An Efficient Loop-Detection Algorithm for SIP Proxies

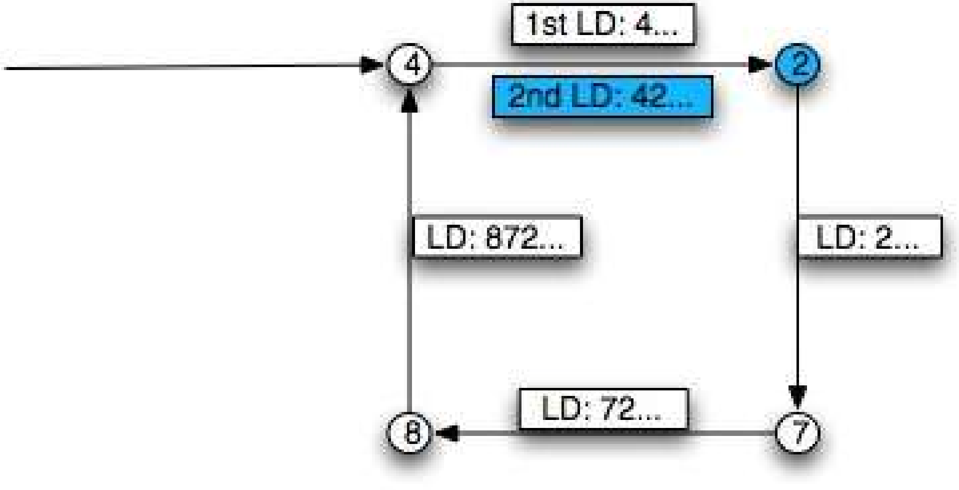
draft-campen-sipping-stack-loop-detect-00.txt

Byron Campen
Estacado Systems

The Algorithm in Brief

- All nodes have a unique number(node value)
- Requests will contain a stack of node values.
- When a request passes through a node (with value x), pop node values until a node value less than or equal to x is found. If we find a node value equal to x, we have found a loop.
- Push x onto the stack, and forward the request.

An Example



When the request traverses the minimal node, the node value that is pushed persists until the request comes back. The value is discovered at that time.

Computational and Space Complexities

- O(n) aggregate complexity, O(1) average for each proxy. Constant multiplier is slightly less than that of RFC 3261 loop detection.
- O(log n) average space requirement. Constant multiplier is btw 17-26 bytes.

Other Desirable Qualities

- Malicious UACs and proxies in the “tail” cannot cause the algorithm to fail in detecting a loop.
- Non-participating proxies will not cause the algorithm to fail, as long as there is at least one participating proxy in the loop.
- Much better than other algorithms at handling a long “tail” (something that could be easily introduced by someone with malicious intent)
- Handles short loops very efficiently.

Possible Shortcomings

- Requires a new header, and additional bits
- B2BUAs can corrupt the state needed for loop-detection (removing/reordering headers)
- Algorithm halts at a random point during the second loop.
- Vulnerable to broken or malicious proxies inside the loop.
- False positives are possible (but unlikely)