Software-defined networks in large-scale radio telescopes

Broekema, P.C.; Twelker, D.R.; Romão, D.C.; Grosso, P.; van Nieuwpoort, R.V.; Bal, H.E.

DOI
10.1145/3075564.3075594

Publication date
2017

Document Version
Final published version

Published in
ACM International Conference on Computing Frontiers

Software-defined networks in large-scale radio telescopes

P. Chris Broekema
ASTRON: the Netherlands Institute for Radio Astronomy
broekema@astron.nl

Damiaan R. Twelker
University of Amsterdam
datwelk@me.com

Daniel C. Romão
University of Amsterdam
d.f.romao@uva.nl

Paola Grosso
University of Amsterdam
p.grosso@uva.nl

Rob V. van Nieuwpoort
University of Amsterdam,
Netherlands eScience Center
r.vannieuwpoort@uva.nl

Henri E. Bal
Vrije Universiteit Amsterdam
h.e.bal@vu.nl

ABSTRACT

Traditional networks are relatively static and rely on a complex stack of interoperating protocols for proper operation. Modern large-scale science instruments, such as radio telescopes, consist of an interconnected collection of sensors generating large quantities of data, transported over high-bandwidth IP over Ethernet networks. The concept of a software-defined network (SDN) has recently gained popularity, moving control over the data flow to a programmable software component, the network controller. In this paper we explore the viability of such an SDN in sensor networks typical of future large-scale radio telescopes, such as the Square Kilometre Array (SKA). Based on experience with the LOw Frequency ARray (LOFAR), a recent radio telescope, we show that the addition of such software control adds to the reliability and flexibility of the instrument. We identify some essential technical SDN requirements for this application, and investigate the level of functional support on three current switches and a virtual software switch. A proof of concept application validates the viability of this concept. While we identify limitations in the SDN implementations and performance of two of our hardware switches, excellent performance is shown on a third.

ACM Reference format:
P. Chris Broekema, Damiaan R. Twelker, Daniel C. Romão, Paola Grosso, Rob V. van Nieuwpoort, and Henri E. Bal. 2017. Software-defined networks in large-scale radio telescopes. In *Proceedings of CF'17, Siena, Italy, May 15-17, 2017,* 4 pages.
DOI: http://dx.doi.org/10.1145/3075564.3075594

1 INTRODUCTION

To expose the increased functionality in packet switching ASICs at the heart of networking equipment, the concept of a software-defined network has appeared. Whereas in a traditional IP over Ethernet network control over the data flow is mostly implicit, SDNs move this to programmable network controllers. In this paper, we study the usefulness of SDNs for an important application: large-scale radio telescopes.

Modern large-scale science instruments, and in particular radio telescopes, are often distributed sensor networks. They consist of large numbers of sensors that sample the object of interest. Specialized custom hardware is used to convert the collected data into the digital domain and perform dedicated signal processing. A centralized general-purpose computing facility reduces data volumes, calibrates the instrument and allows the user to extract science. These components are interconnected with high-volume networks, often based on off-the-shelf IP over Ethernet equipment.

We argue that an SDN adds valuable additional functionality, while mitigating some of the challenges an IP over Ethernet network may cause. Using a proof of concept application, we explore the required SDN functionality for this application, and the current level of support for these SDN features in three available hardware switches and a software-based virtual switch. We also present limited performance measurements, although we note that the investigated switches are of modest scale and performance. While we find significant limitations in some of our investigated switches, we demonstrate the viability of using an SDN in large-scale radio telescopes.

2 IMPACT

In a previous memo we described some of the limitations of an IP over Ethernet network in the LOFAR radio telescope [2]. By replacing the traditional Ethernet-based network infrastructure with an SDN, many of these issues are alleviated. We foresee that by changing the way packets are processed, the robustness of the system is improved. More direct control over the packet forwarding is gained and multi-homed systems are better handled. Furthermore, the increased flexibility offered by the software controlled data plane allows for additional instrument functionality, such as the ability to dynamically change the data flow or to integrate the data flow into the processing model. We discuss some examples below.

Robustness – Packet forwarding. A network switch typically receives data and forwards it to its destination(s). A MAC address table, matching output ports with host MAC addresses, is maintained in the switch, populated by monitoring source MAC address and physical port of incoming packets. When the appropriate output port cannot be determined, data is forwarded on all output ports, with the exception of the ingress port, reducing the switch to a hub. This behavior guarantees that packets have the highest

CF'17, Siena, Italy

978-1-4503-4487-6/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3075564.3075594

possible chance of arriving at their destination, but in normal operation this should rarely be needed. Further traffic, such as unicast acknowledgments, usually allows successive packets to be properly directed.

In a specialized sensor network with custom hardware, parts of this standard network stack may be omitted. A continuous stream of strictly uni-directional data is generated by the sensors and emitted to the network. A misconfiguration or failure of a sensor may cause the network to be flooded by packets forwarded on all ports. In effect this failure mode may be considered a self-inflicted Distributed Denial of Service (DDoS) attack.

In an SDN, the way unrecognized packets are handled is configurable. Non-matching packets may be dropped or have their headers forwarded to the network controller for further processing. Neither of these actions will cause the network to be flooded, although the latter may cause significant load on the network controller.

Robustness – ARP flux. In high-bandwidth systems, it may be necessary to connect multiple Ethernet interfaces in a node to the same network. While there are ways to *bond* these devices into a single virtual Ethernet interface, no guarantees can be given on the effective use of available bandwidth. Therefore we prefer to explicitly address each of the interfaces. Linux nodes tend to answer Address Resolution Protocol (ARP) requests for addresses they host on any interface, which may cause switches to associate the wrong port with that MAC address. This gives rise to *ARP flux*, in which data addressed to a node does arrive, but on the wrong interface, causing the operating system to drop this data. The positive control that an SDN allows over the data flow mitigates this problem. Whereas a conventional network relies on MAC learning to determine the physical output port to forward packets to, in an SDN the output port is explicitly defined.

Flexibility – Publish-Subscribe model. Unicast traffic is very difficult to redirect or duplicate at the network level. Switches may be configured to output traffic to a designated port, but this is resource intensive. Data forwarded on such a monitoring port is not easily consumed, since it will be addressed to the original destination host. Generally, a host operating system will transparently drop data not specifically addressed to it, unless running in promiscuous mode.

An SDN gives us the flexibility to not only forward data to a specific physical output port, but also modify the packet headers to match the receiving host. In essence, this breaks the tight coupling between sender and receiver, since the sending peer no longer necessarily has knowledge of the receiving peer's address. By moving this control to the software layer of the receiving system, we can build what can conceptually be seen as a publish/subscribe system. This is similar to IP multicast in concept, but not in implementation. Where IP multicast relies on the network infrastructure to properly direct data, in an SDN this is done by the network controller. This not only improves flexibility, but also allows us to integrate this functionality more deeply into our software.

In a radio telescope, we may use this functionality to process data more than once, without the overhead of re-transmitting data. This is useful when running multiple processing pipelines, serving multiple observation modes, or for the commissioning of new processing hardware or algorithms. In the latter use case, results from newly installed hardware or software implementations are compared to those from established systems to verify functionality.

Flexibility – Data flow as part of the processing model. The addition of a software component into the data plane of a radio telescope allows us to integrate this data plane into our software systems. This more direct and positive control over the data flow allows us to manipulate the data flow without having to reconfigure the sending peer. Data movement may now be directly coupled to the processing schedule, creating, for instance, a system of data-driven round-robin processing, where data is directed to processing nodes as needed. Going even further we could envision implementing an in-network transpose, similar in concept to an `MPI_AllToAll()`. The directing of data would be based on a combination of Layer 2 and Layer 3 addresses that identify the content sufficiently.

3 SDN IMPLEMENTATION

To explore the viability of an SDN as part of a modern radio telescope we focus on the most prevalent and most widely implemented software-defined networking protocol currently available: OpenFlow [3]. We investigate which features provided by OpenFlow are required to implement a suitable SDN. The support for these features in four available SDN-capable switches, both hardware and software, is summarized in Section 4.

Our proof of concept is simple, but allows us to gauge the level of support in current platforms. Two cases are implemented, both manipulating a data stream emitted from a LOFAR signal generator. This emulates a LOFAR station-processing board and emits a continuous UDP/IP data stream to a single host identified by its IP and MAC addresses.

Two main scenarios are tested that are representative of how an SDN may be used in a radio telescope:

(1) redirect data destined for A to B
(2) duplicate data destined for A to also go to B

Since we are most interested in the feasibility of the concept rather than the performance of these platforms, we limit ourselves to a single data stream of around 700 Mbps, which is $\frac{1}{4}$th of the data rate of a single LOFAR station. We limited ourselves to a single stream of limited bandwidth, but the near embarrassingly parallel nature of the data flow makes this a representative test that allows for an exploration of the required functionality.

Redirecting data. We redirect the data from its original destination to a second host by modifying the Layer 2 destination MAC address and the Layer 3 destination IP address and forwarding the packet to the appropriate port. We use the Set Field functionality defined in the OpenFlow standard to manipulate destination MAC and IP address fields. Note that the implementation of none of the Set Field features is required in the OpenFlow specification.

Duplicating data. Next, we modify the data flow such that it is emitted not just to its original destination, but also to a second host. We forward packets unmodified to the physical port hosting the original destination, while a second copy of the data stream undergoes modification much like described above. Since this requires multiple independent actions on the same data, a different approach is required. OpenFlow Groups allow the creation of action buckets, which are processed and applied independently. We

configure a Group such that all buckets are applied to a packet, a required OpenFlow function. We note however that the applied actions are bound by the same limitations described above: the Set Field functionality is still optional.

4 FUNCTIONALITY INVESTIGATION

Many of the OpenFlow features we use are *optional*: implementation of these is not mandatory and often omitted. We also found that some features that are implemented lack hardware support, leading to significant bottlenecks. Implementation of the use cases described in the previous section relies on the following functional elements in the OpenFlow standard:

- Set Field action destination MAC address
- Set Field action destination IP address
- Group Type=ALL: apply action instructions
- Group Type=ALL: output to multiple ports

We note that to be truly useful, an SDN must support Layer 3 functionality. While we can make a workable proof of concept at Layer 2 by manipulating just the destination MAC address, this requires either that all receiving hosts share the same IP address, or that all host Ethernet interfaces are run in promiscuous mode, which is not feasible in practice. We investigate available functionality in a virtual switch, Open vSwitch, and three hardware Gigabit Ethernet switches: a Pica8 P3290, a Centec V350 and a Brocade ICX7250.

Open vSwitch. Open vSwitch is an open-source virtual switch that can run within a virtual machine, and as control stack on dedicated switching hardware. Since the same software is also used in two of our hardware platforms, it is interesting to compare the feature set supported in a pure software environment to the hardware implementations. We used the most recent stable version of Open vSwitch at the time (2.5.0), which fully implements OpenFlow up to and including version 1.3, including all optional features.

Pica8 P3290. The first physical switch we investigate is a Pica8 P3290, located in the OpenLab testbed of the University of Amsterdam. It is based on a Broadcom switch ASIC and a Freescale CPU and uses the same Open vSwitch software as described above. This switch supports all features required to redirect data based on Layer 3 packet header fields, but modifying the destination IP address causes major packet loss, suggesting this is handled in software. As a work-around, we assigned both receiving hosts the same IP address and redirected data based on just Layer 2 fields. This is not necessarily a viable solution in production, since we rely on hosts sharing IP addresses, which breaks normal IP networking.

To duplicate data we rely on OpenFlow Groups, but in the Pica8 switch these did not work as expected. Only a single, seemingly random, apply-action was executed before packets were forwarded to their respective ports. Thus, correctly addressed packets would only appear at one receiving host, while all others would receive packets addressed to the wrong destination MAC address. For this switch we abandoned the use of OpenFlow Groups, and instead installed a single flow that sets the destination MAC address to the broadcast address and forwards the packets to all relevant output ports. We note that this solution rules out the use of hybrid networking, since the use of the broadcast address in streaming data will cause non-SDN switches to forward this data on all ports, quickly overwhelming the network.

Centec V350. An SDN was included in the latest incarnation of the distributed ASCI supercomputer, DAS-5 [1]. The installation at the Vrije Universiteit Amsterdam has two Centec V350 switches based on custom silicon and the familiar Open vSwitch software environment. Of the investigated hardware platforms, this was the only one that performed flawlessly. All features we use to redirect and duplicate data using either Layer 2 or Layer 3 header information are supported without observable bottlenecks.

Brocade ICX7250. The Brocade ICX7250 is a mainstream top-of-rack switch based on a Broadcom ASIC. While it supports OpenFlow, we found that there are many caveats that limit the usefulness of this hardware for our application. Although the switch is able to match flows based on Layer 3 fields, it can only modify a very limited set of header fields. IP destination address is not one of these, but the destination MAC address can be modified. Directing or duplicating data can only be done based on Layer 2 header fields, not Layer 3 header fields. Furthermore, the switch is not able to output to multiple ports while modifying packet headers, effectively rendering duplication of data impossible.

+: support −: no support ○: support with caveats	Open vSwitch	Pica8	Centec	Brocade
Set Field action destination MAC address	+	+	+	+
Set Field action destination IP address	+	−	+	−
Group Type=ALL: apply action instructions	+	○	+	+
Group Type=ALL: output to multiple ports	+	○	+	○

Table 1: Support for OpenFlow features on our switches.

Summary. In Table 1 we summarize our experiences with the four switches. While we identified missing features, and were not able to duplicate data on the Brocade switch at all, we successfully implemented our proof of concept on all switches. Nevertheless, it is clear that support for the desired OpenFlow functionality varies widely. Furthermore we found an undocumented deficiency in the implementation of OpenFlow Groups in the Pica8 switch.

5 LATENCY AND LOSS

In Section 4 we concluded that not all platforms are capable of Layer 3 packet header manipulation at line rate. Here, we investigate the loss and latency of all four platforms while redirecting and duplicating data using Layer 2 and, when available, Layer 3 packet header information. While the virtual network environment is functionally very well developed, it is limited in performance. Where other platforms will be expected to handle the output of a single LOFAR hardware stream, at approximately 700 Mbps, we reduce this to about 45 Mbps for Open vSwitch to avoid performance related packet loss. The virtual machine environment, and the software-based Ethernet switch make the latency measurements less than reliable. For completeness, and as a comparison against the other hardware-based switches, including these is still valuable. In all cases ten measurements are done, and averages are shown in Tables 2 and 3. Data loss is measured over a period of three seconds, with 12000 packets generated per second.

5.1 Latency and Loss – Redirecting

In Table 2 we summarize the latency and loss of data when redirecting streams. As discussed before, Layer 3 functionality is not available on the Pica8 and Brocade switches. We note that redirecting data using an installed flow that modifies the packet header and forwards packets to the appropriate port will lead to limited loss of data on the Centec and Brocade switches and the software Open vSwitch environment. When the flow is installed, exactly 16 or 32 packets are lost that arrive neither at the original nor the new destination. This number is unrelated to the data rate: we ran a low bandwidth experiment and observed similar lost data. This is likely due to a packet buffer that is flushed when the flow is installed.

To mitigate this behavior, we could implement the same functionality using OpenFlow Groups, in much the same way as we did with the duplication of data. Once data is flowing to the new destination, the original action bucket may be removed. Unfortunately, the RESTful interface to our chosen network controller, ryu, lacks functionality to remove action buckets from a Group. Our analysis above indicates that neither the Pica8 nor the Brocade platform will support this more advanced redirection of data. The

	Open vSwitch	Pica8	Centec	Brocade
Layer 2 latency (ms)	202 (316)	74 (212)	5.9 (0.76)	265 (74)
Layer 2 data loss (packets)	16 (0)	457 (381)	29 (7)	32 (0)
Layer 3 latency (ms)	158 (313)	n/a	6.9 (1.2)	n/a
Layer 3 data loss (packets)	16 (0)	n/a	29 (7)	n/a

Table 2: Average latency and loss while redirecting data. Standard deviation is shown in parentheses.

Pica8 switch exhibits significant loss of data when the appropriate flow is installed. While this amounts to approximately 1.3% of the total data flow, it is not incidental, suggesting a bottleneck in the implementation. While not nearly as significant as observed when we modified Layer 3 headers on this switch, it is still worrisome. We note that both the Pica8 and the Brocade switch exhibit high latency and extreme variance, especially compared to the Centec switch. The stability of the Centec performance is impressive.

5.2 Latency and loss – Duplicating

Table 3 shows our results for the second case study. We install an OpenFlow Group that forwards packets unmodified to the intended destination. A second copy of all packets is modified such that the destination addresses (Layer 2 and Layer 3) match the second receiving host and are forwarded on the appropriate port. The Pica8 switch does not support modification of the destination IP address at this rate, therefore for this switch we only modify the destination MAC address. In our Open vSwitch software environment we measure both Layer 2 and Layer 3 duplication of data for reference. Considering the excellent performance of the Centec switch, and the possible disturbance to the cluster network due to the necessary modifications on the hosts needed to make this work, we only show Layer 3 performance for that platform. The Centec switch once again performs very well. We again note significant data loss in the Pica8 switch, while none of the other workable switches exhibit any observable loss.

	Open vSwitch	Pica8	Centec	Brocade
Layer 2 latency (ms)	29 (38)	34 (70)	-	n/a
Layer 2 data loss (packets)	0 (0)	314 (308)	-	n/a
Layer 3 latency (ms)	59 (55)	n/a	5.1 (0.43)	n/a
Layer 3 data loss (packets)	0 (0)	n/a	0 (0)	n/a

Table 3: Average latency and loss while duplicating data. Standard deviation is shown in parentheses.

5.3 Summary

Both the Pica8 and the Brocade switches exhibit relatively high latencies, especially compared to the Centec switch. The loss of data on the Pica8 switch is worrisome, since it is indicative of a bottleneck that will become more pronounced when we scale up this proof of concept. In all measurements, the Centec switch showed excellent performance. Latency is low and stable, and apart from the initial packets lost when a flow is installed, no further loss was observed.

6 CONCLUSIONS

In this paper we investigated the viability of an SDN in a modern radio telescope. Based on an investigation on the functionality available in four switches and an implementation of a simple proof of concept program, we conclude that the concept is viable and valuable. An SDN by its very definition will mitigate some of the robustness issues we discussed in Section 1. In addition, we have shown that the additional flexibility we envision is feasible on at least some of the investigated platforms.

We note that the supported functionality in the investigated platforms varies greatly. While some offer support for all required features, others do not implement features critical for our application, or implement some in software only. The OpenFlow standard is characterized by a large number of optional elements, that may or may not be implemented by the manufacturer, some of which we depend upon. Although all switches claim OpenFlow v1.3 support, support for optional functionality is often not well documented. It is noteworthy that the way Pica8 implements apply-action instructions in Groups violates the OpenFlow standard.

7 ACKNOWLEDGMENTS

We gratefully acknowledge the review and copy-editing by Ágnes Mika, as well as the comments by the four anonymous reviewers. We thank the SNE OpenLab at the University of Amsterdam and Kees Verstoep with the Vrije Universiteit Amsterdam for their support. This work is supported by the SKA-NN grant from the EFRO/Koers Noord programme from Samenwerkingsverband Noord-Nederland, and the ASTRON/IBM Dome project, funded by the province of Drenthe and the Dutch Ministry of EL&I.

REFERENCES

[1] Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. 2016. A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. *IEEE Computer* 49, 5 (May 2016), 54–63. DOI: http://dx.doi.org/10.1109/MC.2016.127

[2] P. Chris Broekema. 2015. *Improving sensor network robustness and flexibility using software-defined networks.* Technical Report SKA-TEL-SDP-0000021. ASTRON.

[3] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.