ASSESSING THE IMPACT OF SOFTWARE-DEFINED RADIO TECHNOLOGY ON WAVEFORM THROUGHPUT DELAY

Charles A. Linn (Harris Corporation, Rochester, NY USA email: Charles.Linn@harris.com)

ABSTRACT

Compared with the analog hardware based radios of past years, the modern software-defined radio provides an unparalleled degree of capability and versatility, enabling complex waveforms that were inconceivable in the past to be implemented with ease. Legacy waveforms and the systems dependent on them continue to persist in the present day, however, and the new technology has challenges meeting certain parameters that were not issues in older, analog radios. Data throughput delay, the end-to-end time required to transmit data through a radio system, is one of these "problem" parameters.

This paper examines from both a theoretical and practical standpoint the modeling of Throughput Delay (TD) and its effect on system performance. Examples are taken from both the Joint Tactical Radio System (JTRS) system [1] and legacy waveforms employed by that program. Techniques that either minimize the delays themselves or change the nature of the end-to-end communications system such that the delay changes are largely inconsequential are discussed, as well as the role of evolving platform processing capability in addressing this problem.

1. INTRODUCTION

During the last four decades both military and commercial radio systems have been undergoing a transition from analog voice to digital (or digital voice) communications systems. This transition has resulted in the evolution of a vast array of digital communications waveforms and associated data transmission equipment to support this growing need.

During the same time period, the radio platforms supporting these waveforms have been undergoing a different evolution. Whereas as little as 15 years ago the vast majority of radio platforms were principally implemented using dedicated analog and digital hardware, the last decade has seen a strong paradigm shift towards software-defined radios and their corresponding waveform implementations.

Although in most respects these two trends have been synergistic, there exist several well-known areas where these two trends have been in tension. One such area is that of waveform throughput delay (here, TD). Throughput delay can be given many specific definitions, but generally describes the absolute time delay it takes to move a piece of data through a communications system. In some systems, TD is non-critical and has a relatively minor impact of the operation of the system – a millisecond-level increase in a minute-level message. In other systems, however relatively minor changes in TD can have significant impacts of system performance or user quality-of-service (QoS).

To alleviate the problems caused by TD, two approaches can be taken. The first is to identify and optimize the individual areas that contribute to the overall delay. A second approach, when practical, is to re-factor the overall system such that it accomplishes the same end but in a way less impacted by the added delay. Both techniques should be employed to be able to enjoy the beneficial features of software-defined radios without paying significant performance costs.

2. THROUGHPUT DELAY MODELING

Before we can optimize TD, it pays to understand the component contributions to the problem. A communications channel can be defined as consisting of "the part that connects a data source to a data sink" [2]. Since the focus in this paper is on software-defined radios, we will further limit this definition to apply just between two radios in a system as connected by a given "waveform" under consideration.

Several types of communications channels are used in current systems. In government and military systems, these channels could be classified as follows:

1. Analog channel (voice or audio information)
2. Digital voice communications (synchronous)
3. Synchronous data channel
4. Asynchronous data channel
5. Packet-based (usually IP-based) data channel

Of these channel types, this paper will primarily concentrate on the synchronous digital voice and data channel types. From a system performance point-of-view, these tend to be the most sensitive to TD variations – in the case of digital voice, the additional delay is annoying to the user, while synchronous data is often associated with ARQ data links inside the data source/sink.

Synchronous data transmissions systems are characterized as continuous data flow systems. With some systems there is an initial "request-to-send / clear-to-send" handshake to initiate the flow. Once data flow begins, however, the Data Terminal Equipment (DTE) supplies continuous data and associated clock at a fixed channel rate. The delivery continues *without interruption* until the transmit session ("message") is finished. Similarly, on the receiving end, once data arrives and has propagated itself through the radio, it is presented to the radio as a continuous, constant rate flow until end-of-message.

The "data-path" processing from data source to antenna (or from antenna to data source) can typically be generalized as a chain of processing elements, characterized as shown in Figure 1. Each element N receives "blocks" of data from the previous element (N-1), except element 1, which initially collects and presents the data into the system.

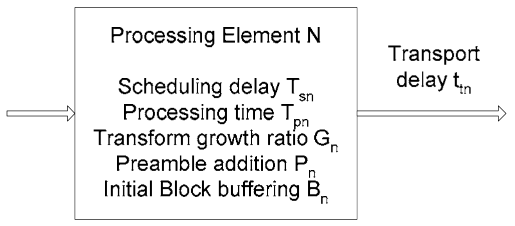

Figure 1: Processing Element Parameters

For a given element, several parameters are considered:

- Once the data has been transported into the proper address space, the operating system must "schedule" the block for execution. This time is represented by t_{sn}.
- The element must then process the data, which is represented in time by t_{pn}.
- As part of its processing, the element may grow or shrink the data rate by a multiplicative factor G_n. Examples of this growth would be the application of error-correcting or detecting codes, upsampling, etc.
- The element could add data before the first block of data (a preamble of some sort). This is assumed to be at the output data rate, and represented by size P_n.
- The element could potentially buffer data, i.e. absorb one or more input data blocks before outputting a block. This generalized buffer behavior is symbolized by B_n, which is the amount of data that must be initially received before the first output block can be issued.
- Finally, the output is sent to the next element. The time required to transport the output data to the proper address space, etc. for subsequent processing is assigned a time of t_{tn}.

There are certainly more parameters that one could characterize, but these are most important in analyzing the TD of a system. Here it is important to note that in a synchronous data system, since data flows continuously *once it starts*, when considering TD one only needs to consider the "leading edge" of the wave for data – for once at any stage in the system data starts flowing, if the average constant rate is not maintained, the system fails in real time. For this reason, it is unimportant here if, for example a element introduces a postamble, as this does not affect the leading edge of the data.

Given these parameters, let us look at the overall delay through several stages. Excluding the impact of the buffering term (which is discussed later), the delay can be calculated as follows:

$$t_{tot} = \sum_{n=1}^{k} \left[t_{sn} + t_{pn} + t_{tn} + \left(\frac{P_n}{\prod_{m=1}^{n} G_m} \right) T_{b0} \right]$$

This formula is simpler than it appears. The first three terms inside the sum simply represent the additive delay of the scheduling, computation and transport delays. The second term merely accounts for the time to transmit a given stage's preamble as related back to the source bit period (T_{b0}) by using the growth ratio G_n of all previous stages. It could just as easily be expressed as a given stages preamble data length divided by the output rate of that stage. With this understanding, it merely states that the total delay is equal to the sum of all the component delays plus the extra delays introduced by the preambles.

Two additional factors frequently influence TD beyond that in the formula above. The first is the B_n parameter, which represents the "initial buffering delay". This is a key factor in influencing TD, but cannot be easily expressed in closed form. For the first element the action of this term is straightforward – B_1 bits need to be first accumulated before the output can be issued at all. This "startup buffering delay" is almost always present in software-based systems, since usually at least a byte (8 bits) of data or more needs to be accumulated before any processing begins. For subsequent stages, however, the effect of such buffering is more complex. First of all, it is important to remember that only the initial "data wavefront" need to considered – so for a given stage, the only thing that matters is if the first output data block will be issued as a result of the first input block of data, or the second, etc. If first received block results in the issuance of an output block, than that stage's the data wave propagates with no buffering delay (and hence has not impact). If additional input blocks are required, however, the TD is increased by the time equivalent to an integral number of data blocks timed at the input of the processing element. For example, if an processing element contains a block interleaver that requires 110 bytes of data, and if data is arriving in 12 byte blocks at a rate of 2000 bytes per second (166.678 blocks per second), the TD *added* by the interleaver would be 60 ms, as it takes 10 blocks of data to deliver (and somewhat exceed) the required data. On the other hand, if data was arriving in 200 byte blocks, the interleaver would add no output delay, as the required amount of data would already be satisfied.

There is a common misconception that when multiple processing elements are "chained", you pay a TD time for each element, since the pipeline "must be filled". While true in many clocked hardware systems, in the typical SW radio case this is often not true. If a proper starting block size (B_1) is chosen to be large enough for downstream blocks, then the first packet of data will flow through the system without additional buffering delays – just the scheduling, processing and propagation times will be additive.

A second additional factor is the time that passes from the initial "data request" (typically represented by "RTS", or request-to-send) to the "clear-to-send" signal, ("CTS") issued to the data source. During this period of time the source wants to send data, but is not allowed to send it. For the purpose of this paper this is considered part of the overall TD period, as it has the same effect of "delaying" the data to its definition as waveform "pipeline delay" alone has.

To tie these concepts together, let us consider the simple representative example shown in Figure 2. Data is received by a serial interface at 16kbps. The element 1 output data "block size" is 128 bits (16 bytes), giving a "block rate" of 8 ms per block – typical for a SW radio. It is then encrypted, which adds a 1000 bit preamble, then encoded using a 2:1 code. The resultant bit stream is then sent to a modulator which modulates it to a 96 K symbol/second stream in conjunction with a 8,000 symbol modem preamble. Given the sample parameters supplied, T_{tot} is calculated at 103.3 ms, to which we add an additional 8 ms to account for the initial stage's buffering of 128 bits, giving an overall TD of 111.3 ms.

3. THROUGHPUT DELAY SYSTEM IMPACT

In many cases a moderate increase in TD has negligible effects on overall system performance. For example, if a synchronous data link is used to transmit a video image (without error correction, which is often acceptable in this case), a 50 ms increase in TD is trivial compared with, for example a 10 minute image transmission time. However, when an external data link protocol is employed outside of the radio itself, performance problems can become more significant. These systems typically use Automatic Repeat Request (ARQ) and TDMA techniques that involve rapid changes of the (half-duplex) radio link between transmit and receive or precise positioning of transmissions in time over the channel. In ARQ data links, a single long message is broken into multiple "slices", with each slice sent, checked at the far end, with an acknowledgement of receipt returned before the next slice is sent (or the previous block is resent). In these systems since the transmissions themselves are very short, an increase in throughput delay constitutes a larger proportion of the channel time. This alone tends to cause a moderate decrease in system performance.

When these systems are combined with multiple channel access techniques, however, the problem can be compounded. In these systems, there are potentially multiple nodes competing for the channel. Most systems of this type use some sort of traffic sensing (carrier sense) on the channel prior to transmission, i.e. "listen before transmit". The problem is that it takes a finite amount of time for the radio to detect the traffic and propagate this information to the data link. In addition, once a given node decides the channel is clear, it takes a finite amount of time (throughput delay) to actually start transmitting on the channel, in addition to the time it takes for the RF wavefront to propagate to the other nodes. During this critical time, other nodes could also be beginning similar transmissions, resulting in collisions between traffic, at which point the entire process begins again. Although many different

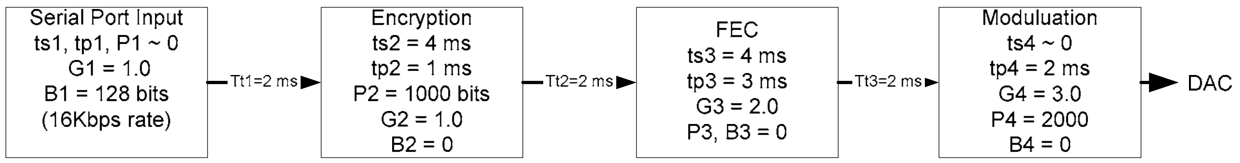

Figure 2: Example Processing Chain

protocols have been developed to avoid unnecessary collisions, for all practical purposes the transmit waveform's "key-time to traffic on channel" (which falls within the TD definition for this paper) is additive to the channels propagation delay time. In addition, if the system relies on actual receive data as the only carrier sense mechanism, the receive TD time also adds in this way. In these cases, even small increases in TD can cause a significant fraction of the channel to be denied in saturated traffic conditions.

Several applicable examples in the military community would be with Mil-Std-188-220 [4] and the Mil-Std-188-184 [5] protocols, both of which are sensitive to TD and other radio delays. Another example employing TDMA techniques would be Mil-Std-188-203 (Link-11) [6]. Link 11 employs a TDMA type structure between a central station and a number of "picket" stations. Picket responses are time slotted in nature, and fixed in the standard– as a result it is up to the radio equipment to comply with the key time and TD requirements or not be usable in the system.

4. ANALOG VS. SW-DEFINED RADIO CHARACTERISTICS

Now that a model TD has been defined, we can discuss in these terms what has happened as we have evolved from an analog, hardware-based platform to today's SW-defined radio platform.

Analog platforms (at least with simple waveforms) are in many cases capable of processing the data bit by bit as it arrives. The encryption, and digital voice processing is traditionally not integrated "into" the radio, further reducing delays. The "processing elements" are often indistinct and little or no scheduling or tasking delay occurs between them. From a model standpoint, here are some typical parameters for, say a VHF-FSK waveform:

- $t_{sn} = 0$ (no processor)
- t_{pn}, $t_{tn} < 1$ ms (hardware, filter delays)
- $G_n = 1$ (inherent in FSK, no coding)
- P_n still present for modem, delays CTS return

Here we see that for this simple waveform, the TD due to the processing elements is quite small. However, the same modem preamble (usually a "dotting" 1/0 pattern) still needs to be emitted. Since the analog hardware cannot "buffer bits", this is accomplished by either delaying the return of CTS, or by having the data source supply the preamble pattern itself. However this is done, this time should be considered as TD, and will be similar to the same element on any radio platform, hardware or software based.

A modern software based radio is quite different for a number of reasons. First of all, almost all implementations process data in blocks of one or more bytes, not "by the bit". Since in many implementations the scheduling t_{sn} and transport t_{tn} times are relatively independent of data block size and significant compared with the data processing time t_{pn}, this pressures the designers to utilize still larger block sizes to limit the processing capability required to host the waveform.

At the same time, cryptographic functions, which have traditionally been external, standalone pieces of equipment, have been integrated into the radio. In addition, the characteristics of the cryptographic equipment itself are changing. Whereas most US government cryptographic devices traditionally processed data one bit at a time, both newer cryptographic algorithms as well as the software encapsulations of the "classic" algorithms frequently process multiples of bytes of data. For example, in the JTRS Security Supplement "transform" API [3], data is submitted to the cryptographic subsystem as a sequence of bytes – no provision is mode for processing less than 8 bits of data. Although in general this does not cause an increase in TD unless the block size used is greater than the output block size of the preceding stage, it does set a minimum block size of 8 bits, hence setting a minimum TD for any system.

It is hard to generalize "typical" software-based radio system parameters, but the following parameter ranges are common in radio platforms designed to the JTRS SCA[1] standard:

- t_{sn} (scheduling time): .1 to 4 ms
- t_{pn} (processing time): .5 – 20 ms, varies with function, block size, etc.
- t_{tn} (transport time): .6 – 3 ms minimum, plus data copy time
- B_1 (initial buffering size, first element in chain): 4 – 20 ms.

The other parameters (G_n and intermediate buffering size B_n) are typically a function of waveform design and not specifically influenced by the SDR architecture per-se.

In summary, when one considers all of the factors it can be difficult to perform true "apples to apples" comparisons between older, analog radios and today's SDRs. However, in many cases the additive delays plus a typical initial buffering delay typically adds an additional 20 – 30 ms to an SDR implementation compared with its analog brethren.

5. REDUCING SDR THROUGHPUT DELAY

To reduce the TD in a system, one works by minimizing the individual contributors in conjunction with intelligent architecting of the waveform itself.

Minimizing t_{sn} (scheduling delay): in an SDR, this represents the time when a given process element can logically begin processing, but has not yet been allotted time (scheduled) by the operation system to do so. Two things are required in most operating systems– the process (or thread) must become the highest priority available to run, and the "context" switch to that new task must be accomplished. Here standard embedded system optimization techniques apply—ensure that the waveform runs at a higher priority than other, less critical functions, and avoid priority inversion situations.

Several techniques can be used to minimize the scheduling delay. First, the sequence of scheduling itself can be modeled to ensure no more context switches are required than are necessary. For example, when an upstream element A sends data to element B, depending on the OS it may be beneficial to have element A at a higher priority than B, so A can send its data, block, and *only then* have a single context switch to B (vs. an A $\rightarrow$ B <runs, blocks> $\rightarrow$ return to A <which blocks> sequence). Many OS's address this problem by, when several processes are eligible to run at the same priority, allowing the currently executing process to complete its processing before the other processes can be scheduled. This of course needs to be traded off with OS "fairness" policy.

A second technique is to minimize the context switch time itself. Operating systems that implement "heavyweight" processes involving separate address spaces provide a useful isolation mechanism, but it comes at a significant cost when context switches must be performed. Consider instead using either a lightweight thread tasking model, or, if possible, implement the two components in the same address space involving a simple "call" relationship. This technique not only minimizes the scheduling delay, but also simultaneously minimizes the transport delay t_{tn} as well.

Minimizing t_{tn} (transport delay): To minimize t_{tn}, first work to collocate components in the same process space, and if possible in the same thread space. In some cases this allows the "transport" to be optimized to a simple function call, with dramatic savings. When compared with a typical waveform using an "each component in its own process space" approach, tens of milliseconds in TD can be potentially saved in this way.

In architectures (such as JTRS) where a middleware layer such as CORBA is being used, be sure that you are using an efficient transport mechanism. Often the "standard" transport mechanism is based on TCP/IP, which is very inefficient and often overkill compared with alternate transport mechanisms such as shared memory or OS message passing protocols.

Choosing an optimal data block size: Several factors affect the choice of block size, and in this area minimizing TD is in tension with minimizing processing power. For example, compare two otherwise identical systems where only the block size defined by the most upstream component (B1) varies. Since in many systems the most upstream block size is used throughout the entire element chain, and the time it takes to collect this data block is proportional to the block size, it is obvious that a system with a one byte block size is going to have both a lower TD and a higher required processing load than a system with a block size of, say 256 bytes. Although many systems use a fixed block size (usually based on the highest data rate in the system), this can result in very long TDs at the lower data rates, even though spare processing power may be available at these lower data rates. Since most systems support multiple data rates, it also appears likely that different block sizes probably make sense for different data rates. To address this, the data sources at the "edge" of a waveform (such as audio and data interfaces in the transmit directions, and the receiving modem FPGA / DSP component in "receive") should be configurable to allow different block sizes. For example, it would be useful to have a configuration property "desired block size" in a Serial Port Device to allow the waveform to set this parameter to appropriate values as a function of data rate and waveform needs.

But what block size should a waveform choose? It is usually best to strive for a (mostly) common "block rate", independent of data rate. For a small, battery powered system this "block rate" may be 20 ms per block, while for a more powerful platform a block size of closer to 8 ms could be used. The block size, where possible should be chosen to be at least as large as the largest initial buffer (max of B_n, adjusted to match the source data rate) in any downstream components. For example, if in a system with a 256 byte interleaver block somewhere downstream, having the Serial Port Device dole out 8 byte packets is most likely to cost processing power without improving TD. When calculating these levels, it is important to also consider the additive effects of any preambles as well as the data rate growth ratios (G_n) in the calculations.

Carefully design clock correction functions: Often in synchronous systems so-called "clock correction" is required. Although outside of the scope of this paper, clock correction is the overall process whereas the transmit signal-in-space timing is adjusted to match the DTE Tx clock, and conversely, where the receive Rx clock supplied to the DTE is adjusted to match that of the incoming signaling.

In hardware based systems, clock correction was a rather straightforward phase-lock-loop problem. In software systems, however, the "control loop" approach is difficult to implement without use of "measure buffers", a.k.a. "leaky buckets". In these systems, changes to the data level in a buffer are used to make clock adjustments. While simple to implement, the data level in the bucket adds to TD. A harder but alternate "open loop" approach minimizes this problem by directly measuring clock procession on one side of the system and communicating the desired changes to the other side.

Optimize RTS / CTS ("keyup") delay: Another contributor to TD is the time it takes for the system to accept data for transmission. In general, standard optimization techniques should be considered to reduce this critical parameter in the system. From the specific context of minimizing TD, however, several other techniques can be employed. The first technique, usable when the keyup delay is rather deterministic, allow the data to start flowing (and hence filling the system buffers) in parallel with preparing the rest of the system for transmission. While effective, this technique is fraught with danger, however. With a synchronous interface, once data flow begins, it cannot be stopped. If for any reason the system is unable to transmit, or takes much longer than anticipated, internal buffers could overflow, and data could potentially be lost from the system. This technique can also lead to the dreaded "execution time dependent software package", and even minor delays in execution could cause system failure.

A similar and somewhat safer technique is to allow preamble transmission to begin before the first data packet has been buffered from the most upstream data device. Consider a system with a encryption preamble of 100 ms operating in conjunction with an upstream Serial Port Device that is configured to provide data blocks at a 16ms block rate. If, one the system is ready to transmit data, the encryption preamble can be passed downstream immediately simultaneous with the issuance of CTS to the data device. In this way, 16 ms of throughput delay is saved. Care must be taken, however to ensure that the transmitted preamble is longer (at a minimum) than the first packet buffer time – otherwise the system could "run dry" of data while waiting for the first packet (a system failure).

6. MIMIMIZING THROUGHPUT DELAY IMPACT

Although not always an option, perhaps the best solution to optimize system performance is to refactor the system to be less sensitive to TD as well as other system timing variations (which also tend to be higher on SDRs compared to legacy hardware-based radios). To do this, external functions are drawn "inside" the waveform, where it is possible to better control the timing. ARQ and TDMA protocols should if possible be run on the non-encrypted side to provide for tighter control of timing as well as to minimize "downstream" TD contributions. Not only does this make the system "tighter", but also does not constrain the system from communicating in critical areas using legacy hardware interfaces.

For an example, when Mil-Std-188-220 [4] is moved from an external piece of equipment to an internal function, some TD is immediately saved because the data link does not need to stream this data to the radio at a fixed bit rate – rather it simply sends a packet downstream. In this way, the "first block" collection time is saved. Further savings are realized since the system does not have to perform clock correction functions, and better signal presence communications are available. Using similar reasoning, internally hosted TDMA systems can more accurately position over-the-air signaling compared with an external unit.

Unfortunately, in many cases the legacy standards are written in such a way that prevents taking full advantage with embedment. Still, it is important to keep these techniques in mind as newer waveforms are designed.

7. CONCLUSIONS

With an understanding of what factors can contribute to TD in a radio coupled with knowledge on how TD can affect an overall system, platform and waveform architects can work to minimize this parameter. While TD can never be eliminated or reduced to match the levels in prior analog radios, yet one more "challenge" can be removed from the promise that software-defined-radios have to offer.

10. REFERENCES

[1] Joint Tactical Radio Systems (JTRS) Joint Program Office. "Software Communications Architecture Specification" *Document JTRS-5000 SCA V3.0,* August 27, 2004.

[2] American National Standards Institute, Inc., "ATIS Telecom Glossary 2000" *Document T1.523-2001*, www.atis.org, 28 Februrary, 2001.

[3] Joint Tactical Radio Systems (JTRS) Joint Program Office. "Security Supplement to the Software Communications Architecture Specification" *Document JTRS-5000 SEC v3.0,* August 27, 2004.

[4] Department of Defense. " DIGITAL MESSAGE TRANSFER DEVICE SUBSYSTEMS" *MIL-STD-188-220C*. 22 May, 2002

[5] Department of Defense. "INTEROPERABILITY AND PERFORMANCE STANDARD FOR THE DATA CONTROL WAVEFORM" *MIL-STD-188-184*. 20 August, 1993.

[6] Department of Defense. "MILITARY STANDARD Interoperability and Performance Standards for Tactical Digital Information Link (TADIL) A" *MIL-STD-188-203-1A*. 8 January, 1988.