

Console Data at 15 Hz

Real-time network communications
Mon, Oct 12, 1998

Introduction

The Acnet data acquisition protocol, known as RETDAT, supports data requests that specify data return rates in units of 15 Hz cycles, the natural cycle rate of the Fermilab Linac and Booster accelerators, which operate at 15 Hz, synchronized to the 60 Hz power line frequency. The Linac beam pulse width is only about 50 μ s, so that the duty cycle of the beam is less than 0.1%. Sample-and-hold circuits are used to capture readings of pulsed signals at the same moment within the beam pulse throughout the Linac. A requirement for successful Linac tuning is that any selected Linac device data be made available to an operator at 15 Hz. On each 15 Hz cycle, Linac beam may be present, or it may not, depending upon the clock event system and the health of the preaccelerator and Linac hardware.

Fast Time Plot protocol

One common tuning display is called the Fast Time Plot, which uses the Acnet protocol FTPMAN, designed to carry time-stamped device data values from possibly multiple front end sources. The FTPMAN client software combines the separate streams of data according to the time stamps and displays the data graphically in a time-correlated fashion. One commonly tunes the Linac by displaying up to 4 beam-related signals plotted on the y-axis against a single controllable parameter, such as a transport line magnet current, plotted on the x-axis. In principle, as long as all the reply records reach the console Vax, one can expect all the data to be plotted.

In practice, however, it takes a carefully coordinated effort to achieve this. All Linac data source nodes operate in synchronism, each beginning its 15 Hz execution by the same triggering interrupt signal. Each node performs its cyclic activities at the same time. These begin with the `update` task, which updates the local data pool with fresh readings from the hardware, executes all enabled "local applications," and prepares any replies that are due (on that cycle) to be delivered to requester nodes that have previously initiated data requests that demand periodic replies.

In fact, the synchronism situation is slightly more complex, in that Vax consoles access Linac data indirectly through a data server node, in order to reduce the number of network messages that must be accepted by the console computer. Using a server node places an extra condition on 15 Hz reply data delivery. The replies from the front ends must be received by the server node before it can forward a suitable composite reply to the console. The Linac server nodes operate on a deadline basis. All replies must be received by a fixed deadline time within the 15 Hz cycle, because at that time, the composite replies are delivered to requesting nodes. This deadline time has traditionally been 40 ms past the start of the cycle. Because of heavily loaded conditions that have recently been observed in Linac nodes, this deadline has been extended to 56 ms, allowing more time for the individual Linac nodes to deliver their replies to the server nodes. (Actually these nodes obtain their I/O indirectly, via a Smart Rack Monitor, or SRM, via arcnet network hardware. Only after receiving the raw data from the SRMs can the data pool be updated, local applications run, and replies to active requests queued to the network.)

Acnet data request protocol

Aside from the Fast Time Plot method of data delivery to consoles for plotting, there is the more usual method of data delivery via the RETDAT protocol, in which replies received from either of the two Linac server nodes are placed into a data pool in the console computer. Console applications periodically poll the data pool. If an application needs to collect 15 Hz data, it must schedule itself to run at a minimum rate of 15 Hz, in order to be sure that it has a chance to sample all 15 Hz data that is found in the data pool. But the situation is a bit worse than that, because the Vax operates *asynchronously* with the accelerator 15 Hz operation. This means that an application must check more often than every 66 ms, because the actual reply message arrival times can exhibit some jitter at 15 Hz. One cycle's reply might arrive and be processed by the Data Pool Manager either a few ms late or a few ms early.

For Vax applications, the VMS scheduler service is invoked to request 66.6 ms periodic activation of the application. But because the scheduler only operates with 10 ms resolution, this means that the application will see ms delays between successive executions of 60, 70, 70, etc, in order to effect an average of 66.6 ms. Even if the front end and network hierarchy can deliver replies separated by a constant 66.6 ms, with no synchronization, it is possible for two successive replies to occur between two successive application executions. Allowing for a bit of jitter in the periodic reply arrival times, the chance for this happening now and again is even greater. As a way around this for an individual application, one could have it schedule itself for 60 ms periodic scheduling, or even less, and ignore the data if it is accompanied by "stale" status. This should give the application a good chance of sampling all 15 Hz data replies via the data pool.

One popular display application used for Linac tuning shows readings of about 50 beam toroid and beam loss monitor signals, all updated graphically at 15 Hz. This display has been observed to exhibit missing beam pulses. The current cycle's data points shown are displayed in yellow, with all previously plotted points displayed in blue. Thus, one can view what values occurred, with highlighting of those occurring at the moment. As a result of analyzing some recent timing measurements made by Jim Smedinghoff, changes were made in this application to improve its performance. Whereas the previous version of the program saved the data values from two successive cycles in order to deliver them to the display hardware together every other cycle, the new version plots the data every 15 Hz cycle. The technique of plotting two sets of data at once—presumably to reduce plotting overhead—was responsible for the display to seemingly miss plotting every other beam cycles, at times when beam cycles occurred an odd number of 15 Hz cycles apart, as occurs for a beam pulse rate of 3 Hz, for example. (The plotting is actually done on an x-terminal, so plotting commands are delivered via the network.) When the call is made to poll the data pool for the latest readings, a check is made that none was updated—via a high-priority reply message—while the entire set of beam data values were being sampled. If an update occurred, then the call is repeated to sample the data pool. This has the effect of insuring that all data collected to be plotted at one time is time-correlated, as all of these beam-related signals are passed through a single Linac data server node.

Commentary

What are we asking for here, when we ask for 15 Hz data to be displayed on a console? Not only must all the data replies be received and processed, but it is also important to deliver the graphics update in a timely manner. Ideally, the graphical display should visually appear to be updated at 15 Hz. It would not be adequate to have two or more 15 Hz cycles worth of data to be accumulated and then blasted onto the graphic screen in a flash. In this modern era of video displayed through a web browser, it would seem not too much to request that 15 Hz beam data be displayed on a controls console of the world's highest energy particle accelerator.

It is not too much to expect to have 15 Hz data displayed in a timely manner, but what method should be designed to make it happen? Perhaps the data pool paradigm in a Vax console that runs applications asynchronously is not adequate to the task. If not, then what is?

Page applications in Linac nodes

A suite of page-style applications is available for use in Linac front end nodes. Such applications operate at 15 Hz, in synchronism with the rest of that front end's operation. Because of this synchronism, the data pool paradigm works. An application knows that it can always sample any data from the data pool at 15 Hz without missing any cycle's data. Without such synchronism, the application would not be any better off than a Vax console. Synchronous operation is the key that makes 15 Hz data acquisition work successfully via a data pool.

How does EPICS do it?

In the EPICS control system, "monitors" are used to deliver time-stamped data values to console nodes when the values change. When the front end performs record processing to access the I/O, a check is made to see whether the engineering units value falls outside a database-specified deadband. If it does, then the value is queued for delivery via the TCP-based Channel Access communications protocol to all client nodes interested in that data. A console user can request that a callback function be invoked when a new data value arrives.

Since the "monitor" results are queued, all interesting data value changes are passed up to the user's application via the callback function, and the user should not miss any data. (There is no data pool mechanism unless the user chooses to create one for his own program. Some user applications may be designed to work exactly this way, with a display portion of the application operating at some user-friendly rate, sampling from such an application data pool.) It will still be necessary, however, to analyze the data points in terms of the associated time-stamps to come up with time-correlated data values for plotting.

SUN Data Request Support

Classic protocol support was implemented for the Sun workstations that were used with the Linac systems, completely independent from Acnet. The Sun always uses a server node to satisfy a request. This means that all the reply data for a given data request arrives in the same datagram. (This is the same situation as exists for the Linac beam signals example above.) The application waits for the data reply, processes the

data, then returns to wait again. But the replies are queued, so that in case the application is slow, it will have a chance to "catch up" when it is free. There is no data pool logic used. Queuing allows the application to see all of the 15 Hz data without worrying about reply jitter.

Future Acnet design

The Controls Department software staff is currently making plans for the future of Acnet. Will the collection and display of 15 Hz data be a requirement that will be supported? If it is, how could it work?

According to my present understanding, which admittedly may be incomplete, the architecture of the new control system is based upon a new layer of server computers called Data Acquisition Engines, or DAEs. Each DAE connects to 5–6 front ends. The console computers will run Java code that communicates with the DAEs that also execute Java code. This system will run more slowly than the current system, both because there is an extra layer and also because the interpreted Java code runs more slowly than compiled C code, say. It is difficult to see *a priori* how this architecture will help deliver reliable 15 Hz data to a host application.

The DAEs will be designed to consolidate all device requests (for those front ends they serve) on behalf of all host clients. To do this, they will support a data pool, probably analogous to the current Vax console DPM data pool. If a front end is designed to act as a server, à la Linac data server, however, the DAE data pool will be updated for all clients at once following reception of a reply message from the front end. When will the replies be delivered to a host client so that the data delivered is all measured on the same 15 Hz cycle? Since the DAEs are really PCs running NT, they are not real-time in design. But perhaps they can schedule 15 Hz execution with higher resolution than VMS. If the delivery is asynchronous with the replies from the front ends, some of the data may be missed, just as it can now be missed by Vax console applications that sample the data pool asynchronously.

The new system designers dismiss concerns for 15 Hz data by saying that the front ends will time stamp all data values they deliver in replies to the DAE. (Or perhaps the DAE will time-stamp the data?) When a console application collects the data values, it will then know when they were actually measured by the front end. But none of this assures that the data collected at one time by a host level application will be time stamped on the same 15 Hz cycle. Because of this, any host application that needs to work with time-correlated 15 Hz data will have a problem of sorting through the various data sets it receives on subsequent cycles hoping to find time stamps that match well enough to be presented together as correlated.

Even with time stamps provided with each data value, the new system will not be able to reliably deliver all the 15 Hz data that the front ends deliver to the servers, unless some means of queuing front end replies, rather than pooling them, is provided.

Summary

In summary, it seems that data consolidation across front ends implies the use of a data pool in the DAEs. But if the DAEs are not synchronous with the 15 Hz accelerator

cycle, then even if 15 Hz replies are reliably delivered to the DAES, it is difficult to imagine how 15 Hz data can reliably be delivered to a console client. With both the DAES and the host consoles asynchronous, the situation promises to be even worse than it is in the present Acnet system.

Consistent 15 Hz data acquisition can be done using a data pool scheme, but all computers in the hierarchy must be synchronous to make it work, which is likely not to be achievable with today's non-real-time operating systems. The other approach is a queued scheme, which should be realizable and should permit 15 Hz data to be reliably collected. The complication comes when the data sources span front ends.

Appendix—Example data acquisition of Linac beam signals

The application requests 15 Hz data readings of 55 devices, all of which are "sourced" from node062E, one of the two Linac nodes that are used as data servers. The reason for the quotes is that these beam-related signals aren't all connected directly to node062E, but also include signals from 5 other Linac nodes. Here is the actual distribution of signals from the various contributing nodes:

| <i>Node</i> | <i>#signals</i> |
|-------------|-----------------|
| 610 | 3 |
| 611 | 1 |
| 614 | 4 |
| 61E | 4 |
| 62E | 35 |
| 62F | 8 |

The devices that are connected to node062E are dealt with in the same way as those from other nodes, in that replies are processed to update the composite reply message. The reply that includes the device readings from node062E has to be transmitted on the token ring network to itself, then received and processed. In a sense, it is more difficult for node062E to process its own data than it is to process data from other nodes, because it has to transmit a reply message to itself. (This was done to simplify the implementation of Acnet RETDAT protocol support.) It may have been wiser not to have any significant data directly connected to node062E, although during the time that the reply message to itself is built, node062E has no fresh replies from other nodes to process. During the time that this example beam display is active, for each 15 Hz cycle, node062E transmits 1 message to itself, receives and processes 6 reply messages, and transmits one composite reply to the console client. This adds up to 120 messages per second. When another console activates the same display, another 120 messages per second must be processed by node062E.

Multiple messages destined for the same target node are concatenated inside the same datagram, in order to reduce network handling overhead. As a result, for each additional console that requests the above set of 55 devices, the datagram sizes increase, but the number of datagrams remains constant, except for an additional one transmitted to the new console.

During a kind of "stress test," a total of 560 messages per second were being processed

by node062E. The deadline time observed by node062E is 56 ms, by which time all replies from the individual Linac nodes must be received and processed. The time for processing all these replies was a total of 16 ms each 15 Hz cycle. Almost no extra time was available. It may be wise to limit the number of consoles displaying this data to a small number. With a CPU upgrade to a 68040 or something faster, we should be able to support rather more such displays.

New console support

Given a console architecture not synchronized with the 15 Hz accelerator, how can reliable 15 Hz data be collected so as not to miss cycles of data? Taking a cue from the EPICS architecture, suppose a console user could specify a callback function to be invoked when device data is updated. A callback function is invoked asynchronously with the application code. If the callback function must be short, it can place the new data in an application pool. But we need to be able to cause the application to run promptly while the data is still fresh and therefore not likely to soon be overwritten. Perhaps the application could be invoked with a reason of "new data arrival." But when there may be many such devices requested, how can it know which are new?

If there were a means of invoking an application promptly following reception of new 15 Hz data, it would give the application synchronous execution and therefore it could actually collect 15 Hz data reliably. Perhaps one could identify a single device for which an update would trigger an invocation of the application soon thereafter.