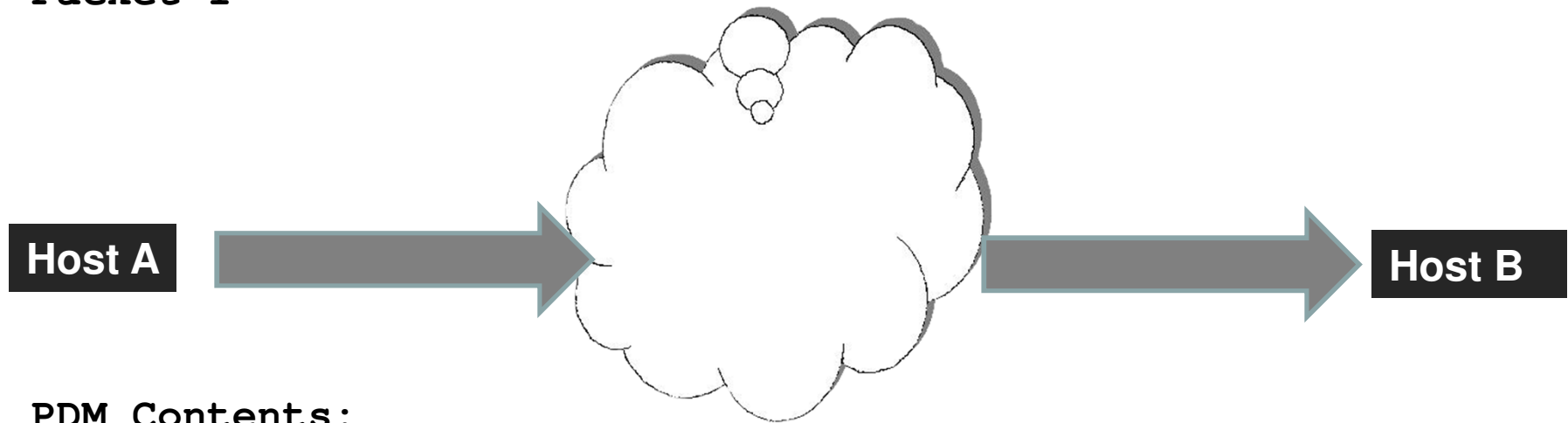
- No time synchronization needed
- All times are in relation to self

Start Flow

- Packet 1 is sent from Host A to Host B. The time for Host A is 10:00AM.
- The time and packet sequence number are saved by Host A internally. The packet sequence number and delta times are sent in the packet.

Packet 1

Packet 1



PDM Contents:

PSNTP	: Packet Sequence Number This Packet:	25
PSNLR	: Packet Sequence Number Last Received:	-
DELTALR	: Delta Last Received:	-
DELTALS	: Delta Last Sent:	-

Keep in Host A

- Internally, within the sender, Host A, it must keep:
- Packet Seq. Number of last packet sent: 25
- Time the last packet was sent: 10:00:00

Keep in Host B

- Packet 1 is received at Host B. Its time is set to one hour later than Host A. In this case, 11:00AM
- Internally, within the receiver, Host B, it must note:
- Packet Seq. Number of last packet received: 25
- Time the last packet was received : 11:00:03

Server Delay

- Host B processes packet 1 and creates a response (packet 2).
- Packet 2 is sent by Host B to Host A.
- This is the time taken by Host B or Server Delay
- $\text{Server Delay} = \text{Sending time (packet 2)} - \text{receive time (packet 1)}$

DeltaLR

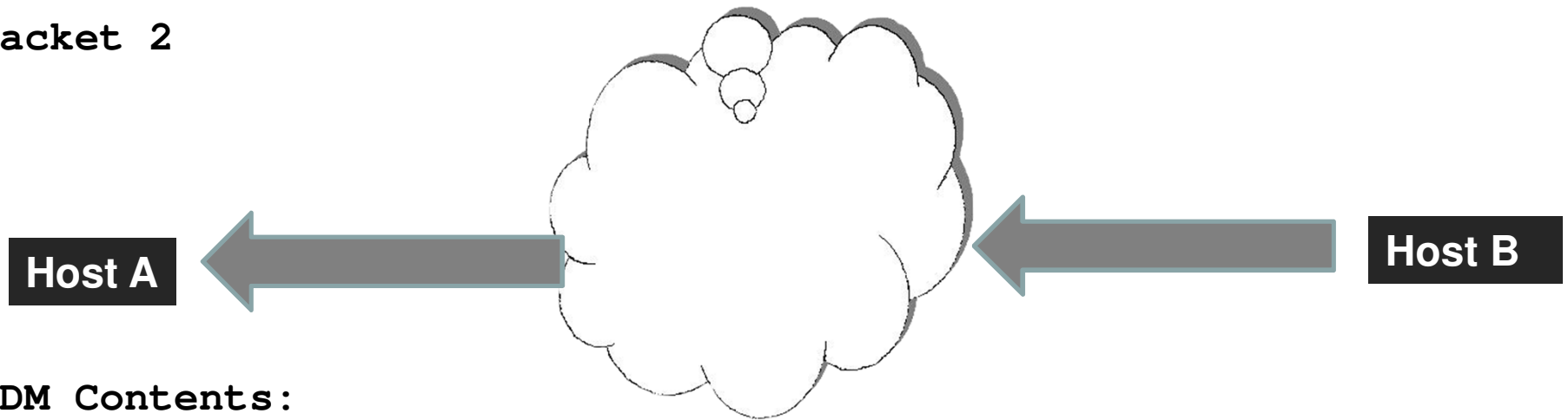
- We will call the result of this calculation: Delta Last Received
- $\text{DELTA LR} = \text{Sending time (packet 2)} - \text{receive time (packet 1)}$
- Note, both sending time and receive time are saved internally in Host B. They do not travel in the packet. Only the Delta is in the packet.

Host B Stats

- Within Host B is the following:
- Packet Sequence Number of the last packet received: 25
- Time the last packet was received: 11:00:03
- Packet Sequence Number of this packet: 12
- Time this packet is being sent: 11:00:07
- DELTALR = 4 seconds (11:00:07 - 11:00:03)
- DELTALR is Server Delay.

Packet 2

Packet 2



PDM Contents:

PSNTP	: Packet Sequence Number This Packet:	12
PSNLR	: Packet Sequence Number Last Received:	25
DELTALR	: Delta Last Received:	4 seconds
DELTALS	: Delta Last Sent:	-

Metrics Needed

- The metrics left to be calculated are end-to-end time and round-trip delay (network time).
- This will be calculated by Host A when it receives Packet 2.

Packet 2 Received

- Packet 2 is received at Host A. Remember, its time is set to one hour earlier than Host B. Internally, it must note:
- Packet Sequence Number of the last packet received: 12
- Time the last packet was received : 10:00:12
- Note, this timestamp is in Host A time. It has nothing whatsoever to do with Host B time.

End-to-End Time

- Now, Host A can calculate total end-to-end time.
- End-to-End Time = Time Last Received - Time Last Sent
- Packet 1 was sent by Host A at 10:00:00. Packet 2 was received by Host A at 10:00:12
- End-to-End time = 10:00:12 - 10:00:00 or 12
- This metric we will call DELTALS or Delta Last Sent

Network Time

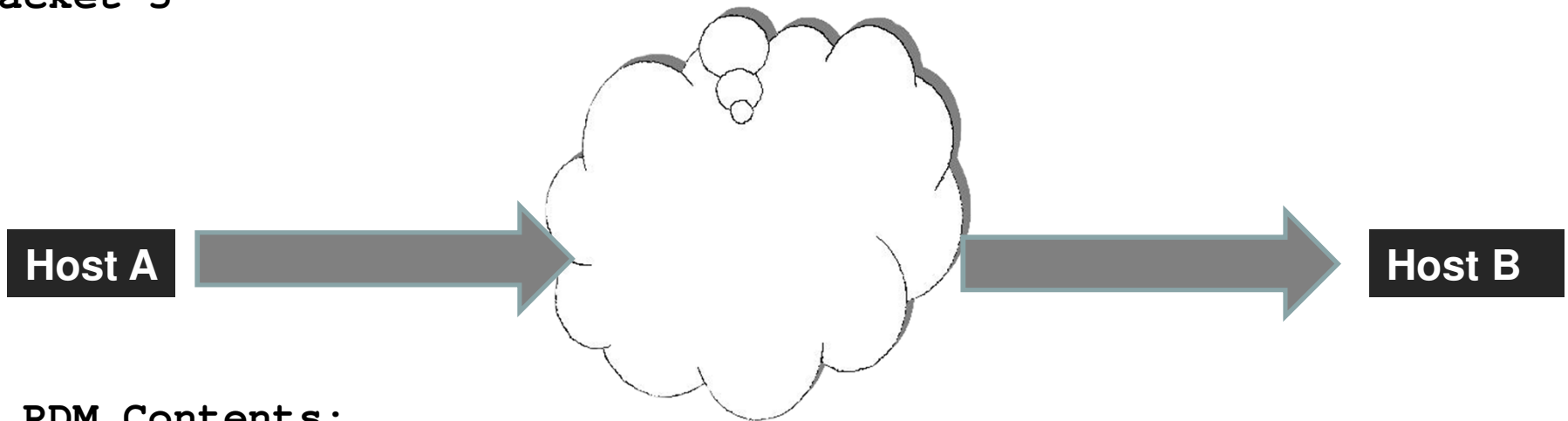
- We can now also calculate round trip delay (network time). The formula is:
- Round trip delay = DELTALS - DELTALR
- Or: End-to-end time – Server Delay
- Round trip delay = 12 - 4 or 8

How to Communicate?

- Now, the only problem is that at this point all metrics are in Host A only and not exposed in a packet.
- To do that, we need a third packet.

Packet 3





























Packet 3



PDM Contents:

PSNTP	: Packet Sequence Number This Packet:	26
PSNLR	: Packet Sequence Number Last Received:	12
DELTALR	: Delta Last Received:	0
DELTALS	: Delta Last Sent:	12 seconds

Show IPv6 PDM Type 16 Header
 Using: Trace File:pdm16
 Sort Order : Packet Number

-	-	Packet Number	Packet Date	Extension Header	Source Address	Destination Address	This Packet ID	Packet Last Received	Delta Last Received (Microseconds)	Delta Last Sent (Microseconds)
1		4	2014-01-10 13:30:22.857512	60	2001::2	2001::1 	0	0	0	0
2		5	2014-01-10 13:30:22.860452	60	2001::1	2001::2 	0	0	0	0
3		6	2014-01-10 13:30:23.865714	60	2001::2	2001::1 	1	0	1006	983
4		7	2014-01-10 13:30:23.877588	60	2001::1	2001::2 	1	0	1017	1017
5		8	2014-01-10 13:30:24.870476	60	2001::2	2001::1 	2	1	1008	974
6		9	2014-01-10 13:30:24.871949	60	2001::1	2001::2 	2	1	994	994
7		13	2014-01-10 13:30:25.879201	60	2001::2	2001::1 	3	2	1005	995
8		14	2014-01-10 13:30:25.88565	60	2001::1	2001::2 	3	2	1013	1013
9		17	2014-01-10 13:30:26.886962	60	2001::2	2001::1 	4	3	1008	985
10		18	2014-01-10 13:30:26.897091	60	2001::1	2001::2 	4	3	1011	1011
11		19	2014-01-10 13:30:27.891001	60	2001::2	2001::1 	5	4	1007	974
12		20	2014-01-10 13:30:27.901722	60	2001::1	2001::2 	5	4	1004	1004
13		27	2014-01-10 13:30:28.894605	60	2001::2	2001::1 	6	5	1004	982
14		28	2014-01-10 13:30:28.905136	60	2001::1	2001::2 	6	5	1003	1003

```
.... 0 .... = IG bit: Individual address (unicast)
Type: IPv6 (0x86dd)
Internet Protocol Version 6, Src: 2001::2 (2001::2), Dst: 2001::1 (2001::1)
0110 .... = Version: 6
  [0110 .... = This field makes the filter "ip.version == 6" possible: 6]
.... 0000 0000 .... = Traffic class: 0x00000000
  .... 0000 00.. .... = Differentiated Services Field: Default (0x00000000)
  .... ..0. .... = ECN-Capable Transport (ECT): Not set
  .... ..0 .... = ECN-CE: Not set
.... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
Payload length: 71
Next header: IPv6 destination option (60)
Hop limit: 64
Source: 2001::2 (2001::2)
[Source Teredo Server IPv4: 0.0.0.0 (0.0.0.0)]
[Source Teredo Port: 65535]
[Source Teredo Client IPv4: 255.255.255.253 (255.255.255.253)]
Destination: 2001::1 (2001::1)
[Destination Teredo Server IPv4: 0.0.0.0 (0.0.0.0)]
[Destination Teredo Port: 65535]
[Destination Teredo Client IPv4: 255.255.255.254 (255.255.255.254)]
[Source GeoIP: Unknown]
[Destination GeoIP: Unknown]
Destination Option
  Next header: TCP (6)
  Length: 2 (24 bytes)
  IPv6 Option (PadN)
    Type: PadN (1)
    Length: 6
    PadN: 000000000000
  IPv6 Option (Unknown 16)
    Type: Unknown (16)
    Length: 14
    Unknown Option Payload: 000103000003ee000003d700b326
Transmission Control Protocol, Src Port: 45862 (45862), Dst Port: 80 (80), Seq: 452248946, Len: 7
Source port: 45862 (45862)
Destination port: 80 (80)
[Stream index: 2]
```

Breakout in
WireShark



Timebase

- Possible values of Time Base:
 - 00 - milliseconds
 - 01 - microseconds
 - 10 - nanoseconds
 - 11 - picoseconds

Scale (DLR / DLS)

- 7-bit signed integer.
- Possible values from -64 to +63.
- Store most significant bits of timer value along with a scaling factor to indicate the magnitude.
- High-order 16 bits.