AREDN network storms

What they look like, how they happen, and how to prevent them

The SFWEM saw a series of message storm events on Saturday, 28th Aug, 2021. This document describes what they look like, how they happen, and how to prevent them.

TL;DR;

Network storms are the result of a defect in the OLSR protocol's detection of duplicate messages. Each OLSRD's messages are sequenced with a 16 bit, modular, incrementing number, starting from a random value chosen at OLSRD's startup. Sequence numbers are used to suppress duplicate packets in the OLSR mesh. Each OLSRD tracks all active nodes (plus an additional 5 minutes once a node becomes inactive) and their most recent sequence number. When an OLSRD sees 16 (DUP_MAX_TOO_LOW) sequential packets 32 below the last highest sequence number from a node, it considers the new lowest sequence number valid, abandoning the higher one. It also considers any sequence number 32767 higher than the last seen sequence number valid. If the correct arrangement of packets are in flight in the OSLR mesh (when an instance of OLSRD restarts), duplicate detection will fail, resulting in the acceptance of alternating sequence numbers as non-duplicates. Each OLSRD instance will rebroadcast these packets for each of its interfaces, and the packets will continue circulation until their TTL counter expires. Each restart of OLSR, and natural sequence number arrival jitter, may cause further duplication.

Assumptions

1. This document only discusses messages and packets generated and managed by the OLSR protocol. Routing and management of "user" traffic is not part of this discussion.
2. OLSR networks consist of a mesh of nodes, connected via wireless, tunnel, or ethernet. OLSR makes no distinction between these types of connections beyond some optimizations which are not material here.
3. OLSR networks contain loops which are necessary to provide redundant paths between nodes with unreliable network connections. This is normal.
4. Examples have been simplified for ease of explanation.
5. We ignore the MPR part of OLSR which is used to reduce the flooding of messages to neighbors. They make no material difference to the problem, but make it harder to explain.

Storm packets

During a storm, we see many large packets contain many messages all from a single originator. The messages (types MID and HNA) have high time-to-live and are broadcast to the network. A problem packet will contain 3 or more messages, some with vastly different sequence numbers.

Example 1:

MID: SeqNr 0x1000	MID: SeqNr 0x7000	MID: SeqNr 0xD000

Or Example 2:

MID: SeqNr 0x1000	MID: SeqNr 0x8FFF	MID:SeqNr 0x9002

These messages are all valid according to the OLSR message deduplication code as each sequence number is greater than the one before. However, because sequence numbers will wrap around (they're only 16-bits) the algorithm uses modular arithmetic and also considers the first message to have a greater sequence number than the last. If this packet is received twice by the same node, the messages will be considered valid and unique both times. The packets containing these message sequences are "immortal" and will not be culled by the network until their time-to-live hits zero.

Every time a node receives such a sequence of messages, it will incorrectly identify them as new. As these are broadcast messages, it will then forward them on to every neighboring node. These neighboring nodes will do the same thing in turn, flooding the network with many duplicate copies of the same messages.

If the mesh network was a directed acyclic graph these messages would eventually reach the edge and be discarded. But the mesh network is not, and has many redundant links between groups of nodes which form cycles. Generally this is a fine thing as it greatly improves network reliability. However these immortal packets will continually traverse these cycles, being duplicated by every node again and again.

The storm will eventually end when the time-to-live of all copies of all of the messages hits 0. As every node duplicates these packages everytime they see them, and no duplicates are ever removed, this can take a while.

Why they happen

For an immortal packet to form, the network needs to contain packets with at least three messages where each sequence number is never more than 0x7FFF from the previous one. Sequence numbers are generated sequentially starting from a random base set when OLSRD starts, so there are only two ways to generate large differences in sequence numbers for messages in a network:

1. OLSRD generates a lot of messages in a short period of time
2. OLSRD resets the sequence number base.

OLSRD generates very few sequence numbers per second, perhaps only 10 to 20 (largely dependent on the number links to neighboring nodes). If we assume that messages stay in the network for 10 seconds (which is high) we would need to generate thousands of messages per seconds to have a chance of producing problematic sequence numbers. And even if we did that, we'd have a nice clean progression rather than the big holes we see in the immortal packets (so option 1 seems unlikely). Given this, OLSRD must be resetting its sequence number base (option 2). This happens when the OLSRD daemon restarts.

OLSRD restart

OLSRD will restart under various circumstances:

1. A node is rebooted
2. A tunnel is enabled/disabled
3. A service is added/removed/changed
4. A watchdog restarts OLSRD
5. OLSRD crashes

Any one of these events will restart the sequence numbers, and the more of these events which occur close to each other, the more likely a problem is to develop. Because the sequence number base is random, the chance that any one of these starts a flood is low, but over time this is bound to happen.

Mitigation

Here are some ways the problem can be mitigated and/or solved. Some are more involved than others:

1. Don't randomly restart the sequence numbers when restarting OLSRD. By always counting sequentially without gaps the problem will never happen.

   Keeping persistent track of the current sequence number across restarts/crashes would require some data being persisted to the file system (and continually writing to flash is never great). We could just update with sequence number N every X numbers, and restart with N + X. Alternatively, we could write this value to a temporary file (stored in RAM). A restart of OLSRD would pick up this value, while a reboot of a node would take "long enough" to allow old messages to drain from the network so a specific starting value is less important.

2. Delay the start/restart of the OLSRD daemon. Simply wait a set period of time before allowing OLSRD to rejoin the network.

   Nodes cache sequence number information for other nodes for 288 seconds (about 5 minutes) after they last saw them, so that might be a good amount of time. Alternatively, it might be sufficient to wait for all messages to have been routed through the network (30 seconds perhaps).

3. Improve the dedup code. Improve the OLSRD dedup code to catch these problems.

   We could narrow the windows of acceptable sequence numbers. That would make it much harder - almost impossible depending on how wide the acceptable window is - to generate a set of sequence numbers caused by restarts which would produce this problem.

4. Listen before joining the network. Improve OLSRD so it listens to make sure it can no longer hear packets from an older instance of itself. This assumes it will hear itself however.

5. Remove repeated messages from the currently assembling packet when only the sequence number is different.

   Even at the best of times, a package can contain the same message contents that it received from different neighbors (due to how flooding of messages work). We could remove these duplicates, although some care here would be necessary to make sure removing duplicates does not change the meaning of anything. This would help, but it's not clear it would completely solve the problem.

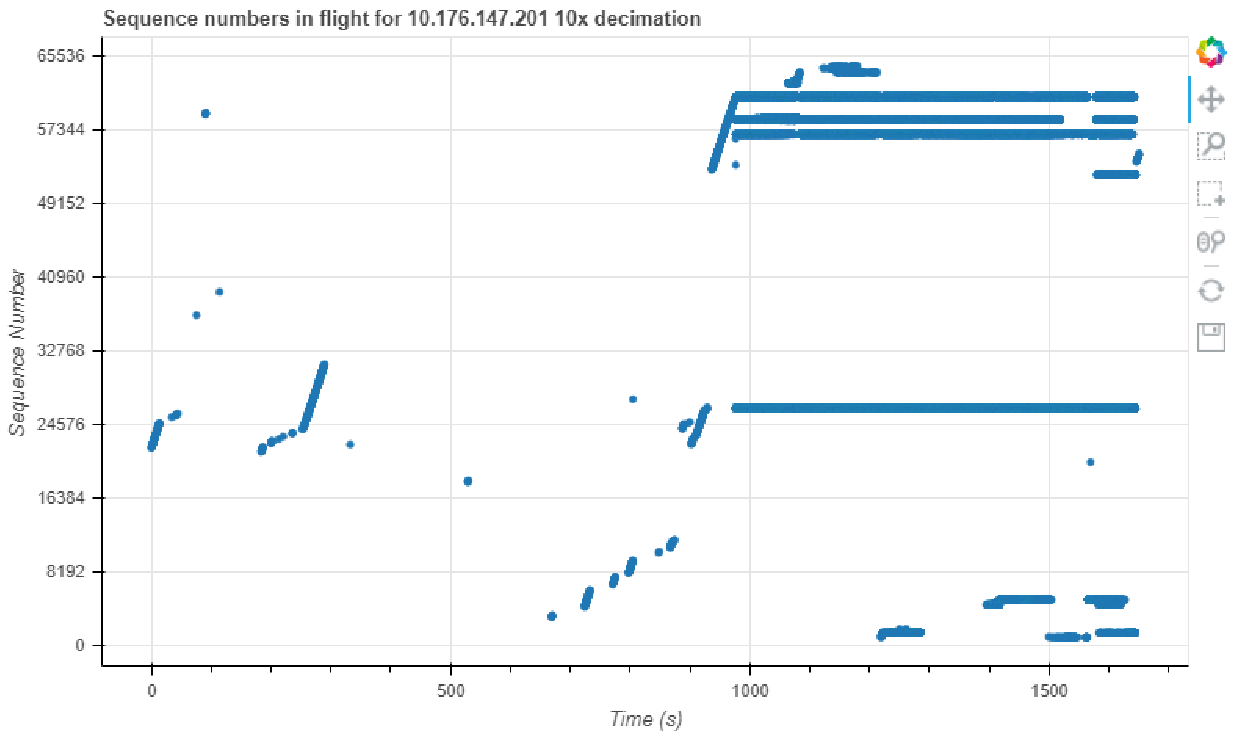
Authors and Contributors

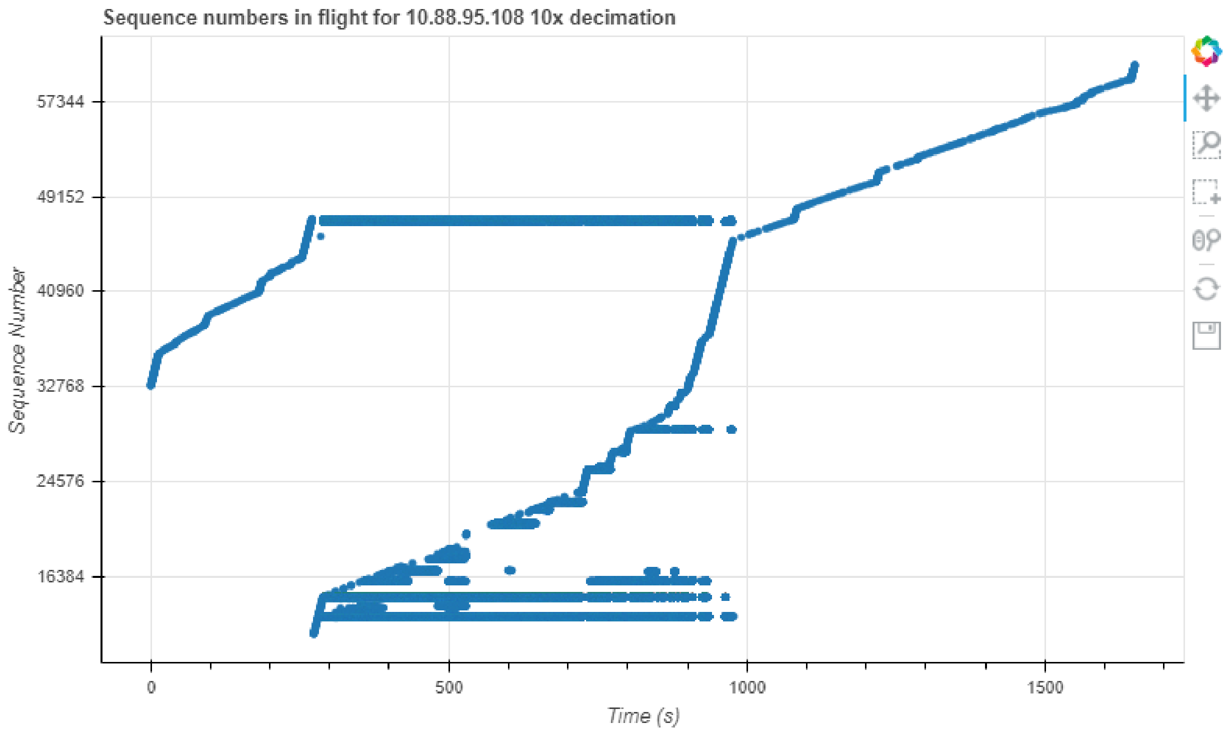
- Tim - KN6PLV (tim.j.wilkinson@gmail.com)
- Pascal - KN6GFJ (pascal@wassamlabs.com)

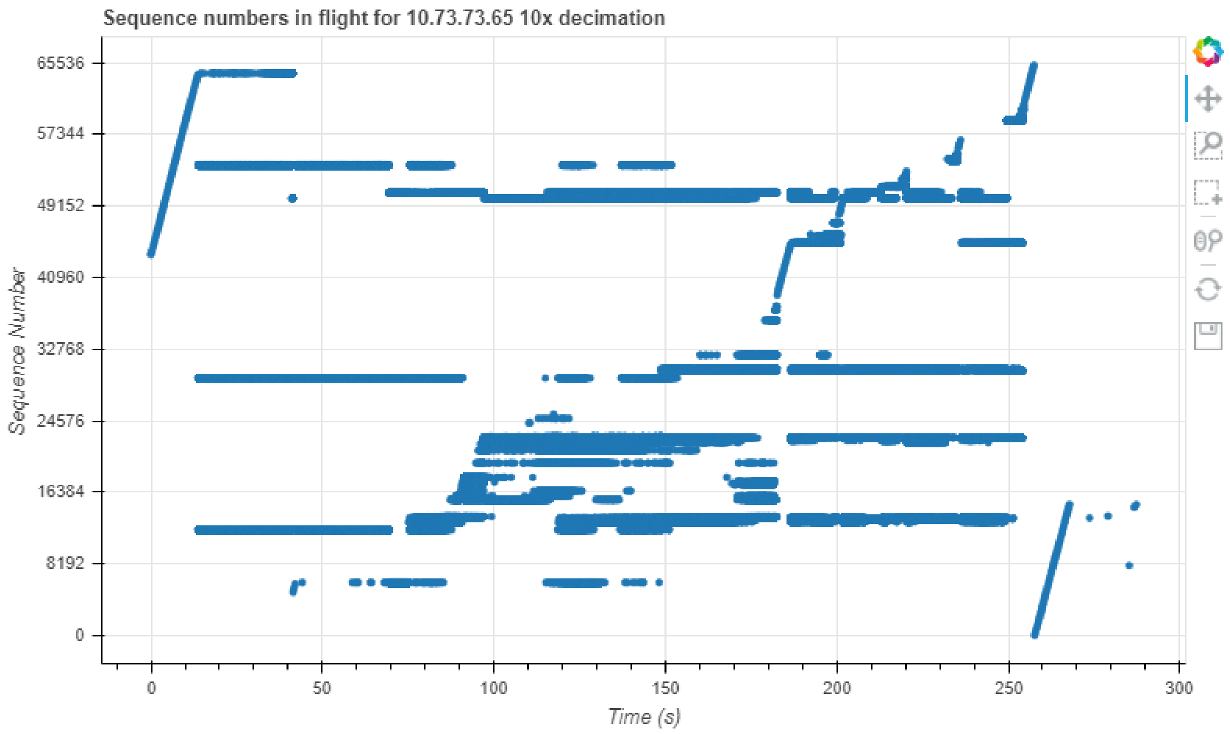
Appendix:

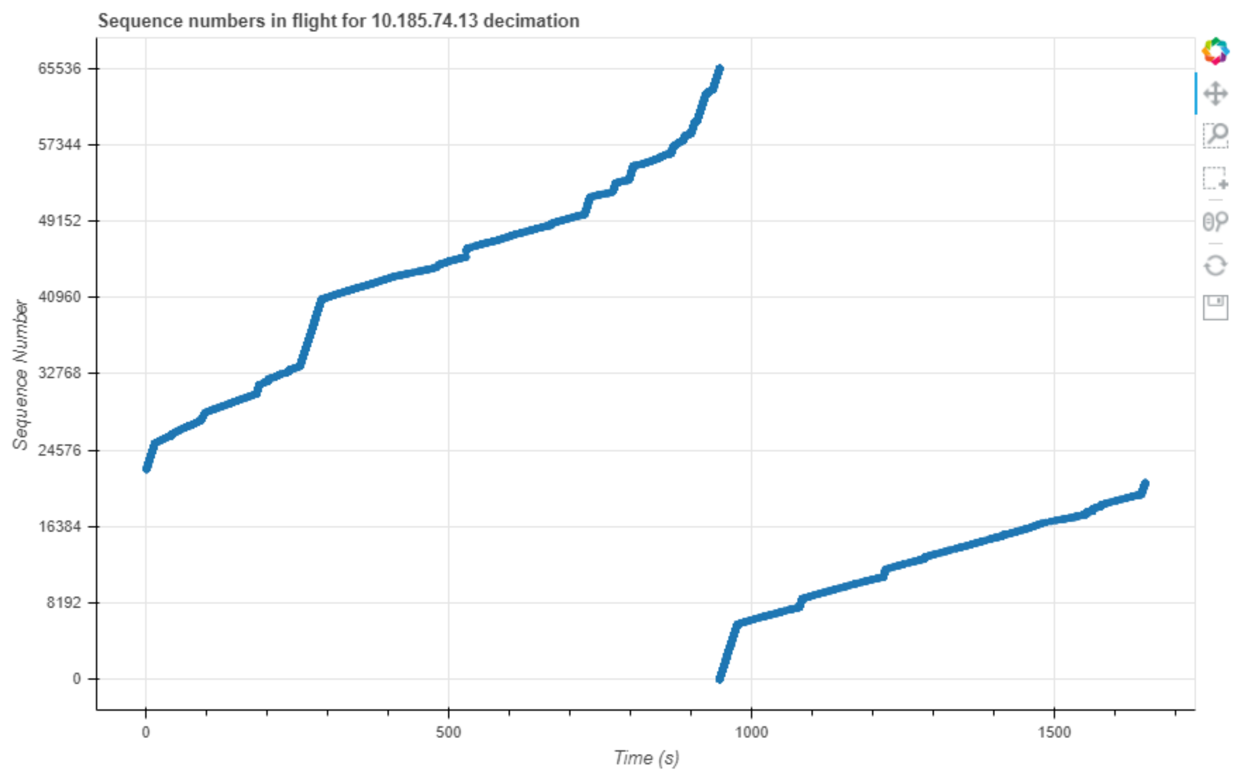
The following are plots from the SFEWM storm on Aug 28 2021. These are plots of all observed olsrd message sequence numbers from a given host as recorded from a single AREDN node. The first 3 hosts selected were all observed participating in storm behavior. These plots are decimated 10x, so small transient events may not be visualized. Not visible at this level of detail is that some of the solid horizontal lines consist of a few lines within a few sequence numbers.

The final plot is an example of a normally behaving host.




Sequence numbers in flight for 10.88.95.108 10x decimation
Sequence Number
57344
49152
40960
32768
24576
16384
Time (s)
0
500
1000
1500


Sequence numbers in flight for 10.73.73.65 10x decimation
Sequence Number
65536
57344
49152
40960
32768
24576
16384
8192
0
Time (s)
0
50
100
150
200
250
300



Sequence numbers in flight for 10.185.74.13 decimation
Sequence Number
Time (s)
65536
57344
49152
40960
32768
24576
16384
8192
0
500
1000
1500