Dataplane probing

Petr Lapukhov,
Facebook
petr@fb.com

Problems we trying to solve

- Detect faults
- Isolate faults
- Do ^^^ quickly (< 1 second)

Traditional approach

- “White-box” debugging
- Device counters
  - Standard counters (SNMP ifMIB etc)
  - Non-standard (fabric drop counts)
- Slow to retrieve
- No full coverage

Blackbox debugging

- Inject active probes to detect loss
- Run traceroute to discover paths
- Analyze and project fault locations
- How we do it:
  - UDP from dedicate machines to all servers
  - Time-to-detect ~10-20 seconds
- Downsides
  - Still slow
  - Not very reliable
  - Hard to isolate

Inband-telemetry

- Advanced silicon functionality
- Embed device’s state in transit packets
- Full line rate maintained
- Could be done in SW switching

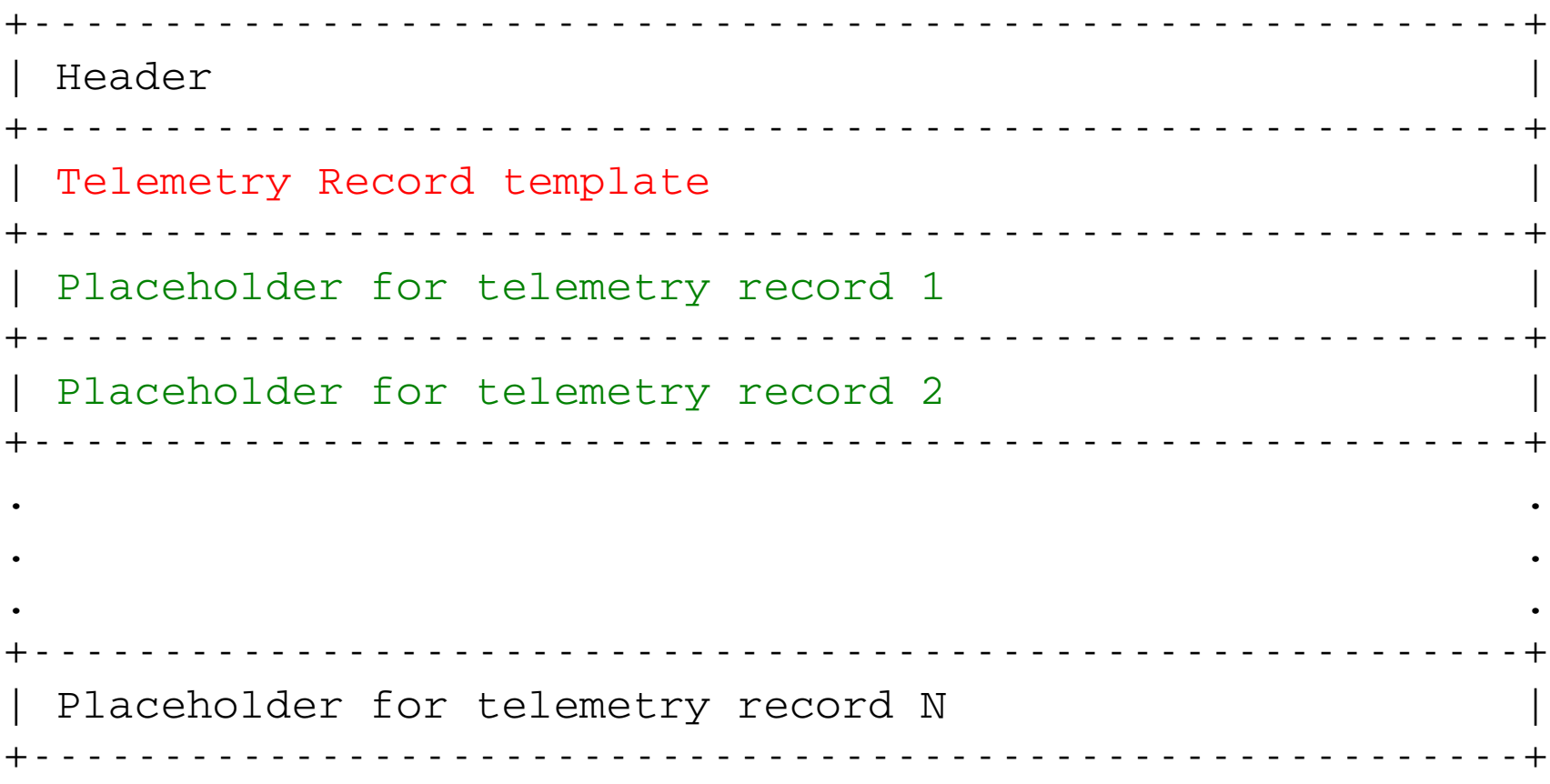
What we propose

- Build on active probing approach with UDP
- Define method to *request* telemetry data
- Define kinds of data to store
- …and format to store data
- Keep it simple and extensible
- Fallback to software

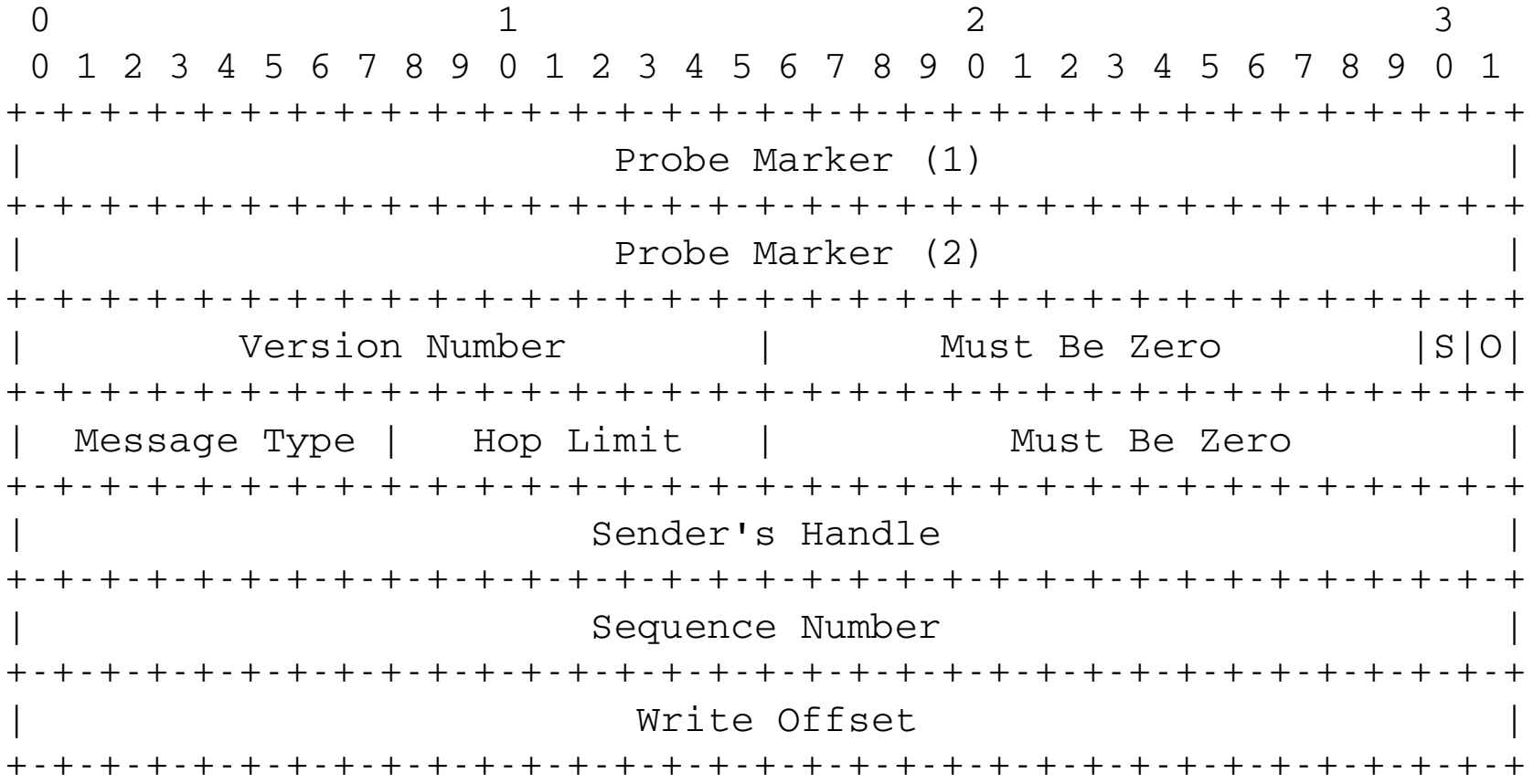
Why UDP?

- Has to be handled by application
- No kernel interference (if port is open)
- ECMP hashed (src port)
- Lots of room to embed data
- Limitations:
  - Firewalls, ACLs etc
  - QoS settings (use DSCP!)

Probe Layout (UDP body)



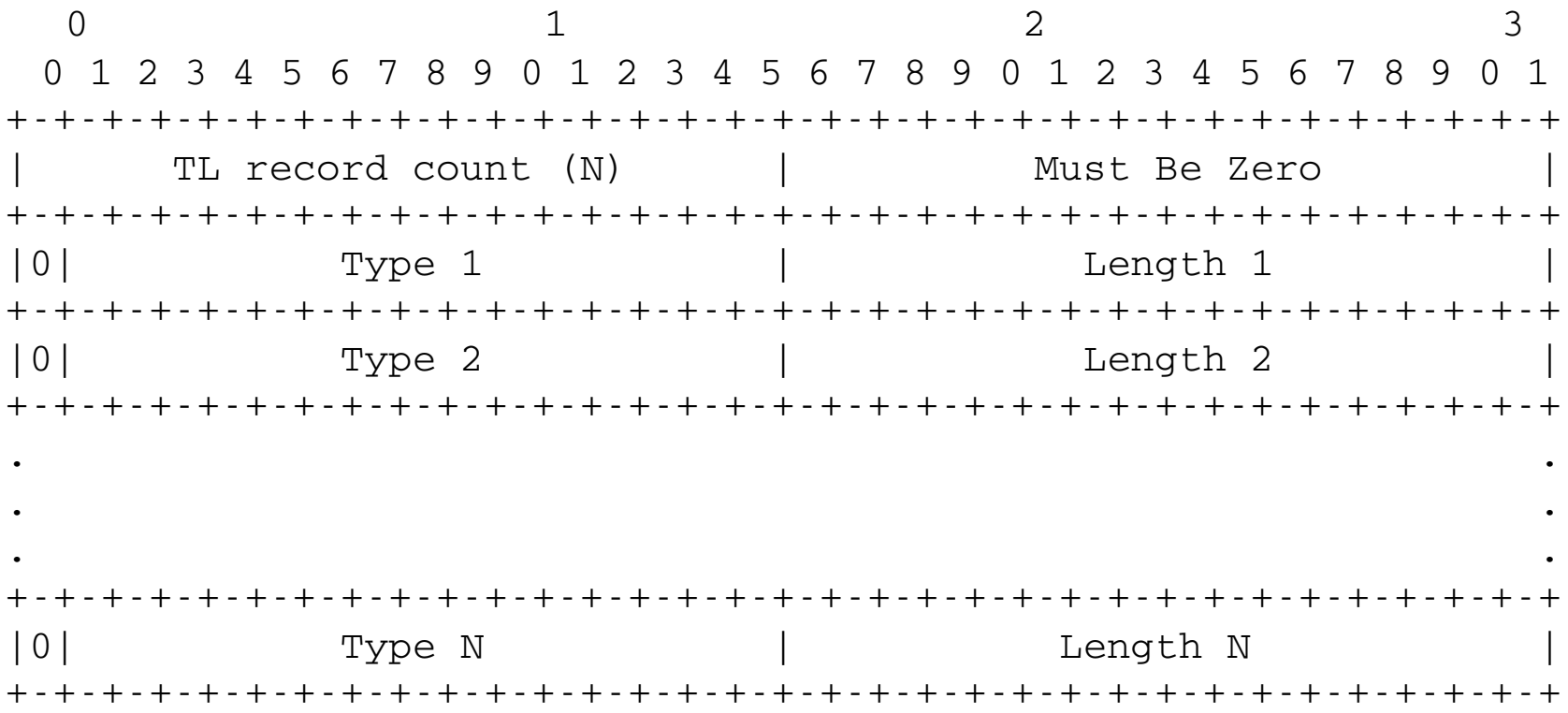
Probe Layout (UDP body)



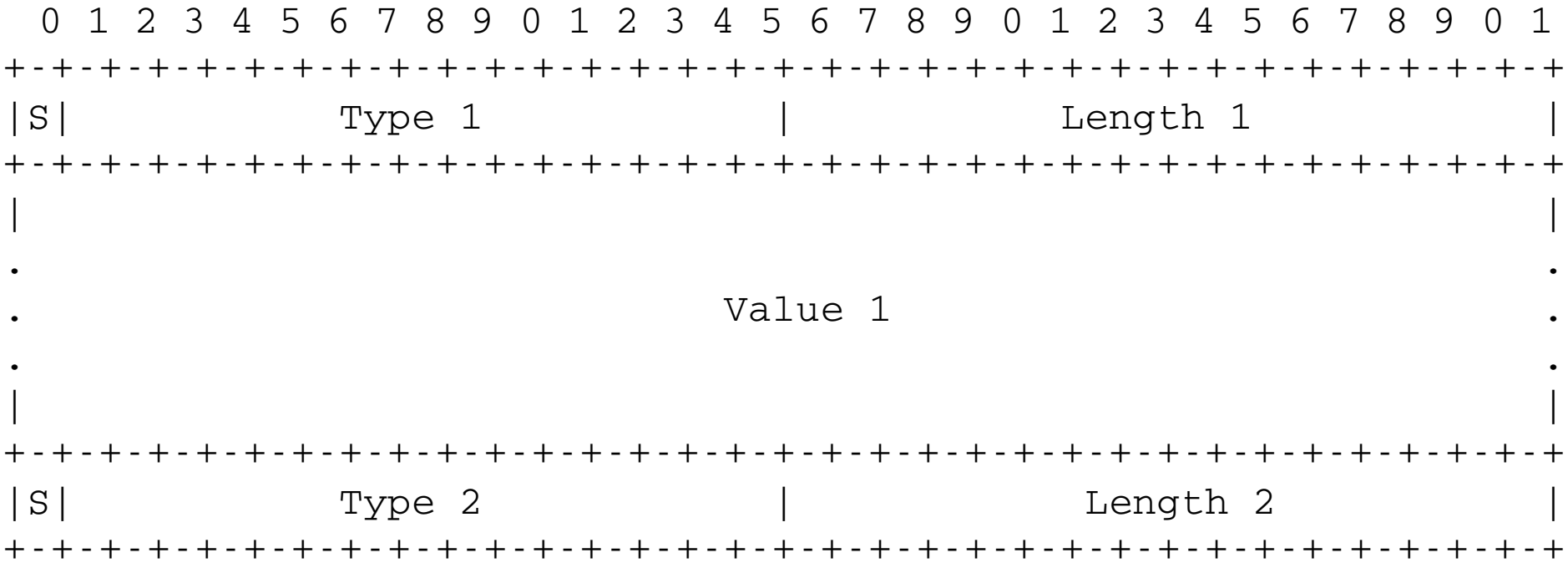
Two message types

- “Probe” vs “Probe Reply”
- Implement “loopback” tests
- Hop Limit is used to set *turning* point
- Packet forwarded by regular routing (IP TTL is normal, etc)
- Loopback test runs at line rate

Telemetry Record Template



Telemetry Record



Types of data to collet

- Device ID
- Input/Output Port*
- Forwarding state (might be tricky)
- Queue depth*
- Enter/departure times

Wrap-up

- Fast detection and isolation
- Embedding device state in active probes
- UDP is flexible and simple
- Path tracing and progressive loopback tests
- Minimal required metrics
  - Other could be added as needed

Other approaches

- Extension headers
- Tunnel header
- Source routing