DBBC3 packets forwarding for eVLBI observations

Giuseppe Maccaferri - IRA-INAF(*Italy*)

EVN TOG meeting, *Torun - 13-14 December 2023*

Overview

In this document it will be explained how Medicina and Noto have configured their new DBB3 and FlexBuffer(FB) backend to perform local recording and even the data forwarding to remote location, like international correlators, through the use of routing and NAT Linux features.
The basic idea is to extend the DBBC2 well tested VLBI configuration and operations actually in use, to the most performing DBBC3 VLBI backend[1].
We will refers to the DDC_Ev126[2] firmware, the last available specifically developed for EVN observations in Digital Direct Conversion (DDC) mode.

DBBC3 and FlexBuffer configuration

The DDC_Ev126 gives first 8 bbc group from IFA, other 8 bbc (9 to 16) from IFB and following 8bbc groups for the others IF. To keep the schema most closed as possible to actual DBBC2, just to make easy the Field System(FS)[3] schedule adaption even to DBBC3, we connected IFA to rcp and IFB to lcp of the "in-line" receiver, so first 8 bbc (1-8) bring rcp and second group (9-16) the lcp. The others IF are supposed to be connected to to others receivers or pols in case of a multi band/beam receivers, like the CTR (Compact Triband Receiver) we soon will mount on Medicina and Noto radio telescope, or stay unused, in case of single band/beam receivers.

For each used IF (or core), only one eth of the four, eth0, is now used and connected to our new FlexBuffer's(FB) 10Gbps interface. In a next firmware release, might be possible that another eth can mirror the same eth0 traffic, so permitting local recording and correlator forwarding. But this option seems not very important, since so far all eVLBI observations never required the simultaneous local recording.

The core3h and FB Ethernet interfaces are configured in private LANs, all prefixed by 192.168. We assigned a different LAN to each eth, because this is a point to point connection schema with no network switch involved. We decided to compose the third IP digit with the core number plus the eth number, just to make easy its identification, while the fourth IP digit is usually '1' for the eth and '30' for the FB. This is the last number available in a /27 class C network and is assigned to FB that act as gateway for each LAN. Below an example table.

			eth				gw/net	Destination0		
Ric/pol	IF	core	0	1	2	3	30/27	who	dev	IP
KR (dbbc2 A)	A	1	192.168.10.1	192.168.11.1	192.168.12.1	192.168.13.1	192.168.10.30/27	FB	ens106f0	192.168.10.30
KL (dbbc2 C)	B	2	192.168.20.1	192.168.21.1	192.168.22.1	192.168.23.1	192.168.20.30/27	FB	ens106f1	192.168.20.30
QR (K1R)	C	3	192.168.30.1	192.168.31.1	192.168.32.1	192.168.33.1	192.168.30.30/27	FB	ens106f2	192.168.30.30
QL (K1L)	D	4	192.168.40.1	192.168.41.1	192.168.42.1	192.168.43.1	192.168.40.30/27	FB	ens106f3	192.168.40.30
WR	E	5	192.168.50.1	192.168.51.1	192.168.52.1	192.168.53.1	192.168.50.30/27	FB	ens106f4	192.168.50.30
WL	F	6	192.168.60.1	192.168.61.1	192.168.62.1	192.168.63.1	192.168.60.30/27	FB	ens106f5	192.168.60.30
	G	7	192.168.70.1	192.168.71.1	192.168.72.1	192.168.73.1	192.168.70.30/27	FB	ens106f6	192.168.70.30
	H	8	192.168.80.1	192.168.81.1	192.168.82.1	192.168.83.1	192.168.80.30/27	FB	ens106f7	192.168.80.30
net_port			46220	46221	46222	46223				

1 G.Tuccari & all - DBBC3 — the new wide-band backend for VLBI (https://pos.sissa.it/344/140/pdf)
2 Sven Dornbusch, MPIfR - Setting up the DBBC3 for DDC_E mode manual
3 Himwich, E., "Introduction to the Field System for Non-Users", IVS 2000 General Meeting Proceedings, N. R. Vandenberg and K.D. Baver, 86-90, 2001

DBBC3 server can host only one client at a time, like DBBC2, but we want control it both form FS and from correlator, so in case of an eVLBI, we adopt the **dbbc_proxy** script, already in use with DBBC2. It is not fully compliant with DBBC3 exchange protocol and probably require some future update, but the actual version is anyway usable for the few commands correlator needs to send to set up a different destination in the core eth and to start and stop the traffic.

The NAT (Network Address Translation) and IP forwarding

In the INTERNET world, each local network (LAN or WAN) are interconnected to each others by a junction ring called router, a device that deploy two or more NIC connected to different LANs and made possible via IP protocol, the transfer of data packets form an Ethernet limited domain to another(routing). In brief through a router/gateways it is possible for any sender reach whatever destination in INTERNET. But this only in case of a public(world recognized) IP address while the private ones are normally not routed, as per IANA definitions[4].

Between DBBC3 and FB we adopt the private LANs because: the majority part of the traffic don't need to go outside, the big number of Ethernet interfaces involved and for security cautions.
In case we need to forward one or more core's output to correlator, we will use a well known technical called IP FORWARDING, that transform a Linux server in a router/gateway machine. It permits to pass the packets from a LAN to another based on the destination address. When there is an external destination address that core eth don't knows because it don't belong to its LAN, the packets are sent to gateway, the FB, as configured in our case. Thanks to the IP_forwarding option, the packets can be now forwarded to the destination address through the public WAN FB interface, in agreement with the routing table. This function didn't require any software installation. It is an already installed kernel option in all Linux installation; it just be simply enabled with:
sysctl -w net.ipv4.ip_forward=1

But core's eth packets start from a private IP address, as said before. To make them able to run over the global networks, we need to replace their private IP with a public one using another well defined technical called Network Address Translation[5] (NAT) . It consist to replace (translate) the source private not routable IP address of each packet, with a public one, normally the one of the gateway/router, in our case the FB itself, thus permitting each packet to be forwarded among many networks until its final destination. The gateway(our FB) will keeps also a table for each translation done, so the returned packets are replaced in the destination address with the original private one, before they get placed in the proper LAN.
In Linux this function is managed by a normally available package called IPTABLES.
For example, with a simple statement we can instruct our FB to translate all packets coming from private LAN 192.168.10.0/27 and going outside with the FB public address 90.147.132.72:
iptables --table nat -s 192.168.10.0/27 -A POSTROUTING -o eno1 -j SNAT --to-source 90.147.132.72

EVLBI operations

The DBBC3 cores are normally configured for local recording, with the destination address of each eth involved pointing its owns FB interface, as shown by :
core3h=<c>,destination 0 (where **c** is the core number) or
core3h=<c>,sysstat.

4 https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml
5 https://www.cisco.com/c/en/us/products/routers/network-address-translation.html

In this case the traffic from the cores is intercepted by **jive5ab**[6] , normally listening on specific port of all FB interfaces, for local recording.

To do and eVLBI observation, DBBC3 and FB must be first properly configured by a FS schedule or manually. Then JIVE people can start to interact with the local system through **dbbc_proxy** script to set the remote destination and control the data flow. The script must be run on a local device with an outside reachable public address. In our case we are using the FB itself.

For example, to set a remote final destination like a correlator, it's enough to set the right IP address in the involved core eth, with a command like:
core3h=1,destination 0 192.42.120.xx:46220
And start and stop the data transmission., with similar command,

Since the new configured addresses are not known by the core's eth, all the packets will be sent , automatically and without any further setup, to the gateway address of each LAN. FB is now configured as router (ip forwarding) and knows, from packets destination addresses and routing table, where to route the traffic through its public connection interfaces and only after the translation of the source address with its one.
The local recording could be restored in similar way with the command:
core3h=<c>,destination 0 192.168.<ce>.30:<port>
were: c is the core number (1-8)
e is the ETH number (0-3)

or completely stop the traffic with:
core3h=<c>,destination 0 none

Differently from the past, here we can have one or more stream of data, typically one for each core involved, even if on the other side the arriving packets have all the same source IP. The thread ID or net port can in this case be useful to discriminate the packets origin.
It is important, anyway that the total amount of sent data must be lower that the max capacity of the FB link to INTERNET.
In our case, with a 10Gbps NIC connected to INTERNET through 10Gbps fiber, we have sent a max around 8Gbps of data (steps are power of two), tested with a single core sending 8Gbps, 2 core sending 4Gbps each, or four sending 2Gbps. In all test, the translation load for FB CPU was very little, only some decimal percent, at least on our server, a SuperMicro SSG-640P-E1CR36H with two XEON 4316 CPU.

6 M.Verkouter - https://github.com/jive-vlbi/jive5ab/blob/master/doc/jive5ab-documentation-1.11.pdf

IP_forwarding and NAT flexbuffer configuration

Here are the setup commands, normally collected in a script we called **doGateway.sh**, to to be launched as root on FB server, to enable IP_forwarding and NAT translation.
First three commands clear tables and chains previously defined. Follow the enable of ip_forward option and 4 NAT rules for the in use private LANs.
The script has to be edited to adjust LAN, public address and out interface parameters and if necessary reduced or extended the NAT rules to the real used core's :

```
sysctl -w net.ipv4.ip_forward=1
cat /proc/sys/net/ipv4/ip_forward
iptables -F
iptables -F -t nat
iptables -X
iptables -A FORWARD -j ACCEPT
iptables --table nat -s 192.168.10.0/27 -A POSTROUTING -o eno1 -j SNAT --to-source 90.147.132.72
iptables --table nat -s 192.168.20.0/27 -A POSTROUTING -o eno1 -j SNAT --to-source 90.147.132.72
iptables --table nat -s 192.168.30.0/27 -A POSTROUTING -o eno1 -j SNAT --to-source 90.147.132.72
iptables --table nat -s 192.168.40.0/27 -A POSTROUTING -o eno1 -j SNAT --to-source 90.147.132.72
```

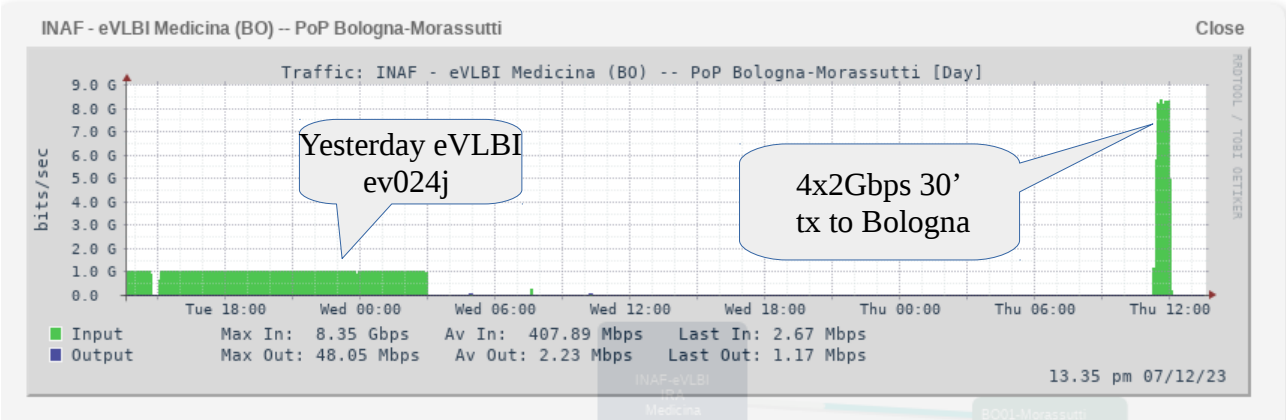

Illustration 1: A 30' transmission to remote of 4 core's @ 4x2 Gbps data