AT4−339 Data Transport Protocols for the PSS−SDP Interface

Benjamin Shaw and Benjamin Stappers

PSS

September 2020

1 Summary

The purpose of this document is to provide an overview of the data transport solutions which could be utilised to transfer pulsar and single-pulse candidate data from the pulsar and fast-transient searching (PSS) sub-element of the Central Science Processor (CSP) to the Science Data Processor (SDP). At the time of writing, no common solution has been agreed upon for instantiating and controlling data transport sessions to the SDP from both PSS and pulsar timing (PST) as each sub-element has specific requirements which are not fully compatible with the other.

Here, we provide a description of the available solutions and assess each in terms of its ability to satisfy the specific requirements of PSS data flow, in the context of reliability, efficiency and maintainability. We also consider the implications of any data storage requirements within PSS. This work is a contribution to the SKA Architecture Design Review 6 [1] and the SKA SAFe feature SP−940.

2 System context

2.1 Description of PSS and SDP operations

During an observation in pulsar and/or fast-transient searching mode[1], data arrives from the individual receptors to a Correlator/Beamformer (CBF). Beamformed data is produced by the CBF, following which it is sent to the PSS sub-element whose purpose is to search the beamformed data for compelling pulsar and/or fast-transient candidate signals. Candidate data produced by PSS is then exported to the SDP for further analysis. A schematic showing data flow in and out of the CSP is shown in Figure 1. Full details can be found in [2].

The PSS component of the CSP performs a number of task on the beamformed data. Each task is performed by a particular PSS module. Task include:

- The removal of radio frequency interference (RFI)
- Dedispersion

[1]Fast-transient search mode may run simultaneously with pulsar search mode but may also run commensally with e.g., pulsar timing mode.

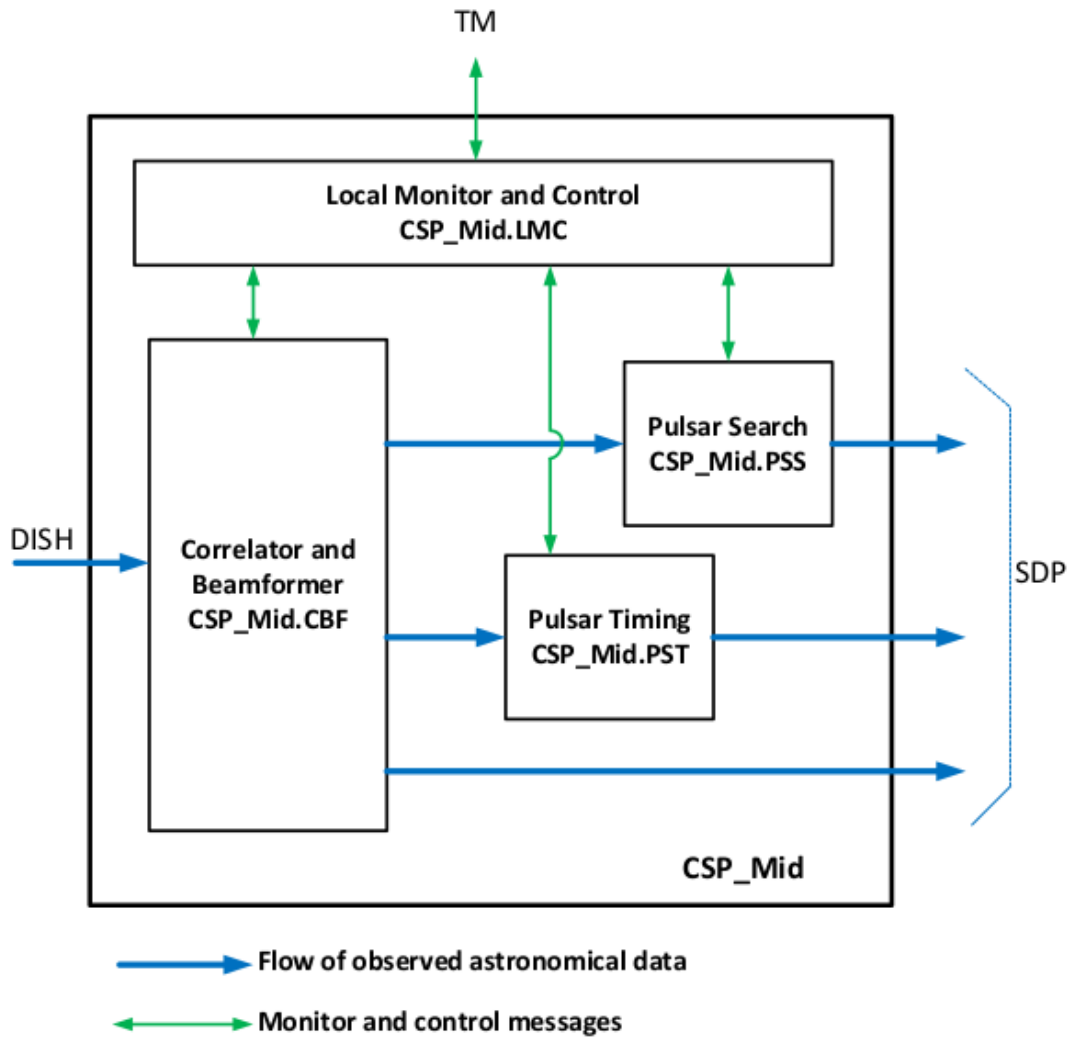

Figure 1: Context diagram showing the flow of data into, within, and out of the CSP for SKA1-MID.

- Single pulse searching
- Acceleration (pulsar) searching
- Sifting and candidate optimisation
- Candidate folding

Due to the high data volumes involved in pulsar and fast-transient searching (see [3]), and the necessity for real-time triggering, data must be processed and output by PSS in as close to real-time as possible [4]. The result of PSS processing is a large number of pulsar and/or fast-transient candidates. Candidate data comprises an ascii list of candidates and their associated metadata (position on sky, beam number, signal-to-noise ratio, period, dispersion measure, etc) as well as the relevant part of the time-frequency-flux data cube in which the candidate signal occurs. Full details of this process can be found in [5].

Candidate data is passed to the Candidate Data Output Streamer (CDOS) which is the final component of the PSS processing pipeline. CDOS's role is to prepare the candidate data for transmission to the SDP and to alert Local Monitoring and Control (LMC, see Figure 1) when the data are ready for transport. LMC, in turn, ensures that sufficient resources are allocated and ready within the SDP for the reception of candidate data. It was initially expected that data rates would be highly variable, with data being transferred to the SDP in 'bursts'. However, recent considerations, along with changes to how processing will be undertaken will result in a more stable data rate.

In the SDP, data is received by a software module[2] (pss-receive. See [6]) whose role is to process incoming data streams. Ultimately these data streams will be passed through a series of algorithms [7] which perform additional filtering on the candidates to remove duplicates, known sources and spurious signals (e.g., RFI). In the current implementation of pss-receive, candidate data is either saved to disk or discarded upon reception from PSS, according to a command-line flag but this will not be feasible for the data volumes in production.

2.2 The PSS−SDP interface protocol

The mechanism by which data is transferred from the PSS CDOS component to the SDP is referred to as the PSS−SDP interface. In the current ICD the PSS−SDP interface should be mediated by the file transfer protocol, *ftp*. At the time of writing however, candidate data are transferred to the SDP from PSS by means of a UDP stream as implemented by the SPEAD2 library [8]. The reason for this is two-fold. Firstly, the requirement for near real-time PSS processing, combined with the data volumes, dictates that data are continuously streamed directly to the SDP rather being stored firstly as files and then uploaded. This immediately rules out ftp as a practical solution. Secondly, UDP over SPEAD2 is the protocol used to stream visibilities data to the SDP from CBF and so this method was chosen when work began on the PSS−SDP interface due to the existing in-house expertise. However PSS has unique requirements in terms of speed and reliability and so it is crucial to assess a number of data transport solutions in order to ensure that requirements are met in the most efficient way possible. This is the purpose of AT4−339.

2.2.1 Factors to be considered

Speed This is a crucial factor in the design of PSS. Data transport must occur in real-time and with low latency (see [9]). PSS buffers must be cleared in time to allow any subsequent scans in the

[2]This module is deployed as a workflow, on demand, by Kubernetes.

scheduling block to take place on time.

Reliability Reliable data delivery is important for efficient searching and minimising time on sky and avoiding unnecessary repeat scans. This has implications for whether or not we use a connectionless protocol. Transfer resumability (e.g., after a long network link downtime) is a consideration here but this only applies the transport of files (see below).

Streaming vs. file transfer Data volumes are large and storage is limited. Scans are nominally 600 s in duration (though they may be as much as 1800 s) which yields ∼38 GB of time-frequency data in pulsar search mode, in addition to the associated candidate metadata. In transient search mode data volumes may be much larger. Due to the requirements trigger alerts in real-time, streaming data from PSS in real-time is necessary.

Data compression Data streams through the pipelines in real-time. As files are not being transferred, we do not need to consider file compression. Compression can be applied to the streaming of files, decreasing network bandwidth and disk usage (where required) but this would increase CPU loading and latency.

Authentication and encryption This is not currently a concern as data is expected to be transported between endpoints within private networks.

C++ bindings The chosen technique should be compatible with the existing PSS infrastructure.

Experience The chosen protocol and its implementation must be easily implementable, testable and maintainable in-house.

3 Data transfer methodologies

Below we discuss a number of data transfer techniques, their features and whether or not they are compatible with the requirements of PSS (denoted by a + or − respectively).

3.0.1 FTP

+ Widely used.
+ Transfer method specified in current ICD. However this was intended to be a placeholder whilst requirements were defined and other methods investigated.
+ Connection-oriented protocol (uses TCP). Packet delivery is guaranteed as long as network link exists.
− Is a file transfer method. Files would need to be written to disk and completed before transport can begin. This in turn would require some method for monitoring filesystem events which can watch for newly closed files and trigger transfer when required. iNotify is one such system and has implementations for c++.
− Requires authentication.

– Not resumable but has implementations that allow for this (e.g., LFTP).

– Slower than streaming.

– Old technology

3.0.2 RSYNC

+ Widely used.

+ Somewhat more sophisticated than FTP - can send only file deltas when required so is effectively resumable. Consequently more reliable than FTP.

+ Faster than FTP. Can compress and decompress at either end.

+ Connection-oriented protocol (uses TCP). Packet delivery is guaranteed as long as network link exists.

– Is a file transfer method.

3.0.3 SPEAD2

SPEAD2 is a implementation of SPEAD[3] (Streaming Protocol for Exchanging Astronomical Data). SPEAD2 was developed within the SKA.

+ Has python and C++ bindings - implementable in both PSS and in any python-based workflows in the SDP.

+ Highly customisable. Can tune buffer sizes.

+ Streaming support.

+ Supports transport over TCP and UDP .

  + Straightforward to switch between UDP and TCP as required.

  +,− UDP is fast and tolerant to intermittent packet loss.

  +,− TCP will retry lost packets but the connection will fail completely after a number of retries.

  – A custom protocol could implement retry on top of UDP.

+ In-house expertise.

+ Currently implemented solution (using UDP).

[3] see https://casper.ssl.berkeley.edu/wiki/SPEAD

3.0.4 TUS

TUS[4] is a protocol that is built on top of http with a specific focus on the ability to resume data transfers that were interrupted (e.g., due to connection loss). TUS is specifically designed for use in unreliable networks and/or system where large volumes of data are handled.

+ Supports (and is designed for) resumable transfers.
− Not widely used and no in-house expertise.
− No current C++ implementation. It would be problematic to integrate this into PSS. C++ support may become a feature in the future.
− No streaming support.

3.0.5 RDMA

RDMA (Remote Direct Memory Access) is relatively young protocol, distinct from TCP/UDP/SSH etc. This protocol allows for the transfer of data directly between the memory of one host and the memory of the other. The advantage of this is that it bypasses the operating system kernel, reducing processor load, in turn enabling rapid transfer of data with small overheads compared to other protocols.

+ Designed for high-throughput, low latency data transport.
+ Faster than TCP.
+ Supports streaming.
− Connectionless - no guarantee of packet delivery.
− Requires network support.
− Limited in-house expertise.

3.1 What do we intend to use and why?

The requirement to stream data in real-time directly to the SDP rather than upload files, substantially reduces the data transfer protocol options for the PSS−SDP interface. FTP and RSYNC can be immediately ruled out as possible solutions for this reason alone. TUS, though promising for its ability to resume transfers after periods of network downtime, also does not currently support streaming and has no C++ bindings, making it problematic to integrate into the PSS pipeline. TUS does however, have a large and increasing number of implementations for various languages, built by the TUS developers themselves by a wider community of open-source developers and so it could become a workable solution in the future. RDMA is fast and supports streaming, however a) it is connectionless and b) requires additional hardware and software support and as a relatively new protocol, currently has limited expertise within the SKA community which could add significant development time overheads.

The remaining option then, from those considered, is SPEAD2 which is the currently implemented solution for the PSS−SDP interface. Though SPEAD2 was originally designed for UDP, TCP is also supported and it is relatively straightforward to switch between the two where required. We currently

[4] see https://tus.io/

use a UDP stream to export data from PSS. However, given the lack of certainty of packet delivery over UDP and PSS' need for reliability, we must consider whether TCP is a more favourable option and identify any risks of using TCP that may be incompatible with PSS requirements. The pros and cons of TCP and UDP in the context of PSS are summarised below.

- UDP is insensitive to packet loss. If the server (SDP) does not receive packet i from the client (PSS), the server cannot communicate this to the client in order to request retransmission. The client will continue sending data regardless, from packet $i + 1$ onwards. In this model, packets are dropped from the CDOS buffer, at the point the client has attempted to send them. CDOS buffer space then becomes available for incoming data therefore dropped packets have no effect on the ability of PSS to continue scanning and processing further data. When the transfer is completed for a particular scan, there may be some missing packets.

- TCP requires an established connection between endpoints and is therefore sensitive to packet loss at the cost of transmission speed. If the server receives packet i from the client, the server sends an acknowledgement of receipt back to the client at which point the client can drop that packet from the CDOS buffer. If the server does not receive packet i from the client, the client will receive no acknowledgement of receipt and will wait a fixed period of time (re-transmission timeout, RTO) before resending. The RTO doubles with each failed transmission for a fixed number of attempts after which, if no acknowledgement is received from the server, TCP will terminate the connection and no further data will be sent. Without a contingency plan, PSS risks losing data if any network downtime $t_{\text{outage}} > \text{RTO}_{\text{max}}$ (see §4).

  In the event of a short-term network outage (i.e., $t_{\text{outage}} < \text{RTO}_{\text{max}}$) data transport can resume after failed attempts, but with temporarily decreased throughput. In the event of an unstable network connection, the throughput will vary, impacting on the total transmission time and leading to potential resource conflicts if the CDOS buffer isn't cleared in time for incoming data, particularly near scan boundaries.

A PSS−SDP interface utilising either TCP or UDP carries risks of data loss. A custom solution could be implemented which uses UDP whilst keeping packets in the client's CDOS buffer until the server provides a receipt. Such a solution would provide packet delivery reliability and suitable alerting/logging when packets are dropped whilst avoiding the risk of early connection termination that TCP brings. In the meantime, during development, PSS will continue using the current implementation of the PSS−SDP interface which uses a UDP SPEAD2 stream. As there is no data transport mechanism that can absolutely guarantee complete and timely data delivery, some tolerance for data loss should be agreed upon by PSS.

4 Future considerations

Though beyond the immediate scope of this document, the PSS team should consider what course(s) of actions should be taken in the event of connection loss between PSS and the SDP (either due to a SDP outage or a loss of network connection), in order to avoid data loss and wasted time on sky. There is scope for temporary data storage on the PSS which can be utilised should the SDP become uncontactable and operational support will be required to ensure that candidate data is re-directed to disk when needed.

A connection loss could occur mid-observation or prior to the commencement of a scheduled PSS observing block. If mid-observation downtime occurs, CDOS would receive pulsar and fast-transient

candidate data but would be unable to prepare it for transfer. Below we outline some key questions and concerns that should be considered in order to address this possibility.

- If using a connectionless protocol, the SDP will not be able to directly notify PSS that it is not receiving data. If an SDP outage occurs during an observation, how is this information communicated back to PSS (via TM?)?
- CSP.LMC will soon have the capability to instantiate and terminate PSS. Will CSP.LMC also be able to monitor the connection between PSS and SDP via TM?
- If so, can CSP.LMC instruct PSS to terminate the observation in the event of an outage? Under what circumstances can processing continue and utilise the local storage for candidate data?
- Similarly, if using a connection-oriented protocol, how should PSS handle connection loss

As there is some available disk space in PSS, the action taken by PSS will depend on the specific search mode it is operating in (transient search vs. pulsar (acceleration) search).

- In transient search mode (which may be commensal or simultaneous with pulsar search mode), single-pulse candidate data is streamed in real-time to the SDP. If SDP downtime occurs mid-observation, some candidate data will have already been streamed to the SDP. At the point of termination, no further single-pulse data will be ready for transmission. Data loss is minimal. If local disk space is available, the scan can continue and data re-rerouted to disk for later transmission to SDP.
- Pulsar search data is transferred only after a scan has ended as the full observation is required before the search for periodic sources can begin (see [4] for full details). This means that during a scan, pulsar searching is taking place on the data of any previous scans in the scheduling block. In the event of an network outage, the current scan will be terminated but the processing of the previous scan's data can continue. This could result in a conflict of resources, as described in [4] as the previous scan's processing must complete before processing of the current scan's data can begin.
  - If there was a previous scan in the scheduling block, can processing be allowed to complete and the candidate and time-frequency data be stored to disk for later transmission to the SDP. How do we check for available disk space? What funtionality currently exists that can route candidate data to disk instead of CDOS?
  - If SDP downtime occurs during the first scan of a scheduling block, can processing begin on the data as soon as the outage occurs and the (shorter than scheduled) scan data saved to disk, space permitting?
- If data was stored to disk during a SDP outage, how soon after a connection to SDP is re-established, does PSS transfer any stored candidate data?
- Is any functionality in place that could send candidate files to CDOS for transmission to SDP. Can this be orchestrated by CSP.LMC to provide SDP resources whilst PSS is not actively scanning?

5 Conclusions

- The current ICD states that FTP is chosen protocol for the transfer of data from PSS to the SDP. This was intended to be a placeholder when the ICD was written.
- PSS must be able to stream data directly to SDP, reliably and in near real-time without reliance on local file storage.
- FTP, RYSNC and TUS are file-oriented protocols and so are not suitable for use in the PSS−SDP interface.
- The current version of the PSS−SDP interface uses a UDP stream via SPEAD2 which was developed in-house, is software-compatible, supports streaming and is highly configurable.
- For reliability, TCP may be a more preferable solution.
- Neither TCP or UDP (via SPEAD2 or otherwise) are immune from loss of data.
- PSS will continue to use UDP streams for the PSS−SDP interface development but acknowledges that other solutions should be investigated in the longer term.
- PSS will consider the destination of pulsar and single-pulse candidate data in the event that the SDP becomes uncontactable.

References

[1] P Wortmaan. ADR-6 protocol for ingesting file-like data (PSS, PST). 2020. https://confluence.skatelescope.org/pages/viewpage.action?pageId=105415435.

[2] SKA1 CSP Local Monitor and Control Detailed Design Document. 2018. Document ID: SKA-TEL-CSP-0000102.

[3] E Keane. Transient Buffer Discussion Document. 2015. https://confluence.skatelescope.org/pages/viewpage.action?spaceKey=SEtitle=2020-04-21+architecture+synch+meeting.

[4] B Shaw. ADR-14 PSS Processing - Subarray Monitoring and Control. 2020. https://confluence.skatelescope.org/display/SWSI/ADR-14+PSS+Processing+-+Subarray+Monitoring+and+Control.

[5] SKA CSP Pulsar Search Sub-element Detailed Design Document (ED-4a). 2018. Document ID: SKA-TEL-CSP-0000082.

[6] B Shaw. SDP Prototype documentation: PSS Receive Workflow (readthedocs). 2020. https://developer.skatelescope.org/projects/sdp-prototype/en/latest/workflows/pss_receive.html.

[7] R. J. Lyon, B. W. Stappers, L. Levin, M. B. Mickaliger, and A. Scaife. A processing pipeline for high volume pulsar candidate data streams. *Astronomy and Computing*, 28:100291, July 2019.

[8] SPEAD2 documentation. 2020. https://spead2.readthedocs.io/en/latest/index.html.

[9] B. Stappers, R. Lyon, L. Levin, A. Karastergiou, and M. Pearson. PSS Discussion on Latency, Single Pulse Candidates and Triggering for ADR. 2020. https://confluence.skatelescope.org/pages/viewpage.action?spaceKey=SEtitle=2020-04-21+architecture+synch+meeting.