fall '22

CSC 412: Computer Networks ([redacted])

Home-work programming project

Analyzing the Impact of clock-rate drifts at sender/receiver stations on data packet lengths

Project duration: 8th Oct. – 24 Oct! 22

Instructor: *K. Ravindran*

Email your HW reports (as PDF) to: *ravir222@hotmail.com*

Packetized data transmission is a foundational concept in the design of computer networks. An example is the sending of a 300-*kbyte* picture as 250 packets of 1200-bytes each over a cellular network. As another example, a live video feed may involve sending, say, 256-512-*kbyte* sized video frames at 25 fps (frames/sec) where the data network element segments each video frame into 400-500 packets of variable sizes (0.8-1.5 *kbyte* each). One fundamental problem in the send/receive of data packets is still about the timing of bit-level transmissions for each packet over the physical links/channels. The home-work project is about the bit-level synchronization of a receiver with a packet sender (for correct reception of the data bits).

A prelude for clocked transmission of packet data-bits

Clocked pulses at a sender station X trigger the injection of data- bits of a packet p as electrical signals on the physical link l connecting to a receiver station Y. Receipt of these bits at Y is triggered by a local clock running in station Y synchronously to read the signals from link l. For a correct reception of data-bits, the clocks in X and Y should run synchronously, as below:

r1: The clock-pulses at X and Y to trigger the start of send and receipt of packet p should occur at the same time;

r2: Both the clocks at X and Y should run at the same rate, say, C bits/sec, during the transmission of packet p.

The requirement **r1** is met by a preamble based clock-reset mechanism in receiver hardware that precedes the start of every packet transmission (e.g., saying "hello" to start a sentence)[1]. Meeting the requirement **r2**, on the other hand, is tied to the quality of hardware clock designs in order to provide a synchronized reception of data-bits over long transmission periods. The home-work is about analyzing how a drift in the clock-rate C would impact this synchronization.

Consider the transmission of a data packet of length R bits (say, $R = 10,000$). When the clock-rate C drifts: say, to a lower rate, the receiver clock may go out of step relative to the start/end of a bit time-slot at sender. This in turn leads to the receiver missing out some of the signal energy during the link-sampling process for a bit-slot (the time-shift gets more pronounced in the latter bit-slots when R is large). Despite a partial miss of signal, the receiver hardware may still be able to reconstruct the bits from a misaligned time-slot — if a significant fraction of the signal energy is sampled[2]. When the time-skew from a correctly synchronized bit-slot is high, the sampled signal energy may not be sufficient enough for a correct determination of the data bit. This missing of data-bit(s) is referred to as a *bit-slip*. On the other hand, faster receiver clocks may result in an over-sampling of bit-slots: which may in turn have a receiver infer extra bits that are even not part of

[1]The hardware consists of line-signal edge detectors, edge-triggered pulsed oscillators, phase-locked loop, etc.

[2]As analogy, speaking a long sentence may cause the listener to drift-off towards the end. In contrast, a short sentence would keep the listener stay more attentive for the entire sentence, but incurs more overhead and inter-sentence pauses. The trade-off is about the optimal sentence length that keeps the listeners attentive while lowering the overhead.

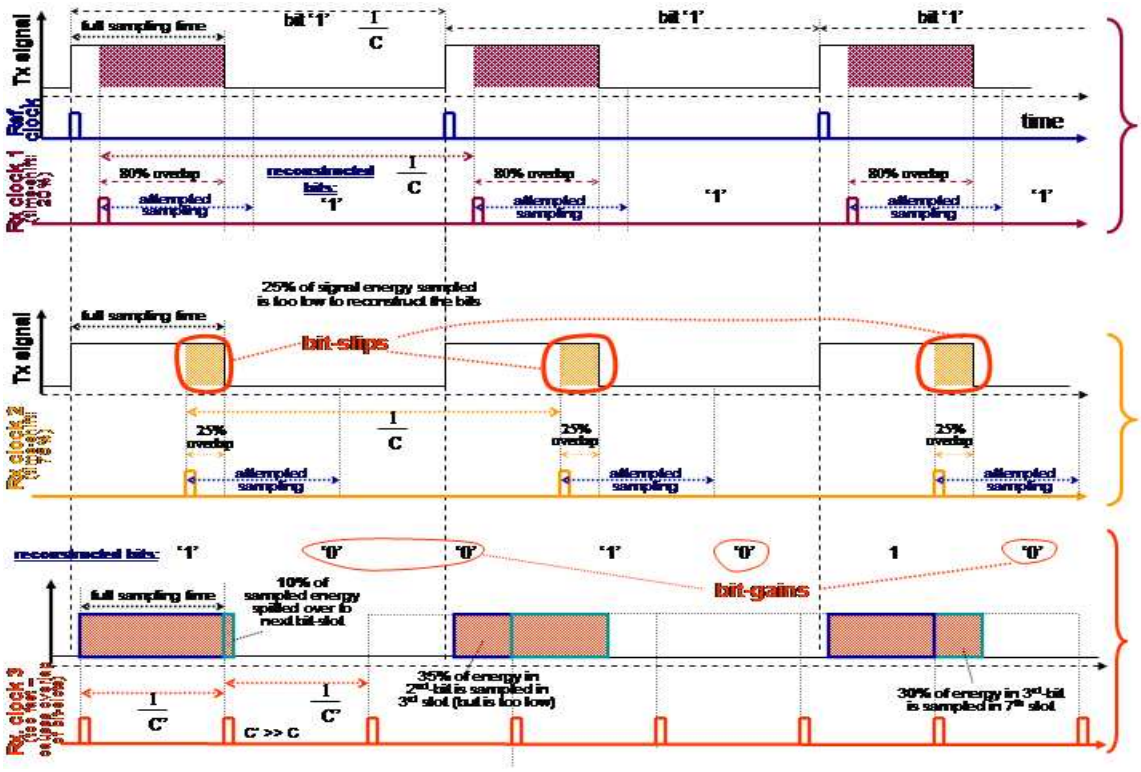

Figure 1: Illustration of bit-slip and bit-gain phenomenon during signal transmission (for packet transfer)

the sender's packet transmission — i.e., there is a *bit-gain*. See Figure 1 for illustration. Whether a bit-slip or bit-gain[3], the link-layer packet-validation mechanism will declare such a received packet as corrupted and discard it[4].

Effect of clock-drifts on choice of packet sizes

Let Δ denote the minimum fraction of time-slot overlap needed between the sender and receiver, for a correct reconstruction of the bits — where $0.0 \ll \Delta < 1.0$. For e.g., $\Delta = 0.4$ means that at least 40% of the signal energy contained in a bit-slot should be sampled in order for the receiver to correctly extract the data-bits of an incoming packet. So, a choice of maximum packet size R_m should be such that even the R_m^{th} bit-slot gets sampled with a fractional time-shift no higher-than $(1 - \Delta)$. Choosing a packet size higher than R_m then runs the risk of bit-slips or bit-gains during a packet reconstruction (based on the direction of clock-drift). Without loss of generality, we simply assume that a receiver clock drifts to a slower rate than the sender clock.

From application standpoint, a conservative approach of choosing a low R_m may result in more bandwidth consumption and higher data-transfer latency (to send/receive the ADUs), because of the per-packet overheads. An aggressive approach of choosing a high R_m, on the other hand, too has similar pitfalls — due to the risk of many packets getting discarded in the link-layer itself (which in turn forces a retransmission of many

[3]Hardware clocks are also used in CPU hardware to trigger the various steps of machine instruction executions. Here, a drift in CPU execution speeds may cause different interleaving of the CPU instruction executions with the external I/O activities (such as the DMA-based disk read/write and NIC data transfers). Here, an asynchronous interleaving of these events, as allowed by the bus-interconnect hardware, enables achieving the high-level parallelism & synchronization needed in an application system (despite the low-level non-determinism arising from the CPU speed changes due to clock-drift).

[4]With erroneous reconstruction of even a single-bit, the link-layer packet processing algorithms (such as parity-check and check-sum based validations) can detect the packet 'corruption' — and hence discard the entire packet.

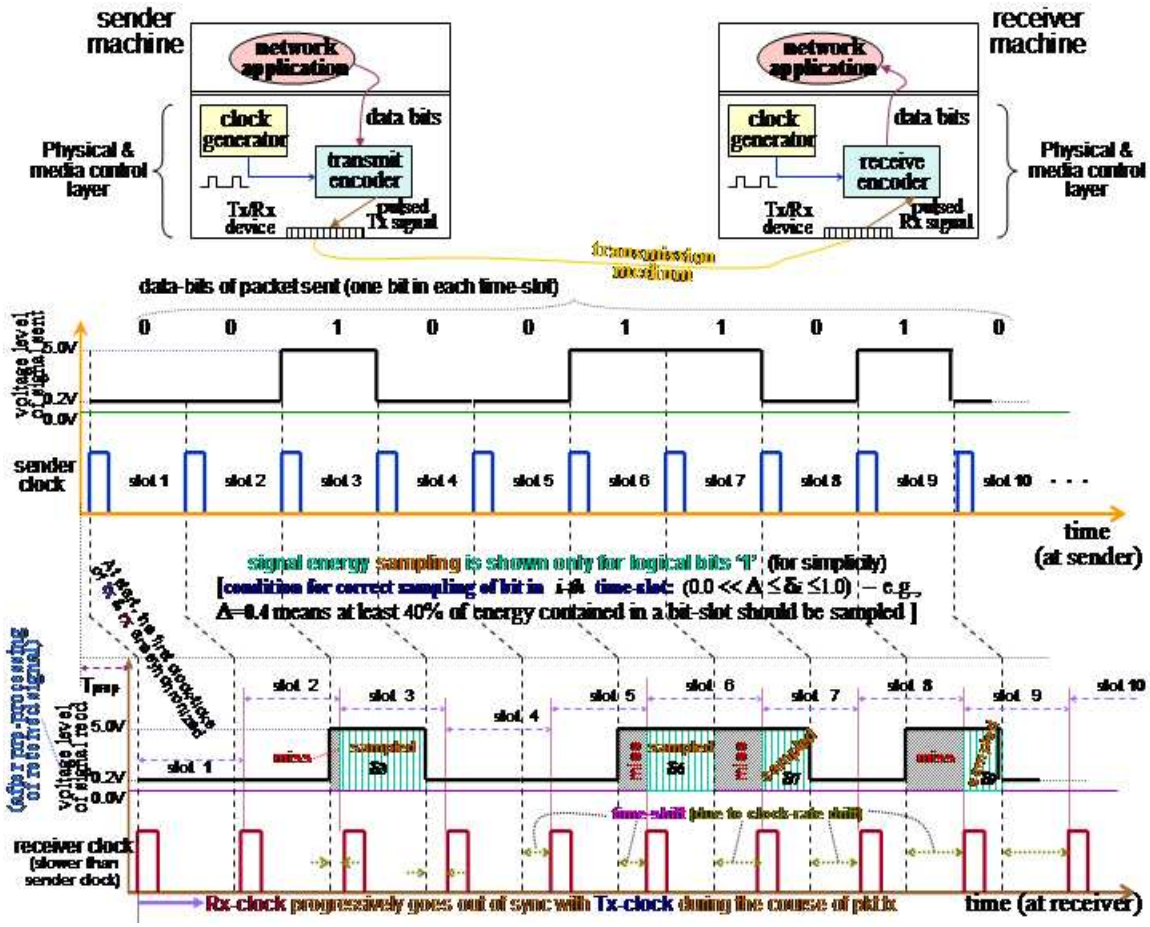

Figure 2: Illustration of time-shift parameters used in the packetization control layer

packets). An optimal choice of R_m is thus a critical element of network designs — particularly, for real-time applications (such as video and audio). Given that clock-drifts are inevitable, one should take cognizance of the above impact of clock synchronization when designing packetization mechanisms (besides other influential factors).

Manufacturers of hardware clocks specify a maximum drift rate (MDR) that a system designer should be prepared for. Empirically, MDR is given as a worst-case of shift in the bit-level time-slots at a receiver relative to the sender clock — e.g., 200 bit-level shifts out of every 1 million bits clocked per second (i.e., a maximum possible timing error is $200\mu sec$ over a 1-sec period). Quantitatively, MDR is captured as how fast a drift in the receiver clock-rate might be (at worst): $|\frac{dC}{dt}|_m$. With a progressive shifting of time-slots as bits are transmitted, the parameter $|\frac{dC}{dt}|_m$ allows a designer to estimate the maximum packet size R_m that would guarantee the safe transmission of data packets (despite the partial miss of bit-level synchronization due to the inherent drifts in clock-rates). A determination of R_m is based on two estimates at signaling hardware-level:

- How much time-shift in bit-slots would occur in a worst case during the course of transmitting R_m bits;
- whether a receiver samples enough signal energy to reconstruct the R_m^{th} bit but not enough in $(R_m+1)^{th}$ slot.

These estimates are determined from the parameters $|\frac{dC}{dt}|_m$ and Δ respectively. A packet transmission scenario shown in Figure 2 illustrates the progressive shifts in receiver clock-timing. The fraction δ_i of signal energy

sampled in i^{th} bit-slot ($i = 1, 2, \cdots, R_m$) becomes lower as i increases: i.e., $\Delta \leq \delta_i < 1.0|_{\forall i}$ with $\delta_{R_m} = \Delta$ being the limiting case. The HW project is to determine R_m by mathematically modeling the clock-sync problem and writing a numerical solver program[5].

Mathematical treatise of clock-drift aware packet size selection

The reference clock-rate C is provided by the sender's clock. For a given C, the maximum drifted clock-rate of receiver at the end of transmission of an R-bit packet is:

$$C' = C \pm (|\frac{dC}{dt}|_m * \frac{R}{C}),$$

where $1 \leq R \leq R_m$. With a progressively changing receiver clock-rate at each successive bit-slot (higher or lower), we simply use the average of total drift encountered by a packet of maximum size R_m: namely, $|\frac{dC}{dt}|_m * \frac{R_m}{2C}$, as an offset to C. In other words, we use a constant clock-rate:

$$C' = C \pm (|\frac{dC}{dt}_m| * \frac{R_m}{2C})$$

to determine each of the R_m bit-slot positions. Assuming the case of a slower receiver clock, the time-shift in R_m^{th} bit-slot should then satisfy the condition:

$$[\frac{R_m}{C'} - \frac{R_m}{C}] \leq \frac{(1-\Delta)}{C},$$

where $C' = [C - (|\frac{dC}{dt}|_m * \frac{R_m}{2C})]$. In the limiting case, the above condition is stated as:

$$[\frac{R_m}{(C - (|\frac{dC}{dt}|_m * \frac{R_m}{2C}))} - \frac{R_m}{C}] = \frac{(1-\Delta)}{C}. \quad (1)$$

Determination of R_m requires solving this non-separable form of expression (1) containing the parameters (C, R_m) and the constants $(|\frac{dC}{dt}|_m, \Delta)$.

Since a closed-form solution is not possible, the formula (1) should be solved iteratively over multiple steps of incrementally assigned values for the parameter R_m.

What you need to do as home-work project

Write a program to solve the equation (1) over iterative steps using numerical techniques. The program can be written in languages such as MATLAB, JAVA, C++, and Python (or, even C). Choose a suitable parameter granularity for use in your program's iterative steps (for better accuracy of the computed results).

Plot the computed values of R_m for different clock-drift rates $|\frac{dC}{dt}|_m$, reference clock-rate C, and signal sampling threshold Δ. You can start with parameter values: $|\frac{dC}{dt}|_m = 200 * 10^{-6} bits/sec^2$, $C = 1\ mbps$, and $\Delta = 0.4$. Re-run the program for different parameter values, and plot the results. Your report should also explain how the graphs showing your computed results can be scientifically rationalized.

[5]At run-time, a packetization algorithm would dynamically choose a packet size R such that: $1 \ll R \leq R_m$, where R is determined from upper-layer considerations and other engineering factors. An actual design choice of R may be based on, for instance, how fast/slow the source-level data generation process is and how much communication parallelism is desired among the store-and-forward activities in packet-switches.

A historical perspective of how packetization concept evolved

[Packetization as an Enabling Driver for Systems Engineering & Business Economics of Data Communication Networks]

The idea of packetization of an application data unit (ADU) — or, message — dates back to the early 70's when communication links offered low capacity transmission rates in the range of 2.4-9.6 *kbps* (often, over phone lines), the link error rates were high, and the communication-level processing speed of data forwarding switches was slow. For e.g., a 500-byte packet transmission on a 9.6 *kbps* link would take about 400 *msec*, and the transfer of a 100-*kbyte* file would then involve sending 200 packets one after the other. Typical transmissions were in the range of 100-512 byte long packets. The need for such a packetization arose from two considerations:

1. Breaking up an ADU to small-sized packets reduced the amount of retransmissions caused by link errors;
2. Tapping of parallelism between the packet transmissions over a multihop path and the packet processing at intermediate switches.

Despite technology changes in the last 50 years where link speeds and packet switch processing speeds increased million-fold and the link error rates decreased 1000-fold, the idea of packetization is still very foundational to the design and management of Computer Networks. Today, the ADUs be very large data-units: such as cellphone pictures, video frames, terrain images, visualization data, multi-player game & VR data, and the like. These data units are nevertheless still packetized for transmission over computer communication networks: say, the Internet.

The idea of packetization has evolved from one of being a necessity to support digital data communications in the mid-60's to the early 90's to one of being an enabler today for complex network engineering and revenue-oriented business models. Variable-sized packets are extensively employed in the packetization layer of data networks even today (though the issue of clock-drifts has been overcome by the improved technologies). For instance, statistical multiplexing of packets is an engineering mechanism that allows the sharing of network bandwidth among multiple applications without wastage. And, this mechanism is widely employed by ISP's (e.g., Verizon, Spectrum) who manage a large bandwidth slice (leased from a carrier provider — e.g., Sprint) as a commodity shared among multiple retail customers[6].

That data communication industries still use packetization as a core concept is partly driven by these considerations. Even from a systems engineering standpoint, making a technology transition today from packet-switching (back to message-switching) is very hard — because of the already-deployed millions of packet routers and switching elements in the core Internet (since the 80's and 90's) that is the backbone and life-line of our society today[7].

---

[6] The principles of cloud-computing which took root in the 2000's based on "virtual machine" (VM) slices are also analogous to the packet switching concepts in data communication networks. Here, a large physical machine capacity is sliced to into smaller VMs for allocation to customer applications (on a revenue-basis). An allocation of VMs on-demand requires the ability to slice variable-sized VMs.

[7] An analogy is the manufacture of small and big cars, as seen from the industry standpoint during the 1950's and in our present times. Consider, for e.g., a house-hold with 6 family members deciding to have a large car with enough seating capacity or have three small cars. In the 50's, a large car was deemed as the only possibility due to various factors (such as manufacturing capability, parking spaces, gas consumption, etc). Though these problems have become trivial today, the small cars still dominate the auto industry today. Besides the higher quality of driving & comfort attainable with smaller cars, their market dominance is governed by the road transportation logistics and the industry's business models.