ISBN: 978-3-95450-200-4 doi:10.18429/JACoW-PCaPAC2018-WEP19

TINE RELEASE 5.0: A FIRST LOOK

P. Duval, J. Szczesny, T. Tempel, DESY, Hamburg, Germany
S. Weisse, DESY, Zeuthen, Germany
M. Nikolova, EMBL-Hamburg, Germany
J. Bobnar, Cosylab, Ljubljana, Slovenia

Abstract

The TINE [1] control system evolved in great part to meet the needs of controlling a large accelerator the size of HERA, where not only the size of the machine and efficient online data display and analysis were determining criteria, but also the seamless integration of many different platforms and programming languages. Although there has been continuous development and improvement during the operation of PETRA, it has now been 10 years since the last major release (version 4). Introducing a new major release necessarily implies a restructuring of the protocol headers and a tacit guarantee that it be compatible with its predecessors, as any logical deployment and upgrade strategy will entail operating in a mixed environment. We report here on the newest features of TINE Release 5.0 and on first experiences in its initial deployment.

INTRODUCTION

Originally a spin-off of the ISOLDE control system [2], TINE is both a mature control system, where a great deal of development has gone into the control system protocol itself, offering a multi-faceted and flexible API with many alternatives for solving data flow problems, and it is a modern control system, capable of being used with both cutting-edge and legacy technology. In addition to publish-subscribe and client-server transactions offered by many other control systems, TINE supports multi-casting and contract coercion [3]. As the TINE kernel is written in straight C and based on Berkeley sockets, it has been ported to most available operating systems. Java TINE, with all of its features, is written entirely in Java (i.e. no Java Native Interface). All other platforms, from .NET to Matlab to LabView to Python, make use of interoperability with the primary TINE kernel library. Furthermore, any client or server application based on TINE and its central services does not require any non-standard or third party software (i.e. there are no LDAP, MySQL, Oracle, Log4j, etc. dependencies).

The transition to TINE Release 4.0 was reported some time ago [4], where numerous features of TINE were enumerated, some of which (e.g. multicasting, redirection, structured data) set it apart from other control systems in common use. In addition, TINE offers a wide variety of features designed for efficient data transport and communication in large systems.

A series of meetings in 2012 identified long-term goals and established a roadmap for the future Release 5.0. Many of these goals have been realized over the past several years, showing up in new minor release versions of TINE, the last being version 4.6.3. What sets Release 5.0 apart and warrants a new major release number are some necessary changes to the protocol headers.

In the following we will identify and discuss those relevant embellishments which have ensued since the 2012 meetings and have culminated in TINE Release 5.0.

RELEASE 4 ISSUES

As noted in the introduction, a general collaboration meeting in 2012 identified certain aspects which needed to be addressed. These include the following.

Protocol Issues

The TINE protocol makes use of Berkeley sockets and TINE Release 4 originally did not properly support IP version 6 (IPv6), as the socket API calls used were all IPv4 centric. Although there is no mad rush to use IPv6, it does offer advantages which could be of interest in the not too distant future.

Header Issues

Several nice-to-have features, which potentially make life easier for administrators tracking connectivity problems, could only be added by expanding the existing protocol headers (and thereby requiring a new major release). For instance the process ID and application type of a connected client are not available under Release 4.

In addition, some supported features required workarounds under some circumstances, which could also only be ironed out by additional information not currently available in the Release 4 protocol headers. For instance, a generic client making a request to a server for a property's canonical data set can ask for the DEFAULT data set (and thus avoid an independent query to obtain the property characteristics). The returned data header will in fact provide the proper data format, but not explicitly give the correct data size. The latter can usually be inferred from the number of data bytes returned. However, if the request in question was truncated by the server, then the property data size which *should* be used in a request is an unknown quantity.

Finally, large data sets often require packet reassembly in the TINE kernel. For example, IPv4 jumbo datagrams can have a maximum length of 64 Kbytes. Any larger data set will require assembling multiple packets. In Release 4, the request and response headers hold the total message size in bytes in an unsigned short, i.e. precisely the 64 Kbytes of an IPv4 jumbo datagram. TINE transfers can of course use a TCP stream, or shared memory, rather than datagrams, but the same packet reassembly exists.

ISBN: 978-3-95450-200-4 doi:10.18429/JACoW-PCaPAC2018-WEP19

That in itself is not a problem, except that it is often useful to specify a larger number for the message size in bytes, necessitating a 4-byte integer in the transport headers, rather than the current 2-byte integer.

Other Issues

A TINE server developer can choose among a variety of platforms on which to write his server, including Java, Python, LabView, Matlab, and .NET, not to mention the operating system. Nevertheless, a number of production servers are written in C or C++, making direct use of the C library API. C++ developers are most likely to make use of Standard Template Library (STL) or Microsoft Foundation Classes (MFC) libraries and headers. If this is indeed the case, then certain measures must be taken to avoid namespace collisions when *tine.h* is included in the same code module as the STL or MFC headers. This primarily has to do with macro definitions attempting to override e.g. a class name and cannot be trivially solved by using a namespace wrapper around *tine.h*.

RELEASE 5 SOLUTIONS

Protocol Issues

TINE Release 4.5.0 introduced the standard IPv6 socket API to the TINE library and by Release 4.6.3 the TINE libraries in both C and Java were fully implemented. The general strategy is for clients and servers to make use of a dual stack if possible, where a single bound listening socket can support either protocol. IPv4 clients will then only ever see an IPv4 address. Likewise an IPv6 client will always see an IPv6 address, be it a real one or a mapped IPv4 address (with a leading '::ffff:'. Thus this aspect was concluded prior to the advent of Release 5.0.

Aside from removing the administrative headaches involved in making use of private networks and exhausting the IPv4 address space, IPv6 also offers jumbo datagrams up to 4,294,967,295 bytes.

Header Issues

The TINE Release 5 request headers have indeed been modified to pass a client's process ID and application type to a server, along with associated diagnostics which pass this information along (see Figure 1). The application type is composed of an 8-character string identifying the principal kind of client making the call. A middle layer server acting as a client will supply the text "FEC", for instance, whereas a Python client will supply the text "PyTine". A client's process ID is perhaps of little or no use if the client is a command line tool such as *tget* used in a script. However, for persistent clients it is a useful identification number which can expedite the search for a specific client application should it become problematic.

The new response headers also categorically supply a contract's canonical data size as well as the size in bytes and in elements returned in the call.

Entities such as the message size or MTU are also now categorically 4-byte integers.

A contract response header also continues to supply additional system and user *stamps* as 4-byte unsigned integers. These are in addition to the associated data's time stamp, and are typically used to provide an event or cycle number tag to the associated data. That is, these were specifically *not* upgraded to 8-byte integers, primarily to avoid issues on 32-bit (or 16-bit) platforms which do not support them. A quantity such as an event number will wrap only every 14 years or so even if incremented at 10 Hz, so this should not present a problem in the short term, and will essentially never present a problem if the said *event number* is reset at the beginning of a run. In the long term, these quantities can be upgraded at some future time, should the need arise.

Both the request and response headers also provide information on the endianness of the host machine and the character encoding in use in the data provided. In the current release (5.0.0) the endianness is fixed as little endian and the character set is fixed as ASCII standard. One could argue that a modicum of efficiency could be squeezed from the system if only one of a client-server pair engaged in byte swapping when it needed to. However, as a Release 5 server will need to support Release 4 clients, which expect little endian payloads, it would have to make the decision to swap or not at the point of delivery. This is currently problematic for some data types, such as a user-defined tagged structure and some non-fixed length data types and requires extensive refactoring of code. Therefore this issue has been postponed for some future release, which of course will have to make the identical swapping decision based on a client having a release number greater than e.g. 5.2, etc.

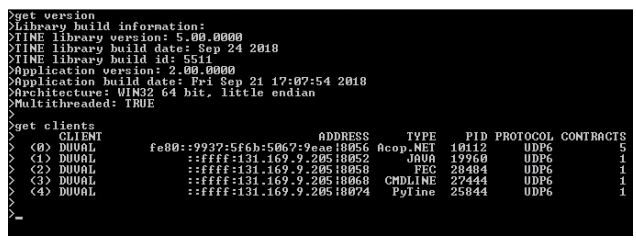

Figure 1: An example of a server's console command to show its attached clients. New to Release 5 are the PID and TYPE columns.

Other Issues

The problematic macro definitions (largely error/status codes) have essentially all be replaced with enumerations in Release 5. This has the advantage (as in the case of a macro definition) of not requiring constant variables in a program's data segment. Furthermore enumerations easily lend themselves to being used within a C++ namespace wrapper. Thus using the TINE API directly in C++ code no longer requires any extra measures to avoid namespace collisions. A namespace wrapper around the TINE header file *tine.h* is entirely optional.

UPGRADE STRATEGY

Any control system component making use of TINE Release 5 must be fully compatible with earlier releases

ISBN: 978-3-95450-200-4 doi:10.18429/JACoW-PCaPAC2018-WEP19

of TINE (predominately of Release 4 vintage). Release 5 servers must seamlessly interface with Release 4 (or Release 3) clients. And Release 5 clients must likewise be able to access earlier vintage servers. With this as an ansatz we can contemplate upgrading the control system elements adiabatically, with the expectation that legacy components will remain operational for months, if not years.

There is no *best moment* to roll out a new major release such as this, other than perhaps during a long shutdown, where there is often a prolonged re-animation of the machine. This happens infrequently. In any event, in the case of the PETRA III complex, no amount of unit testing will catch all compatibility issues, largely due to the multicultural aspects found in machine control there. For instance, there are critical Java servers running on both Windows and Linux hosts. There are 32-bit and 64-bit servers running not only on Windows and Linux hosts, but on VxWorks and LabView (also Windows) as well. Client applications are liable to be rich client Java applications using ACOP beans or jDDD (with its own complexities) or rich clients using Matlab, LabView, or using ACOP.NET [5].

As TINE is feature rich, there tends to be a wide variety of *ways to do things*. This in itself tends to increase the general entropy in a test environment.

Thus the path to general deployment was to test as much as possible, making use of the TINE unit server and client in combinations of Release 4 interfacing with Release 5, and then to deploy and react during the machine studies following a *mini* shutdown and prior to a user run. Here one can see which hiccups occur during normal operations and either rollback if necessary or find and fix (if the operators can tolerate the hiccup during a bit of extreme programing).

In fact, there were surprisingly few hiccups - three to be exact, two of which led to a rollback. Nonetheless, at the conclusion of the machine studies, TINE Release 5 was in place as the de-facto standard, although there will be a mixed scenario for some time to come. And to be sure, one still must continue to be on the lookout for hiccups and be ready to react.

LESSONS LEARNED

With the rollout of TINE Release 5, one generally hopes that, as far as the users and customers are concerned, *nobody notices anything*. That is to say, there are no new bells and whistles that would make transfers more efficient or offer new paradigms for application development. The API is basically unchanged. On the other hand, developers (especially server developers using the C library) will appreciate many of the new embellishments. Likewise administrators will find it easier to track communication problems.

The TINE core team will also be able to navigate through both the Java and C library code more easily, due to extensive refactoring. And as the latest TINE transport headers are extensible, it should prove to be a straightforward task to add fields to the existing headers at some time in the future should they be needed.

One somehow anticipates that by the mere act of shaking things up, i.e. not letting sleeping dogs lie, so to speak, various real problems (e.g. hidden race conditions) will be exposed. In the initial phase of the ensuing users run, two further upgrade issues in fact became apparent. One of these, a long-standing TCP issue which might occur when large input data sets are being collected at the server side, had *almost* no chance of expression in the Release-4 world and only became visible when the contract request headers increased in size in Release 5. This issue surfaced on a particular server and led to a local rollback until it was understood. The second issue involved a check on multi-channel contract coercion logic versus the minor release and revision numbers, which suddenly jumped back to 0 and 0. This latter issue had no *visible* consequences and was only noticed in that certain applications appeared to suffer in certain aspects of transfer efficiency. Both of these problems were promptly dealt with and neither had any direct bearing on the user run.

Introducing a new major release (or any systematic upgrade, for that matter) is not something one takes lightly at any time. In the absence of a full-blown mock facility which is actually used under *real* conditions, there is virtually no way to *catch things* other than to deploy and standby in extreme programming mode. All in all, there were surprisingly few hiccups due to specific software problems.

An additional hiccup was due to the non-synchronized deployment of system libraries. Although it had nothing to do with any software problems or Release 4/Release 5 compatibility issues, it would of course not have arisen had there been no attempt at an upgrade. Yet, this in itself exposed an existing problem (in this case, a misunderstanding in the software deployment on Windows hosts).

The moral of the story is: It sometimes takes a user run to expose problems! By now the dust has settled, so to speak, and one is gradually beginning to breathe more easily.

REFERENCES

[1] TINE website; http://tine.desy.de

[2] R. Billings *et al.*, "A PC Based Control System for the CERN ISOLDE Separators", in *Proc. ICALEPCS'91*, Tsukuba, Japan, Nov. 1991.

[3] P. Duval and S. Herb, "The TINE Control System Protocol: How to achieve high scalability and performance", in *Proc. PCaPAC'10,* Saskatoon, Canada, Oct. 2010, paper WECOAA02.

[4] P. Duval *et al.*, "TINE Release 4 in Operation", in *Proc. PCaPAC'08,* Ljubljana, Slovenia, Oct. 2008, paper MOX01.

[5] P. Duval *et al.*, "ACOP.NET: Not Just Another GUI Builder", presented at PCaPAC'18, Hsinchu, Taiwan, Oct. 2018, paper THCB1, this conference (and references therein).