Packet White Paper: 255
Status: Request for Comments
Date: October 2025
Author: Paula Dowie

Packet Network Monitoring Project

Abstract

This preliminary draft proposes a network-wide monitoring system for the Amateur Packet Radio network to help diagnose outages. The recommended architecture includes a central server/database, node reporter clients, a web UI, and a data "hose line" for applications.

The document suggests a collaborative project management approach and recommends a UDP transport protocol for reports formatted in JSON, specifying minimum required fields like node callsign, timestamp, and report type for unique identification. The document aims to initiate discussion and project development.

Status of This Memo

This document is a preliminary draft for discussion purposes only. It has no official standing and is subject to change without notice. Feedback and comments are welcome.

Table of Contents

1. Introduction

The Amateur Packet Radio network is experimental, with a significant probability of outages. Reasons for outages include, but are not limited to, power cuts, faulty software, hardware failures, human error, interference, anomalous propagation, and "finger trouble".

When things go wrong, it is often difficult to find out WHY and WHERE they went wrong, meaning that lessons are not learned, and the problem may keep recurring. Therefore the view has been expressed several times that there ought to be some sort of network-wide monitoring system.

The purpose of such a system would be to provide a real-time overview of the state of the network, and to record historical data such that the network state at any moment in time could be reconstructed.

This document aims to stimulate discussion and maybe "get the ball rolling" on such a project.

2. Project Management

This is not the sort of project that should be dictated by a web developer alone. It needs to be a multi-way collaboration between node authors and higher-level developers, not a top-down approach.

Without an intimate understanding of the inner workings of node software, a web developer is likely to specify a reporting protocol which is convenient for them, but impractical for node authors to implement. The node authors must agree on, and implement a reporting protocol that is also acceptable to the database operator. The role of any project manager, if one is required at all, should be to coordinate, not micromanage every detail.

3. System Architecture

One possible implementation of a packet network monitoring system would consist of an internet-based server/database, a simple reporting client in every node, a web UI, and a "hose line" of all status messages, allowing other developers to build custom applications around the data.

A "Radio-Only" implementation could use standard packet mail, or a variation thereof, to move aggregated status reports to a central server. It would be slower and less reliable than an internet based solution, but nevertheless might be an interesting toy.

4. Server / Database

The server needs to be independent of the node software authors and available to all types of node software. Is this the sort of thing the "OARC Compute" service could be used for?

In its simplest form the server could consist of an endpoint which receives reports from nodes, and queues them for the database to pick

up, plus a TCP server to redistribute reports to hoseline clients.

The database could be integral to the server, or a separate process that pulls data from the server's output queue.

To avoid loss of data, the server core needs to focus on keeping the input queue as clear as possible. Hoseline and database operations are lower priority.

5. Reporter Clients

The aim would be to include a reporter client in every type of node software, if possible.

The client must be easy for all node authors to implement, otherwise the project will never materialise. Node authors should not be expected to add large amounts of complex code, or to require external dependencies. Let us invent our own wheel, not slavishly use someone else's.

The clients would report node status and events to the server in a timely manner.

6. About Reports

6.1. Transport Protocol

UDP would be simplest to implement, and would place fewer demands on both client (the node software) and server. The client can simply "fire and forget" the reports. A few datagrams might be lost, but such losses are relatively rare, and in the vast majority of cases would be of little consequence.

TCP is theoretically more reliable, but more complicated. The server might have to deal with peaks of hundreds or even thousands of connections per second, each consuming resources during its life cycle. The node software can't just "fire and forget", it has to manage a thread for the duration of that connection.

The overhead and time delays involved in creating and tearing down one connection per report could be reduced by each node maintaining a persistent connection with the server, but that may not be very scalable.

Ideally the server should support both TCP and UDP options.

6.2. Report Format

Whichever transport is chosen, the reports should ideally be as concise as possible, because some node sites may have limited internet data allowances. However such sites are probably rare, so the project must not be unduly hobbled to accommodate them.

Each report should ideally be human-readable, and clearly delimited so that the server may simply pull them off a text queue.

Although JSON is relatively verbose, it is probably the best compromise between size, ease of implementation, flexibility and

future extensibility. It is also understood by web applications and developers. Therefore it is the recommended choice.

Each report would consist of an unnamed JSON object, containing a number of fields, some of which may themselves be objects or arrays of objects. The report would begin with an opening curly brace '{' and end with a closing curly brace '}'.

If UDP transport is used (the preferred option), each datagram may contain more than one report, but must not contain partial reports.

In the following sections, the JSON field names are shown in full for clarity. They *could* be shortened to single characters, but that goes against the spirit of JSON, i.e. its human readability. Making the field names more cryptic would make things harder for both developers and users. At the end of the day, it would be humans reading these reports in the course of fault tracing.

The suggested reports deliberately contain redundancy, firstly because reports may be lost in transport or processing, and secondly because each report should be as self-contained as possible.

7. Minimum Report Fields

Each report MUST contain at least the node callsign, a timestamp, and the report type. The report should preferably include a serial number, to help disambiguate multiple reports with identical timestamps.

The following JSON fields are suggested:

Name	Type	Description
"type"	String	Type of report, e.g "linkUp", "linkDown"
"node"	String	Reporter's Node callsign (including SSID)
"time"	Integer	Reporter's timestamp (Secs since 1/1/70)
"serial"	Integer	Report serial number (16-bit)

The combination of nodecall, timestamp and serial number forms a globally unique "Report ID". There is probably no need to use serial numbers greater than 16-bit, as It would be extremely unlikely for the same 16-bit serial number to occur twice within the same second!

Nodes should preferably try to preserve the last-used serial number across restarts, but this is not essential.

8. Basic Reports

The reports in this section are merely a suggestion. If implemented, they would allow an overview of the network and the state of its AX25 links at any point in time. That may be of limited use, but is a framework upon which to add more functionality as required.

8.1. NODE_UP Report

Sent when a node starts up, and at regular intervals thereafter as "proof of life". MUST include the boot time, allowing intervening crashes, reboots or lost "nodeUp" events to be detected.

Name	Type	Description
"type"	String	"nodeUp"
"node"	String	Node callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970
"serial"	Integer	Report serial number (16-bit)
"since"	Integer	Time at start (secs since 1/1/70)

Optional fields:

Most of the following data COULD be obtained from the existing map server, but it might be useful to include in this project.

Name	Type	Description
"alias"	String	Node alias, e.g. "KDRMIN"
"locator"	String	Maidenhead locator, e.g. IO83VJ
"software"	String	Node Software type, e.g. "BPQ", "JNOS"
"version"	String	Software version, e.g. "504j"

The "regular intervals" mentioned above may need to be determined by trial and error, but five minutes is suggested.

8.2. NODE_ERROR Report

Optionally sent upon system error, such as "out of memory" / "out of buffers" / "node table full", that might affect the node's ability to handle packets correctly.

Name	Type	Description
"type"	String	"nodeError"
"node"	String	Node callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970
"serial"	Integer	Report serial number (16-bit)
"error"	String	Description, e.g. "no handles"

Errors such as "no buffers" may occur many times per second, so nodes should take steps to limit the number of identical error reports to a "reasonable" rate, yet to be decided.

Nodes may optionally send this report with "error" of "none" when all error conditions are safely cleared.

8.3. NODE_DOWN Report

Sent when a node is in the process of going offline. Unlikely to be sent when a node crashes, unless the SIGSEGV handler is hooked. However an unplanned "down" event may be inferred if a node fails to report within a reasonable interval, or if a subsequent NODE_UP event is received with a more recent "since" field.

Name	Type	Description
"type"	String	"nodeDown"
"node"	String	Node callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970

"serial"	Integer	Report serial number (16-bit)
"reason"	String	Reason for the shutdown, e.g. "reboot"

8.4. LINK_UP Report

Sent when an AX25 L2 link enters the fully connected state, and at regular intervals thereafter, for the lifetime of the connection. This report MUST include the remote and local callsigns, the port ID (number or mnemonic), the direction (in/out), and the start time. It MAY include other data such as the ax25 version, maxframe, paclen etc.

Name	Type	Description
"type"	String	"linkUp"
"node"	String	Node callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970
"serial"	Integer	Report serial number (16-bit)
"since"	Integer	Time at link start (secs since 1/1/70)
"port"	String	Port ID, e.g. "3" or "4mlink"
"remote"	String	Other end's callsign
"local"	String	Our end's callsign
"direction"	String	Direction: "in" or "out"

Optional fields:

Name	Type	Description
"modulo"	Integer	8=normal ax25, 128=Extended ax25
"version"	String	AX25 version, e.g. "2.2"
"maxframe"	Integer	Local maxframe for this link
"paclen"	Integer	Local paclen for this link
"qcount"	Integer	Number of packets currently in queue
"qlen"	Integer	Total queue length in octets
"sent"	Integer	Total numbered INFO frames sent so far
"resent"	Integer	Total numbered INFO frames retransmitted

The "since" field helps to uniquely identify the link, follow its progress over time, and mitigate against lost reports. A link serial number could be used instead, or in addition. The advantage of using "since" is that it makes it easier to locate the point in time where the link began.

The "qcount" and / or "qlen" fields could be included in the routine "keepalive" reports if they are non-zero. An excessive queue length could also trigger a report.

The "sent" and "resent" fields would allow the retry rate of the link to be estimated. A high retry rate might explain a subsequent link failure.

8.5. LINK_ERROR Report

This report would be sent if an error condition, such as FRMR or "retry count exceeded", occurs on an AX25 L2 link.

Name	Type	Description

"type"	String	"linkError"
"node"	String	Node callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970
"serial"	Integer	Report serial number (16-bit)
"since"	Integer	Time of link start (secs since 1/1/70)
"port"	String	Port ID, e.g. "3" or "4mlink"
"remote"	String	Remote callsign
"local"	String	Local callsign
"error"	String	Type of error, e.g. "FRMR Sent"

8.6. LINK_DOWN Report

This report would be sent when an existing AX25 L2 link is torn down. The fields are as follows:

Name	Type	Description
"type"	String	"linkDown"
"node"	String	Node callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970
"serial"	Integer	Report serial number (16-bit)
"since"	Integer	Time of link start (secs since 1/1/70)
"port"	String	Port ID, e.g. "3" or "4mlink"
"remote"	String	Remote callsign
"local"	String	Local callsign
"direction"	String	Initiator : "in" or "out"

The "direction" field indicates which end of the link initiated the disconnection, "in" indicating a received DISC frame, and "out" indicating a sent DISC frame.

8.7. CIRCUIT_UP Report

This report would be sent when a NetRom L4 circuit enters the fully connected state, and at regular intervals thereafter, for the lifetime of the circuit.

The report SHOULD include all the fields detailed below. It MAY include optional fields.

Name	Type	Description
"type"	String	"circuitUp"
"node"	String	Node callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970
"serial"	Integer	Report serial number (16-bit)
"since"	Integer	Time of circuit start (secs since 1/1/70)
"remote"	String	Remote address: "usercall@nodecall"
"local"	String	Local address (nodecall or applcall)
"remCct"	Integer	Remote end's circuit number
"locCct"	Integer	Local end's circuit number
"direction"	String	Initiator: "in" or "out"

Optional Fields:

"svcNum"	Integer	L4X "service number", e.g. 80=http
"qCount"	Integer	Unacked packets in TX queue
"qLen"	Integer	TX queue length in octets

8.8. CIRCUIT_ERROR Report

Name	Type	Description
"node"	String	Node callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970
"serial"	Integer	Report serial number (16-bit)
"since"	Integer	Time of circuit start (secs since 1/1/70)
"remote"	String	Remote address: "usercall@nodecall"
"local"	String	Local address (nodecall or applcall)
"remCct"	Integer	Remote end's circuit number
"locCct"	Integer	Local end's circuit number
"error"	String	Description of error

8.9. CIRCUIT_DOWN Report

This report would be sent when an existing NetRom L4 link is torn down. The fields are as follows:

Name	Type	Description
"type"	String	"circuitDown"
"node"	String	Node callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970
"serial"	Integer	Report serial number (16-bit)
"since"	Integer	Time of circuit start (secs since 1/1/70)
"remote"	String	Remote address: "usercall@nodecall"
"local"	String	Local address (nodecall or applcall)
"remCct"	Integer	Remote end's circuit number
"locCct"	Integer	Local end's circuit number
"direction"	String	Disconnected by: "in" or "out"

9. Future Additions

9.1. Routing Tables

The system as described so far would give an overview of the network, or at least the participating stations, and some of the AX25 and NetRom interconnections.

What it lacks is knowledge of the routing tables that dictate which interlinks are made. Such information would be highly useful, because many of the problems observed in the network appear to be caused by faulty routing.

By the time a problem comes to light, the routing has changed, so it is very difficult to track down these problems in real time. The ability to wind back and examine "snapshots" of the routing tables at any past instant would be invaluable.

However, such functionality would come at a cost, especially for the server. Node tables may be large, and sending them in JSON format would expand the amount of data to be sent. To be useful, the tables would need to be uploaded at regular intervals, say every 30 minutes maximum.

If the costs are considered worthwhile, a suggested report format

would be as follows:

Name	Type	Description
"type"	String	"nodeItem"
"node"	String	Sender Node's callsign (including SSID)
"time"	Integer	Seconds since 1st Jan 1970
"serial"	Integer	Report serial number (16-bit)
"data"	Object	The data for one node entry as below

The "data" object would contain the following fields:

Name	Type	Description
"dest"	String	Destination node's callsign (incl SSID)
"alias"	String	Destination node's alias
"via"	Array	1 or more egress routes (neighbours)

Each element of the "via" array would be a JSON object with some or all of the following fields:

Name	Type	Description
"nhbr"	String	Neighbour's callsign
"port"	String	Port identifier, e.g. "1", or "70cm"
"qual"	Integer	Netrom quality via this neighbour
"nttl"	Integer	Netrom Time To Live (minutes)
"ttime"	Integer	Trip time (if known) in 1/100ths sec
"hops"	Integer	Hop count (if known)
"ittl"	Integer	INP3 Time To Live (minutes)

The nhbr" and "port" fields are mandatory. The "qual" and "nttl" fields may be omitted if there is no netrom domain route info. Likewise the "ttime", "hops" and "ittl" may be omitted if there is no INP3-derived data. The information for at least one or the two domains must be included.

The "nttl" field would convey how much longer this route will remain viable, before being obsoleted in the netrom domain. This is a better method than using Obscount, Obsmin etc, which vary from node to node. The "ittl" field does the same job for the INP3 derived data.

Example:

This example shows two routes from G8PZT to GB7BDH, both with viable netrom metrics, but the second one with "horizon" INP3 values...

```
{"type": "nodeItem", "node": "G8PZT", "time": 1759688220,
 "serial": 5525, "data": { "dest": "GB7BDH", "alias": "SWINDN",
 "via": [{"nhbr": "GB7BDH", "port": "8", "qual": 10, "nttl": 120,
 "ttime": 2, "hops": 1, "ittl": 120}, {"nhbr": "M1BFP-1", "port":
 "18", "qual": 56, "nttl": 60, "ttime": 60000, "hops": 30,
 "ittl": 0}]}
}
```

A full node entry with all the fields populated could use several hundred bytes, so it is probably best to send them singly to avoid

ICMP "requires fragmentation" errors.

10. Timing Errors

If a node is reporting, it must therefore have an Internet connection, and in most cases is likely to be synchronised to a time server. However the latter point may not always be true. For example, Puppy Linux only time-syncs at boot-up, or manually thereafter. And some sysops may be running their systems on BST, not GMT.

An hour's discrepancy between the report and server timestamps could easily be corrected by the server. But that would hide an error which might need to be brought to someone's attention. Therefore the server should not tamper with the report timestamps, nor should it use them to arrange reports in order.

Reports must be entered into the database in the order of reception, along with the server's timestamp.

11. Random Thoughts

The project must be extensible, otherwise it risks being bogged down by trying to predict every future requirement. It should be able to start simple, and grow organically.

The project is intended for the UK packet network only.

There will be some "holes" in the system, because not all nodes have an Internet connection, some node sysops may opt-out, and some node software may not be capable.

12. Security Considerations

Some sysops may consider this system to be too "Big Brother", but it is completely voluntary, and they are free to opt in or out.

The system absolutely MUST NOT permanently record or divulge the IP addresses of the reporters.

Tracking individual packets through the network is theoretically possible, and would be fairly benign. But reporting the packet contents would be a step too far.

As it stands there is nothing to stop malicious actors injecting fake reports into the system. However the same is true of the existing node map API's. What would a malefactor gain from injecting fake reports? Would there be sufficient reward to make it worth his while? Is it really worth adding layers of security overhead to guard against something of little consequence?

A simple mitigation might be to validity check the reports, and discard any that fail to conform exactly to the minimum required format, or that contain unacceptable content. Any further reports from the same sender would be discarded.

13. Author's Address

Paula Dowie
Email: g8pzt@blueyonder.co.uk
WhatsPac: G8PZT
Packet: G8PZT@GB7BBS.#24.GBR.EU