CALCnet$^{2.2}$: A Robust, High-Speed Network Protocol for Two-Wire Devices

Christopher Mitchell

Abstract

While devices and machines with clock cycles to spare have evolved to harness increasingly-complex network protocols, especially the widespread IEEE 802.3 (Ethernet) and IEEE 802.11 (WiFi) families of specifications, less-powerful devices have historically engaged in only limited two-unit communication. A particularly unfortunate example is TI graphing calculators. This ubiquitous, relatively-inexpensive computing platform is owned by millions of high-school and college-aged students around the world, but only natively supports a two-unit transfer mechanism. Built around a z80 microprocessor executing at 6MHz or 15MHz, it is incapable of participating in modern networks, but has sufficient computation power to support a robust network tailored to its strengths and weaknesses. CALCnet$^{2.2}$ presents such a protocol, boasting point-to-point and point-to-multipoint communication, asynchronous tranceptions, failure resistance, robustness against noisy channels, and many other features of a reliable and extensible network protocol. The implementation is small, fitting into under 1KB of z80 assembly code, and is shown to perform with superior reliability and at comparable speeds than even existing two-unit transfer protocols.

1 Introduction

CALCnet$^{2.2}$ offers a robust, asynchronous end-to-end transport protocol capable of dynamic, stateless network topology reconfiguration, error detection and recovery, and full network features such as arbitrary simultaneous disjoint communication. It supports one-to-one directed transmissions and one-to-many broadcasts to support as wide a set of applications as possible, and is targeted at utilities, chat programs, file-transfer programs, conference applications, and especially multiplayer and massively-multiplayer gaming on TI graphing calculators. As of this publication, the CALCnet$^{2.2}$ protocol has been written into Doors CS 7.1, a shell and GUI developed for calculators by the author. It can be used on TI-83+, TI-83+ Silver Edition, TI-84+, and TI-84+ Silver Edition graphing calculators, as well as via the TI-84+ emulator on the TI-nspire calculator. This document presents first the motivation and existing alternatives, then the frame-level, byte-level, and bit-level protocol as well as the steps and caveats in implementing CALCnet in new programs, and finishes with a survey of performance, branding, and future work. The latest information on CALCnet and Doors CS can be found at http://www.cemetech.net; Doors CS-specific downloads and information, including its SDK, are at http://dcs.cemetech.net.

2 Motivation and Existing Alternatives

CALCnet$^{2.2}$ was motivated by the author's desire to make calculators do more. The hardware is relatively inexpensive, and due to its requirement by the vast majority of modern secondary math and science education programs, very widespread. TI graphing calculators are reasonably

powerful, with a microprocessor, memory, LCD, and serial communication port all sufficient for high-functioning software, which have been used to make a wide variety of programs and games over the years. Although hardware modications and peripherals for the devices have also been explored, much less development effort has been devoted to this area because of the additional skill, cost, and risk associated with hardware additions and modifications.

CALCnet$^{2.2}$ is the culmination of close to eight years of brainstorming and effort devoted to developing a feasible, fast, small, and inexpensive solution for linking multiple calculators together. The operating system (the TI-OS) that ships with each TI-83+, TI-83+ Silver Edition, TI-84+, and TI-84+ Silver Edition contains a unit-to-unit linking protocol that allows programs and other variables to be send between two devices directly connected by a two-wire serial link cable. However, the official protocol does not include any provisions for linking more than two devices simultaneously[4]. In addition, the protocol is poorly-suited for gaming and other high-performance applications. Two community alternatives have been created, TachyonLink by Michael Vincent[8] and BELL by Tim ”Timendus” Franssen[2]. A third alternative called CLAP, also by Franssen[3], is the only other well-known protocol to offer multi-calculator linking. Finally, two I^{2}C implementations for calculators, one my Mr. Vincent[7] and one by O. Suominen and V. Crabb[6], can be used for multi-calculator linking but have not been widely disseminated.

2.1 Existing Link Protocols

The TI-OS linking protocol can only reliably allow two calculators (or a calculator and a peripheral or host device) to be connected[4]. Its speeds are disputed, reported by one source as 45 to 50 kilobits per second (kbps), and by another as around 15kbps[5]. A third reputable source estimates the TI-OS linking speed at about 8kbps or 1KBps[1]. It is severely limited for authors of third-party software who wish to harness communications between multiple calculators, and to this end, three community projects have been published before the completion of CALCnet$^{2.2}$. Michael Vincent’s TachyonLink is targeted at game development, and offers a synchronous transmission protocol that is reported to achieve transfer rates up to 4.3 kilobytes per seconds (4.3KBps or 35.4kbps)[8]. As of this publication, no applications or programs can be found that make use of TachyonLink for performance comparison. A similar protocol called Binary data Exchange Link Library, or BELL, was coded by Timendus, né Tim Franssen. It offers similar operation to TachyonLink, including synchronous (blocking) operation and transmission of multi-byte chunks of data. Its documentation reports top speeds of around 10.66kbps or 1.33KBps, but does not provide any data integrity or parity checking[2].

Two implementations of the Inter-Integrated Circuit protocol, commonly abbreviated I^{2}C, have been written for TI graphing calculators. The first, created for the TI-85 and ported to the TI-83, was created by Messrs. Suominen and Crabb[6]. It implements the basic I^{2}C protocol as well as a networking layer called ”MBus” on top to facilitate use for calculator networking. MBus uses I^{2}C addresses, which are limited to an 8-bit or 10-bit address space, and must be negotiated at connection time by each calculator by picking a random address, pinging it, and then claiming that address if no response is received. I^{2}C has a special broadcast address that can be used to send to all devices on the bus at once, although MBus does not implement broadcast compatibility[6]. The MBus protocol’s frames are surprisingly similar to CALCnet$^{2.2}$ ’s, with the exception that it defines a ”protocol” byte in each frame falling into categories like games, chat programs, file transfer programs, network utilities, raw data, and testing applications. Each MBus program is designed to use a unique protocol byte, theoretically limiting the protocol to 180 unique programs or applications that utilize its capabilities[6]. The end of the MBus readme and documentation makes a surprising legal point about the use of I^{2}C: it may be contrary to the original Philips I^{2}C license to operate an I^{2}C bus without any Philips integrated circuits present on the bus. The second I^{2}C implementation was written by Mr. Vincent for his spectrum analyzer

project[7]. No in-depth documentation of his libraries are available, but they are presumed to provide basic I^2C functionality without an additional calculator-specific layer.

The only alternative networking protocol to CALCnet$^{2.2}$ currently available not built on I^2C is Mr. Franssen's Calculator Linking Alternative Protocol, or CLAP[3]. The protocol offers networks as small as two and as large as 255 calculators. Each network is required to have a calculator that acts as a master, assigning addresses to other calculators joining the network, while all other calculators act as slaves. It contains provisions for assigning a new master if the current master leaves a network. No information has been published on its performance or throughput. It is believed that CALCnet$^{2.2}$ represents a significant advancement in the state of the art of calculator networking. It offers the following improvements:

- A decentralized, true peered network in which each calculator is responsible only for its own communication. No calculator need hold an exhaustive table or register of network members outside of its application's needs.
- A reliable communication model, ensuring accurate transmission even over an inhospitable or noisy network, at the bit level, the byte level, and the frame level. Collision detection, backoff, and other techniques from existing network architectures are used to effect failure detection and recovery.
- An arbitrarily-large address space sufficient to simultaneously encompass all currently-manufactured TI-83+ through TI-84+ Silver Edition graphing calculators, but contributing no statistically-significant reduction in network burst capacity.
- The ability to conduct one-to-one directed transmissions or one-to-many broadcasts within the same network framework.
- Asynchronous transmission and reception of data requiring no intervention from the host user program, allowing the program to read and write the send and receive buffers at its convenience.

2.2 Electronic System

To understand the protocol that CALCnet$^{2.2}$ represents, it is necessary to understand that electronic systems available for inter-calculator communication. Every z80 graphing calculator has a two-wire link port, that is, two signal lines and a ground. These two lines may each represent a binary bit, a logical one represented by electrical ground or 0 volts and a logical zero represented by 5 volts, as per convention. Each line floats at 5v and can be pulled low by any calculator. If one calculator pulls a line low, it is pulled low for all calculators, and only when all calculators have released that line does it float high on every calculator. These two lines can be respectively termed *tip* and *ring* from their positions on a 2.5mm stereo phono plug, but in CALCnet are called the *clock* and *data* lines. In the canonical CALCnet$^{2.2}$ implementation, the data line of every calculator is connected to every other calculators' data line, all the clock lines are tied together, and all calculators share a common ground.

Therefore, no special hardware is required in order to set up a CALCnet network. Future designs for network extenders or boosters, CALCnet$^{2.2}$ over Internet (globalCALCnet) adapters, and wireless modules may be constructed, but the simplest type of CALCnet network involves simply splicing link cords together. Figure 1 shows a seven-calculator network with six of the calculators connected. The circuitry on the breadboard is partly to support LED network activity indicator and partly a microcontroller board to trace and debug the network, but none of that peripheral circuitry is necessary for CALCnet$^{2.2}$ itself to function.

3 Protocol Details

Every byte of data carried across a CALCnet$^{2.2}$ network is encapsulated in a frame containing the IDs of the sender and receiver, the length of the data, and a checksum to guard against corrupt

Fig. 1. A seven calculator setup; six of the calculators have been connected to the network.

transmissions. CALCnet is designed to allow transmission over noisy or lossy channels, although for simplicity it performs minimal error correction in favor of retransmission. The following sections describe CALCnet$^{2.2}$ from the top down, starting with the structure of each frame, narrowing to the structure of each byte sent, and finishing with the timing and functionality of transmitting a single byte.

Note that as mentioned previously, the TI link protocol follows an active-low model. Normally, the data and clock lines (tip and ring) both float at electrical high, +5V, or logical false or zero. When a calculator "pulls down" a line to 0V, it represents a logical true or 1. Electrically, when one calculator pulls a line low, all other calculators' congruent lines are pulled similarly low, but another calculator releasing the line will not return it to electrical high / logical low unless all other calculators release that line.

3.1 Frame Structure

Every CALCnet frame is a series of 1 to 255 payload bytes sent from one calculator to another calculator or broadcast from one calculator to every other calculator on the network. A schematic view of the frame transmission protocol is provided in Figure 2. Every CALCnet$^{2.2}$ frame begins with a period of 5,000 to 15,000 6MHz cycles, or 0.8ms to 2.5ms, in which it waits to see if anything else is sending. It starts counting up from zero to 5,000 cycles; every time it sees activity, it resets that counter. If 15,000 cycles have passed and the counter has not yet reached 5,000, then it assumes that the network is busy, and waits until the next time the interrupt is triggered to try sending. At any point before 15,000 cycles have passed, when 5,000 idle cycles are seen, the protocol transitions from listening phase (1) to jamming phase (2). In phase (2), the calculator holds the clock line of the network low at electrical ground / logical high, serving two purposes. First, it prevents any other calculator from beginning a transmission. Secondly, the sum of the time spent in (1) and (2) is the same as the interval between interrupts triggered 110 times a second (110Hz), so even for calculators with staggered interrupt timing, every calculator on the network will notice the jamming and get ready to receive data. In the ideal case, after (2) completes after a total of at least 9.8ms of (1)+(2), every other calculator in the network is waiting for the clock or data line to start to fluctuate, indicating the beginning of the frame's

real contents.

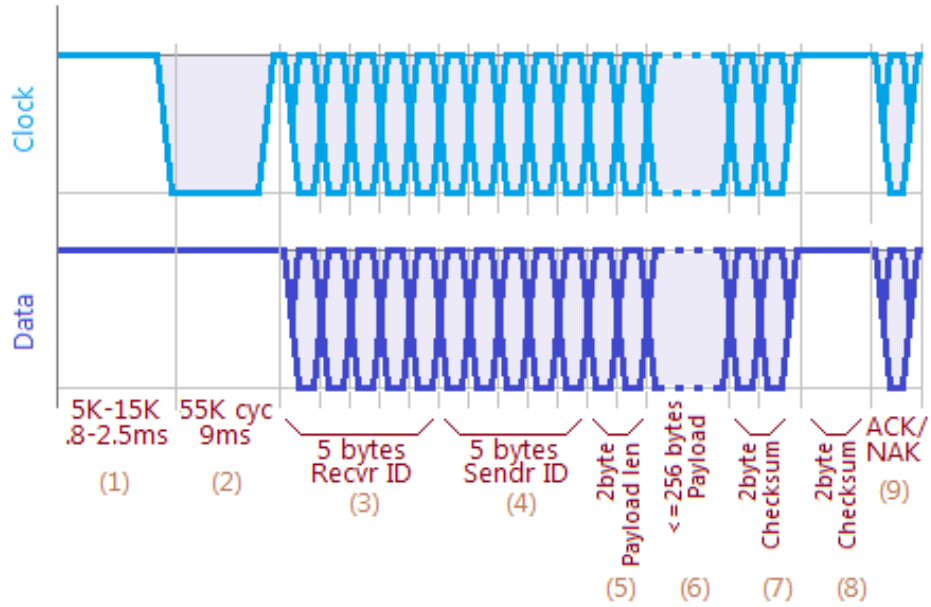


Fig. 2. Frame-level transmission protocol, including network acquisition guard, transmission of metadata and payload, and reception and confirmation of the checksum. (1) monitor the network for other transmissions to ensure that the network is idle; (2) hold the clock line low for the length of one 110Hz interrupt to acquire the network; (3) transmit the receiver's ID, or `000000` for a broadcast; (4) transmit the sender's ID; (5) transmit the data length; (6)(7) transmit the payload and its 2-byte checksum; (8) wait for a checksum from the receiver; (9) ack or nak the received checksum.

First, five bytes are send, containing the recipient calculator's ID. There are two special values: `AAAAAA` represents the gCn bridge on the network, if present, while `000000` denotes a broadcast transmission that should be received by all calculators on the network. The recipient ID is placed first in a frame so that every calculator can immediately stop receiving and resume normal execution if the frame is not intended for them. Next are five bytes of transmitter ID; the only special value is `AAAAAA` for a transmission from a gCn bridge. Note that the special value of `AAAAAA` is merely a convention for ease of finding the bridge, and may be changed or supplemented in real-world CALCnet$^{2.2}$ usage. Next is a two-byte payload length field. Unfortunately, because of buffer size, CALCnet$^{2.2}$ should be restricted to a maximum payload length of 255 bytes, but future versions may allow larger frames. Next are the actual payload bytes, between 1 and 255, in stage (6). Stage (6) is a simple two-byte modulus checksum of all the bytes in the payload only. The receiver should be summing the incoming bytes in parallel with their reception so that when the last byte is received from the transmitter, it can immediately send back the checksum it calculated. The transmitting calculator then ACKs the checksum with the one-byte value `$AA`, or may choose to either NAK with a different value or immediately stop transmitting, triggering a timeout on the receiver. Any NAK should invalidate the frame.

Note that broadcast frames do not follow the checksum/checksum/ACK-NAK convention. After the transmitter sends its checksum, the frame is complete, and receivers are solely responsible for checking the checksum and validating or invalidating the received data. A valid set of data received is indicated in the buffer with the high bit of the most significant byte of the data length set. If that bit is not set, there is no valid data available. Similarly, a calculator may only begin receiving if that bit is reset, so after a calculator program reads data out of the buffer, it

is responsible for resetting this bit so that further data may be received.

3.2 Byte-Level Protocol

The byte-level protocol is shown in Figure 3. Every byte starts with a 312-cycle stage (1) (still 6 MHz cycles, so 52μs) in which the calculator pauses to allow any receiving calculator to catch up, add the previously-received byte to the stored checksum, and perform other quick management tasks. It then listens to the network for an additional 660 cycles, or 110μs, to ensure that no collision occur. If any link activity is seen during this period, as indicated by the data or clock line transitioning to logical high / electrical low, a collision is assumed to have occurred, and the frame is immediately aborted. The calculator will then proceed to jam the network for 3,780 cycles, or 630μs, to ensure that all calculators see the collision condition, whether receiving or transmitting, and cease activity so that the network can recover. If no collisions are seen in stage (2), the transmitting calculator then holds both the clock and data lines low (at electrical ground / logical high) in stage (3) for another 52μs to signal the start of this byte. It follows with another 43μs period with both lines released, stage (4), to guarantee sychronicity, then transmits eight bytes in quick succession as per the bit-level protocol. An important note: although to function correctly, the routine that CALCnet[2.2] implements to transmit reverses the byte to be sent, the most significant bit (MSB) of the byte is sent first, and the LSB last. No ending guard is provided to mark the end of the byte, as the provisions of the bit-level protocol are sufficient. As explained in the following section, bits to be transmitted are inverted before being sent, so a 1 bit is sent as electrical high, by releasing the data line, and a 0 bit is sent as an electrical low, by pulling the data line low.

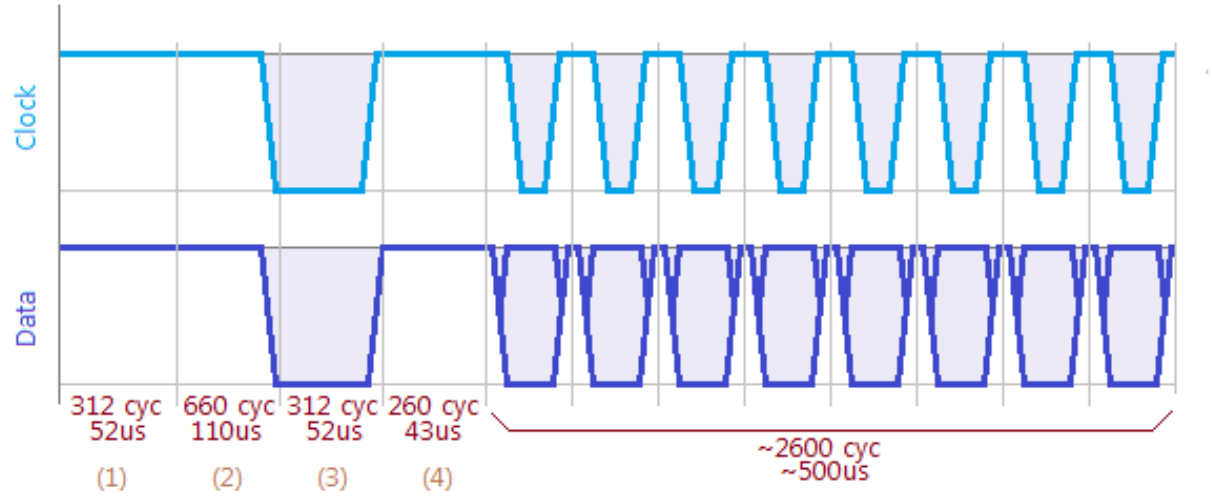


Fig. 3. Byte-level transmission protocol, including (1) and (2) a time to synchronize and allow for client computation; (3) a 52μs guard marking the beginning of the byte; (4) a second 43μs guard for synchronization; and (5) the 8 bits of the byte.

3.3 Bit-Level Protocol

The bit-level protocol is fairly straightforward compared with the byte-level protocol, requiring three stages totaling 104μs; it is shown in Figure 4. This design could yield a maximum raw throughput of 9.6kbps, but this of course is infeasible given all the synchronization, error-proofing, and checksumming that must be performed. First, the data line is either released (for

a 1 bit) or pulled low (for a 0 bit), based on the value of the bit to be transmitted. 17μs later, the clock line is pulled low. As soon as the clock line pulled low, each receiver should immediately read the data line and store the bit seen. After another 35μs, the clock and data lines are both released, and a 52μs pause, denoted stage (3) above, is added to allow the receivers to rotate the byte being received and loop to accept another bit.

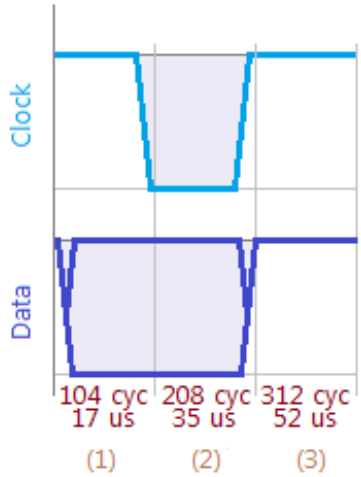


Fig. 4. Bit-level transmission protocol, including (1) a period with the data line set with the bit to transmit and the clock released; (2) a period with the data line set and the clock pulled low; and (3) a period with both the clock and data line released.

4 Using CALCnet$^{2.2}$

CALCnet$^{2.2}$ is designed to be relatively easy for programmers to use with minimal complications and training, and to that end exposes only four functions to assembly programmers, two of which need not be used in general programs. The four functions are as follows:

- **Cn2_Setup**: Initialize the CALCnet$^{2.2}$ system, including starting the interrupt and setting up the memory areas and buffers to be used by CALCnet . Once this routine has been called, 547 bytes starting as SavesScreen (`$86EC`) will be used by CALCnet and should not be used other than the methods to clear buffers manually or automatically as outlined below. In addition, the RAM segment of the CALCnet$^{2.2}$ interrupt is stored in 42 bytes starting at `$9999` and ending at `$99C3`, and the jump table uses 257 bytes from `$9A00` to `$9B01`.
- **Cn2_Setdown**: Disabled the CALCnet$^{2.2}$ interrupt, after which the CALCnet buffers can be used again as normal safeRAM.
- **Cn2_ClearRecBuf**: Clears the receive buffer, a total of $256+5+2 = 263$ bytes. In practice, the receive buffer may be conceptually cleared as far as the interrupt is concerned simply by resetting the MSB of the size field in the buffer, after which you may receive a new frame.
- **Cn2_ClearSendBuf**: Clears the send buffer, a total of $256 + 5 + 2 = 263$ bytes. In practice, the send buffer may be conceptually cleared by the interrupt by it resetting the MSB of the size field in the buffer, after which you may load in a new frame.

There is one additional function that CALCnet$^{2.2}$ programs should use; because the TI-OS interrupt that handles GetCSC is disabled while CALCnet is active, programmers should use `call Cn2_GetK` instead. It will debounce the keys entered, so for non-debounced, repeating keypresses like movement in a game, direct input should be used. Programs should also be careful of TI-OS ROM calls (`bcalls`) that disable or enable interrupts, such as `_RunIndicOff`

and `_RunIndicOn`, and use alternatives for such routines. The CALCnet$^{2.2}$ interrupt will handle turning the calculator off and on via the [ON] key, so user programs need not provide this functionality. Finally, be aware that during the CALCnet interrupt every calculator, regardless of type, will be running in 6MHz mode. Programs that require 15MHz mode are recommended not to simultaneously use 15MHz mode and CALCnet$^{2.2}$: although the interrupt will properly handle transitioning to 6MHz mode for its duration and returning to 15MHz mode before the user program resumes execution, 15MHz mode will cause the CALCnet interrupt to trigger at 275Hz instead of 110Hz, thus spending proportionally more time in the interrupt than in the user program, and possibly nullifying the objective of 15MHz mode (ie, faster execution of the user program).

As mentioned previously, CALCnet$^{2.2}$ relies on the use of buffers to send and receive data. Programs need not call any routines in order to initiate transmission or reception of data, allowing for asynchronous, nonblocking communication. This means that a program can leave data to be sent over the network in a buffer and continue to execute; CALCnet will handle attempting to send the specified data until it succeeds. Similarly, the interrupt listens constantly (technically, 110 times per second) for data to be sent to the calculator, and will store any such data in a buffer until fetched by the userland program. Needless to say, it is important to periodically check for received data and remove it so that CALCnet can accept another frame. The memory areas used by CALCnet are enumerated below in Table 1, and are further discussed in Sections 4.2 and 4.3.

CALCnet$^{2.2}$ Buffer Usage			
Address	Offset	Size	Function
`$86EC`	0	2	*not for userland use*
`$86EE`	2	5	Current calc's ID (do not modify)
`$86F3`	7	5	Receive buffer sender ID
`$86F8`	12	2	Receive buffer size word
`$86FA`	14	256	Receive buffer
`$87FA`	270	5	Send buffer receiver ID
`$87FF`	275	2	Send buffer size word
`$8801`	277	256	Send buffer
`$8901`	533	14	*not for userland use*

TABLE 1
CALCnet$^{2.2}$ uses a 547-byte chunk of memory for its temporary storage and buffers. User programs should interact with the receive buffer and send buffer via direct memory access.

It has been shown that these sets of functions and features are sufficient to create a large set of network-aware games and applications. A screenshot of a sample application that has been released called `NetPong v1.0` can be seen below in Figure 5.

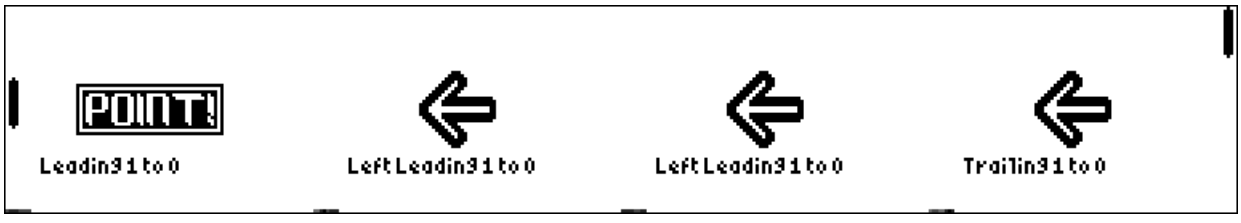


Fig. 5. `NetPong` game demonstrating four-calculator gaming over CALCnet$^{2.2}$.

It has been found that there are several TI-OS related caveats of which to be aware when using CALCnet$^{2.2}$. All such caveats are due to the interactions between the TI-OS and user interrupts

(Interrupt Mode 2 ISRs, such as CALCnet). It has been found that certain TI-OS bcalls, especially those that potentially interact with the LCD, maybe improperly disable interrupts including the CALCnet ISR. Known culprits including `_vputs` and `_vputmap`, but other ROM calls are likely to cause problems. It is strongly recommended that programs that using the CALCnet[2.2] ISR re-enable interrupts via `ei` after one or more of such calls. If there are sections of your code that may or may not have interrupts enabled, the following construction may be used (thanks to Brenden "Calc84Maniac" Fletcher for his assistance):

```
1         ;begin protected section
2         xor a
3         push af
4                 pop af
5         ld a,i
6         jp pe,sectionbody
7         dec sp
8         dec sp
9         pop af
10        add a,a
11        jr z,sectionbody
12        xor a
13 sectionbody:
14        push af
15                ;di ;uncomment if interrupts should be disabled here
16                ;------code that may have disable interrupts or-------
17                ;------need interrupts disabled goes here-------------
18                pop af
19        jp po,sectionend
20        ei
21 sectionend:
```

4.1 Finding Network Members

CALCnet[2.2] does not contain an explicit method of finding network members; the send and receive routines were initially designed to be purely point-to-point. Programs that already know the 5-byte address of the receiving calculator can simply fill in that address in a frame to be sent, and the data will arrive at the receiver. However, for dynamic networks, it will be necessary for a program to find other calculators in the network. In order to do so, each calculator can be set to periodically broadcast its address to other calculators during a discovery phase or even as a packet type during normal operation. Because CALCnet[2.2] only defines at most the bottom three layers of the OSI model (physical protocol, the data layer, and the physical layer), it is up to each program to define the meaning of the contents of the data section of frames. However, if a program puts a frame in its calculator's CALCnet send buffer with a 5-byte receiver address of `000000`, CALCnet will interpret that address as indicating a *b*roadcast frame, and will send that frame to every calculator on the network. Note that broadcast transmission is somewhat less reliable than point-to-point transmission. Whereas point-to-point guarantees that once the data disappears from the sender, the receiver has successfully received and acknowledged the data, successful transmission of a broadcast frame does *not* guarantee that every or even any calculator has received the data. Repeated broadcast transmission of the broadcast packet should offer a reasonable chance that each calculator will receive the packet, but the broadcast should not be continually attempted to avoid monopolizing the network.

4.2 Sending

To send data via CALCnet[2.2] , a calculator program must first place the receiver ID, the frame contents, and the size of the frame to be transmitted in the associated memory areas, in that order (see Table 1). The order is important because CALCnet notes whether a frame is pending transmission by checking the most significant bit of the two-byte send buffer size word. Because the z80 follows the little-endian storage model (the less significant byte of a word is stored before the more significant byte), the most significant bit is bit 7 of the byte at `$8800` (see Table 1 if the derivation of this address is not clear). If that most significant bit is reset, the CALCnet[2.2] interrupt assumes that there is no frame pending transmission. Once the bit is set, CALCnet may begin transmitting the frame at any point thereafter. Due to the asynchronous nature of interrupts, the interrupt may fire at any time, so it is vital that the high bit of the size word must only be set when the low size byte, receiver ID, and frame data contents have already been written. To facilitate direct copying into the buffer from RAM or ROM via `ldir` or an equivalent, the size may be copied with the MSB unset, then the MSB of the size word's high byte should be set. The biggest danger of prematurely setting the bit and thus prematurely indicating to the CALCnet interrupt that a frame is pending transmission is that a partially-complete frame may be transmitted. Testing and verification shows that this can happen frequently under real-world operation if the caveats dictated in this paragraph are not followed.

Once a completed frame has been placed in the send buffer, the interrupt may take a number of seconds to transmit the given frame in the range $[\epsilon, \infty)$, in practice roughly $[0.01, \infty)$ seconds. Transmission will take a finite amount of time if the receiving calculator is on the network and the network is sufficiently non-noisy and has low enough end-to-end latency to ensure proper transmission. A back-of-the-envelope calculation, taking the minimum granularity of network time as the unrealistically strict value $10\mu s$, dictates that a maximum transmission distance based on the speed of light is 29979 meters, or 30 kilometers or 18 miles. Therefore, any transmission over a wide-area network should be performed via an intermediary, such as the planned **g**lobalCALCnet (gCn) system. If the receiving calculator is not on the network, for example if the receiver's ID was incorrectly entered into the associated field in CALCnet send buffer, then the transmission will take an infinite amount of time, as the remote calculator will never acknowledge the packet. Transmitting calculators should provide a means of removing a frame from the CALCnet send buffer if it has been pending for a long time and has not been send successfully; it is up to the programmer to decide how to do this. One recommended method follows in the code listing below:

```
1     xor a
2     halt              ;halt to ensure that the store
3     ld (8800h),a      ;happens immediately after an interrupt
```

A program will almost definitely need to know when it is safe to load a new frame into the CALCnet send buffer. This can be done via the same method that the interrupt itself uses, ie, loading the byte at (8800h), masking off the most significant bit, and loading a new frame if and only if that bit is reset (zero). One note: the MSB is not considered part of the size, so that a size of `$8002` is considered a 2-byte frame, not a 32770-byte frame.

4.3 Receiving

Receiving a packet carries similar caveats and complexities as transmission. As with transmission, there is a single bit to read and write to indicate the presence or absence of pending data. As with transmission, there are three field in the relevant buffer: the two-byte size of the data (for which a value of `$8004` is considered a 4-byte frame, not a 32772-byte frame), the five-byte

ID of the sending calculator, and the 1- to 255-byte data section. As with transmission, broadcast frames must be handled, although from the receiver side, a broadcast frame is indistinguishable from a normal directed (point-to-point) frame. Finally, in symmetry to the transmission routine, the user program must clear the present-data bit when it has read the data out of the receive buffer in order to allow the CALCnet interrupt to accept another frame.

As with transmission, the memory areas relevant to receiving data and their locations and sizes are in Table 1. The receive buffer will contain a size word indicating the size of the data in the buffer; the data is only valid if the highest bit of the size word is set. CALCnet uses the receive buffer to store frames currently in transit to this calculator before each frame's checksum is verified, so only when the high bit of the size word is set should the data be read. Once the receive buffer is used, the CALCnet will ignore further frames transmitted to this calculator until the receive buffer is cleared, as indicated by the high bit of the size word being reset, so to maintain a low-latency network, the receive buffer should be checked and handled as often as possible.

5 PERFORMANCE

While the technical details and functionality of the CALCnet$^{2.2}$ protocol are important to its usage, an equally significant factor is its efficiency and speed. While CALCnet$^{2.2}$ does not surpass existing implementations in raw speed, it achieves the same order of magnitude transfer speeds in a network environment as existing protocols achieve for simple one-to-one transfers, while offering many more features and increased reliability[2], [8], [1]. CALCnet offers checksumming to ensure correct transmission, better synchronicity, collision detection, and related aspects vital to minimize transmission error and maximize throughput in a network.

Throughput was estimated by a program written to perform both unidirectional and bidirectional tests. In the unidirectional tests, one calculator takes the role of a permanent transmitter for a large stream of data, while another acts as the permanent receiver. Constant-size frames are transferred from the transmitter to the receiver, and are counted as received when the receiver finishes acknowledging a frame's checksum. In the bidirectional tests, two calculators take turns sending equal-sized frames. In such tests, each calculator sends a frame when it has finished receiving a frame from its partner. These tests are imperfect due to granularity and overhead, but give a good ballpark estimate of real-world throughput ceilings. A screenshot of the program can be seen in Figure 6 below, while a discussion of the results, including the maximum throughput figure of 3500 baud, are discussed in Section 5.1 below.

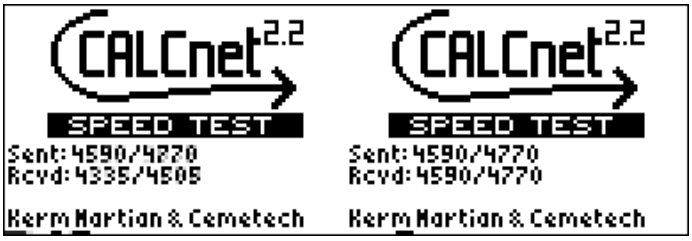
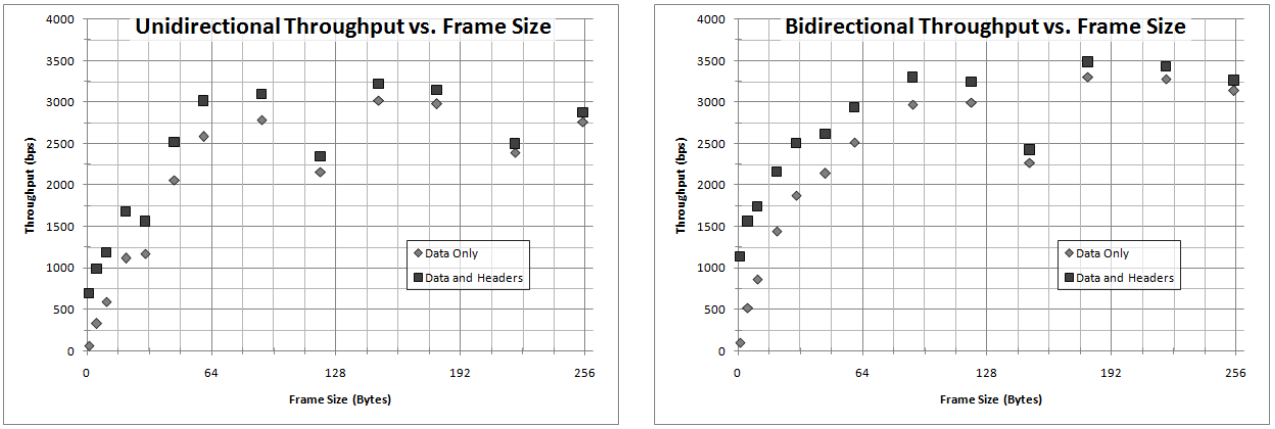


Fig. 6. `SpeedTst` program for performing unidirectional and bidirectional throughput tests.

Recovery from network faults, and robustness against benign or malicious network interruptions were also tested and found to be sufficient for real-world situations. While for obvious reasons CALCnet is not built to be backwards-compatible with existing point-to-point graphing calculator transfer protocols, it is sufficiently resilient to resume communication without ant data loss or permanent detriment following the repair of such conditions. Because it uses fixed address space, CALCnet$^{2.2}$ is impervious to problems of network segmentation to which the competing protocol is susceptible[3].

5.1 Network Throughput

In tests, maximum speeds of 3483 bits per second transmitted were achieved in bidirectional tests with a frame payload size of 180 bytes, as shown in Figure 5.1. On either side of this maximum, throughput decreased as frame size was lowered and raised, a result visible in Figure 7(b) and tabulated in Table 1(b). Note that because the testbed program, called `SpeedTst`, incurs non-negligible overhead while drawing the text for throughput statistics and copying the LCD buffer to the LCD, the lower end of the graph and table are artificially suppressed. A similar set of results can be seen for unidirectional testing, as tabulated in Table 1(a) and graphically displayed in Figure 7(a). The maximum throughput achieved was 3220 baud (bits per second) at a frame size of 150 payload bytes per frame, with a 180-byte frame a close second at 3141 baud. The estimated error bars on the results achieved are an estimated $\pm 5\%$, including human error in timing, insufficient test length, and granularity. The skew from display overhead cannot be accurately estimated.



(a) Unidirectional Throughput

(b) Bidirectional Throughput

Fig. 7. Experimental performance of CALCnet$^{2.2}$ under testing in a two-calculator network. Note that the low ends of both graphs are artificially low due to the overhead associated with rendering speed text and updating the LCD. (a) in a unidirectional testbed where one calculator sends a new frame each time the receiver acknowledges each frame; (b) in a bidirectional testbed where each calculator sends a new frame each time it receives one from its partner.

5.2 Collision and Fault Recovery

In tests, it was found that CALCnet$^{2.2}$ is qualitatively robust in recovering from collisions and interruptions, including intentional and malicious network sabotage. When segments of a network are made disjoint and later reconnected, transfers resume seamlessly. In the competing protocol, such disconnection would cause each subset of the network to reorder its address space, making recovery impossible[3]. It was also found that injection of suboptimal network conditions such as electrical noise, ground faults, and devices attempting to operate an incompatible protocol on a CALCnet$^{2.2}$ network were only sufficient to disrupt network activity for the duration of the injection, after which transmissions resumed with no data loss. The fault-tolerance and recovery and data integrity features of CALCnet were therefore found to be sufficient for real-world scenarios.

6 Branding

In order to distinguish CALCnet$^{2.2}$ hardware and software applications, including programs that take advantage of its routines and capabilities, the following set of logos and guidelines have

been created. It is requested that if in HTML, .doc, PDF, or other rich-text format, the manual or readme of such programs contains one of the three scales of CALCnet$^{2.2}$ logos from Figure 6. In addition, if possible, it would be appreciated if one of the oncalc logos in Figure 8(b) could be included in the binary of the program. In general, CALCnet$^{2.2}$ may be referred to as CALCnet in written documentation, as long as the first time it is mentioned it is named with the full string "CALCnet$^{2.2}$ ". Notice that the first four letters of CALCnet$^{2.2}$ and CALCnet are capitalized, while the remaining three are lowercase (a "backronym" has been derived from the name: Component for Asynchronous Linking of Calculators via a network).

7 Future Work

While CALCnet$^{2.2}$ is a complete and extremely well-tested and robust protocol, there is always room for further expansions and improvements. One possible improvement might be the addition of some packet recovery mechanism. Currently, invalid packets are simply discarded based on the checksum acknowledgment system. However, a simple Cyclic Redundancy Check (CRC)

(a)

Unidirectional Benchmarking							
Frame Bytes	Test Time (sec)	Bytes Data	Bytes Total	Bps Data	Bps Total	bps Data	bps Total
255	30.28	10455	10865	345	359	2762	2871
220	32.41	9680	10120	299	312	2389	2498
180	45	16740	17670	372	393	2976	3141
150	31.4	11850	12640	377	403	3019	3220
120	30.24	8160	8840	270	292	2159	2339
90	31.27	10890	12100	348	387	2786	3096
60	37	11940	13930	323	376	2582	3012
45	30.57	7875	9625	258	315	2061	2519
30	35.88	5250	7000	146	195	1171	1561
20	31.55	4400	6600	139	209	1116	1674
10	32.11	2370	4740	74	148	590	1181
5	53.51	2195	6585	41	123	328	984
1	32.73	257	2827	8	86	63	691

(b)

Bidirectional Benchmarking							
Frame Bytes	Test Time (sec)	Bytes Data	Bytes Total	Bps Data	Bps Total	bps Data	bps Total
255	31.21	12240	12720	392	408	3137	3260
220	32.76	13420	14030	410	428	3277	3426
180	31.86	13140	13870	412	435	3299	3483
150	37	10500	11200	284	303	2270	2422
120	30.45	11400	12350	374	406	2995	3245
90	51.68	19170	21300	371	412	2967	3297
60	36.68	11520	13440	314	366	2513	2931
45	95.02	25425	31075	268	327	2141	2616
30	40.04	9390	12520	235	313	1876	2501
20	30.32	5460	8190	180	270	1441	2161
10	30.53	3310	6620	108	217	867	1735
5	30.73	1995	5985	65	195	519	1558
1	41.72	539	5929	13	142	103	1137

TABLE 2
Detailed CALCnet$^{2.2}$ benchmarks for unidirectional and bidirectional transfers, as shown graphically in Figure 5.1. (a) Unidirectional tests, in which one calculator functions as the transmitter and another as the receiver; (b) Bidirectional tests, in which calculators take turns being the transmitter and receiver.

(a) Caption of subfigure 1

(b) Caption of subfigure 4

(c) Caption of subfigure 2

(d) Caption of subfigure 3

Fig. 8. Logos for use with CALCnet$^{2.2}$, including (a) (c) (d) computer-side logos for documents, HTML, and PDFs, and (b) oncalc logos and icons.

or other parity mechanism might provide the means to reconstruct damaged data and decrease the number of retries necessary for a successful transmission on a noisy or crowded network. Continued adjustment and fine-tuning of network timing might allow higher throughput with negligible increases in retries or failures.

Secondly, current frames are capped at 255 bytes of data payload, although the size field of frames is two bytes to allow for larger payloads. It is currently infeasible to provide larger frames due to the size of the buffer, but future expansions including allowing dynamic adjustment of the buffer address might remove this limitation. In addition, a possible streaming mode might allow additional functionality impossible with the block-based architecture of CALCnet$^{2.2}$.

Finally, though the protocol has been thoroughly tested on different combinations of TI-83+, TI-84+, and TI-84+SE graphing calculators, further testing under larger networks, both in terms of endpoints and physical end-to-end length, new hardware developments, and more demanding applications will allow the reliability and robustness of CALCnet to be further improved.

8 Conclusion

CALCnet$^{2.2}$ has been shown to be a powerful, fast, robust, and effective network protocol for TI graphing calculators in this paper. The frame-, byte-, and bit-level protocols have been presented along with the electrical specifications of a CALCnet network. Complete guidelines and documentation for implementing CALCnet$^{2.2}$ support in games and programs has been provided. Experimental results for performance, resilience, and throughput are presented and analyzed. The author hopes that with the availability of such a powerful and extensible protocol via the Doors CS 7.1 calculator shell, programmers will implement multi-user and multi-player systems in their applications and games going forward.

9 Acknowledgments

The author would like to thank the members of Cemetech for their feedback, advise, and stress-testing assistance over the past eight years. Thanks especially go to Shaun ”Merthsoft” McFall

for his moral support and hub-building experiences, Albert "AlbertHRocks" Huang for his similar efforts, Thomas "Elfprince13" Dickerson for encouragement over the years, and to the other members of Cemetech and the calculator programming and hacking community. Thanks to Chrystina Montuori Sorrentino for moral support, Billy Donahue, Timendus, and Michael Vincent for their algorithmic advice, and Prof. F. Fontaine and the μLab research laboratory of the Cooper Union for providing the computational resources.

REFERENCES

[1] P. Davidson, "The Ultimate TI Calculator FAQ - Transferring files," http://www.ocf.berkeley.edu/~pad/faq/xfer.html.
[2] T. Franssen, "BELL: Binary data Exchange Link Library," http://bell.timendus.com/.
[3] ——, "CLAP: The Calculator Link Alternative Protocol," http://clap.timendus.com/.
[4] R. Lievin, "TI Protocol Specifications," http://www.ticalc.org/archives/files/fileinfo/113/11382.html.
[5] B. Ryves, "Benchmark of TI-OS Link Protocol Speed," Internet Relay Chat conversation.
[6] O. Suominen and V. Crabb, "MBus v0.99: Multimaster network and I2C driver routines for TI-83," http://www.ticalc.org/archives/files/fileinfo/101/10122.html.
[7] M. Vincent, "michaelv.org - calculators - spectrum analyzer," http://www.michaelv.org/programs/calcs/sa.php.
[8] ——, "TachyonLink," http://www.ticalc.org/archives/files/fileinfo/277/27718.html.