Harro Verkouter, September 23 2009

Performing 1024Mbps e-VLBI - packet and/or channeldropping

The MkIV formatter running at its highest rate produces 1024Mbps of data. This is very close to, actually a bit more, than the linerate of a 1Gbps (1000Mbps) ethernet link, with which most EVN stations are connected to JIVE. Using the simplest high-level standardized network protocol, UDP (User Datagram Protocol), there remain, after subtracting protocol overhead, roughly 980Mbps of usable datarate for payload.

The formatter works in power-of-two octaves, meaning that the next lowest datarate is 512Mbps, leaving almost 50% of the linkcapacity unutilized as well as reducing the sensitivity of the observation by a factor of roughly 1.4 (squareroot of 2 to be precise).

In order to maximize both the observation's sensitivity and usage-percentage of the link, the sending datarate must be brought below, but as close to, 980Mbps as possible.

Several solutions have been implemented and tested over the last year, using the jive5a code.

Packet-dropping

First, a simple data-dropping scheme was implemented and tested. By deliberately not sending all packets, the sending datarate can be lowered to something close to the network limit.

This idea was induced by the correlator's designed ability to cope with invalid or missing data due to its origin in the tape-based VLBI era where tapes were errorprone. Data marked as invalid will not be correlated and as such will not affect the quality of the correlation.

In fact, this ability was already exploited in e-VLBI since e-VLBI makes use of the UDP protocol. Packets sent via this protocol may or may not reach their destination at all. Also there is no guarantee the packets are delivered to the receiving application in the same order as they were sent.

In order to overcome those non-guarantees the e-VLBI sending software inserts a strictly monotonic incrementing sequence number in each packet it sends. In the receiving Mark5 software an algorithm is implemented which inserts fake data (and marks it as invalid) for packets not received or reorders the data in memory in case of out-of-order reception. Inserting fake-but-invalid data is a must since the correlator is operating synchronously and must be fed by a constant-bit-rate datastream.

The packet-dropping scheme works by tricking the receiver into believing data was lost on the network by letting the sender increment the sequence number but not actually sending the packet. The receiver cannot tell this situation apart from a packet sent but not actually delivered and will happily insert invalid data.

Packet dropping - version 1

A command to set the packet drop rate (pdr) was added to the jive5a software. When set to a non-zero value, the sending Mark5 would drop 1 in every pdr'th packet. When tested this indeed brought down the sending datarate below the networks' maximum. pdr values of ~20 - 25 were found to be adequate, effectively dropping 4-5% of data.

Multiple issues were found with this - apparently oversimplified - approach.

1) Unconditionally dropping one in every pdr'th packet lost too many headers, apparently. The MkIV datastream contains headers (amounting to approx. 1% of the total datarate) with vital information for the correlator, most notably, a syncword and a high-resolution timestamp. The correlator needs the syncword to detect where the sampled data begins and the corresponding high-resolution timestamp tells the system which time the samples represent. It turned out that if too many headers were not detected by the correlator (three in a row), it would lose synchronization without being able to recover, effectively losing all the data from that station from that point onwards.

2) The packets being dropped at individual stations are independent. Effectively what this means is that if two stations drop ~4-5% of their data and those streams are correlated, on average, roughly 8-10% of data on that cross-correlation is lost since in those 8-10% of the timeslots one of the signals will contain invalid data. Having a total loss of output data of ~10% is unaffordable.

Packet dropping - version 2

Based on the findings in version 1 it was realized that issue #1 could readily be solved by

making the sender a bit more carefull not sending packets. An algorithm was inserted that anaylyzes the packets immediately before deciding to send them or not. If the algorithm suspected header information in a packet it would not drop it even though the 1 in pdr'th value would indicate just that. Rather, the sender will, in that case, drop the first non-header-containing packet after that.

When tested this proved to solve issue #1 succesfully: the correlator never lost synchronization using this form of packet dropping.

Due to the nature of being completely independent stations issue #2 can not be solved in the packet dropping scheme. It would require all participating stations to drop the exact same packet, based on time, which would require synchronizing the station clocks to a level of much better than 70 microseconds (the timeslice of one packet of data). This is unrealizable in practice.

Channel dropping

Typically, an astronomer would sooner delete a whole channel (one polarization of one frequencyband) altogether, rather than having all subbands with gaps in the time-axis. The latter is what happens when packets are dropped: all subbands are affected at that point in time. One option to synchronize the dataloss across all participating stations would be to drop the same channel at each station.

Due to how the VLBI datastream is formatted (suited for tape - a longitudinal recording format) and how CPUs are sequentially oriented, this poses some practical hoops to be jumped through.

The channels are recorded in parallel bitstreams, one bitstream per tapetrack. At 1024Mbps there are 64 bitstreams in the datapackets. Each bit in a 64-bit word represents one bit of one bitstream. The next 64-bit word contains the next bit of each bitstream. At 1024Mbps one channel is recorded in four separate bitstreams. In order to drop one channel the four bits composing this channel must be deleted out of each 64-bit word. This will reduce the datastream by a factor of 4/64th, or 6.25%, enough to drop below the 980Mbps limit.

Unfortunately CPUs do not allow easy cut-and-paste bits from one word into another. An algorithm was developed which compresses the datastream as follows. The unwanted four bits in a 64-bit word are overwritten with four wanted bits out of the next word. This sequence repeats for the next word (filling up unwanted bits with wanted bits from a different word). This compressed data is sent across the network. Upon receipt of a compressed packet the receiver decompresses the packet, in order to make the bits appear at their originally recorded positions.

One bottleneck that might be a real limiting factor is CPU-speed. Compressing and decompressing as described, is a very CPU-intensive task, primarly because a CPU is not optimized to work at the bitlevel. It was questionable wether the Mark5's CPU's could keep up with realtime (de)compression at 1024Mbps.

Test-implementations of the algorithm were made and it was shown that the modern, dual CPU equipped Mark5s should be able, if barely, to keep up with (de)compression at 1024Mbps.

This algorithm was implemented in the jive5a software and tests at 1024Mbps were succesfully concluded, showing the feasibility of this approach.

Conclusion

Observing at 1024Mbps over a 1Gbps link can be made feasible by bringing down the datarate sent across the network. Doing this by means of channel dropping, subject to CPU power present, is highly favourable over packet-dropping for reasons explained earlier.

In case the CPU speed of a Mark5 at a station is not adequate for allowing realtime compression it is always possible to fall back to packet dropping. As long as the number of stations doing packet dropping is (very) limited its impact on the final dataproduct is acceptable: all data received above 512Mbps is a win, compared to running the whole observation at 512Mbps.