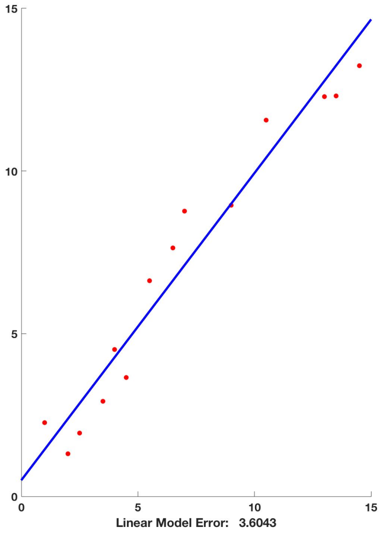
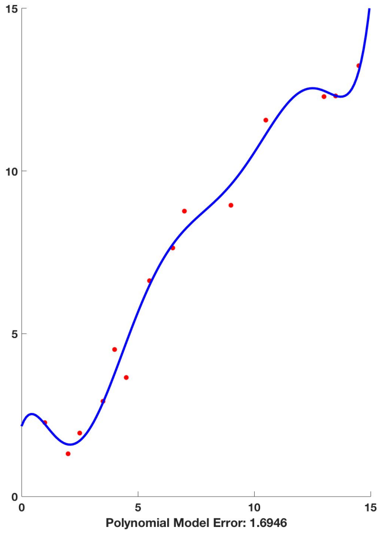
Computer Networks in the Control Loop

by William Wyatt Maleta

Finite versus Infinite

- When a computer network is in the loop, inputs are really data points sent at a rate
  - But the expected a smooth input with a smooth response in the form of continuous motion
- We expect the response for a given input to be :
  - Accurate
  - Timely
  - Predictable





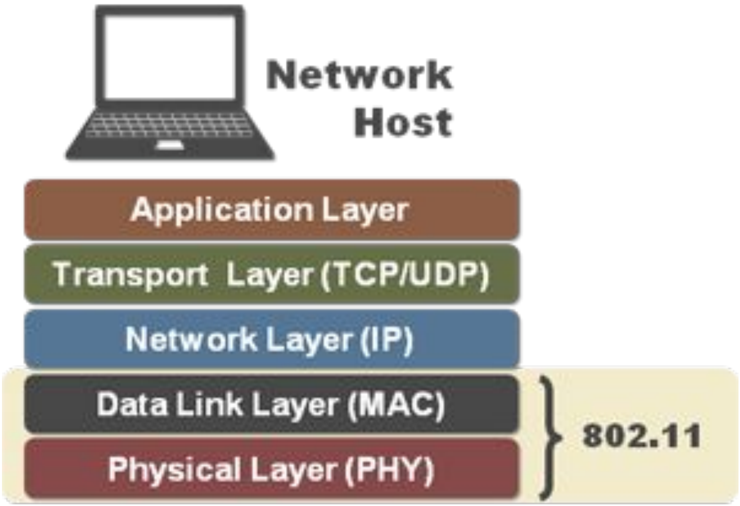
Example 1: Spot the Dog (possibly)

- The Network is most likely not the bottleneck for data through the control loop
- per Joystick input, there is a difference in position from the current position
    - Joystick does not determine absolute position
- For arm control or movement, user expects accurate and predictable control over a snappy response
    - Especially considering the MBs/sec data rate of a typical wifi connection

Example 2: Aircraft Control

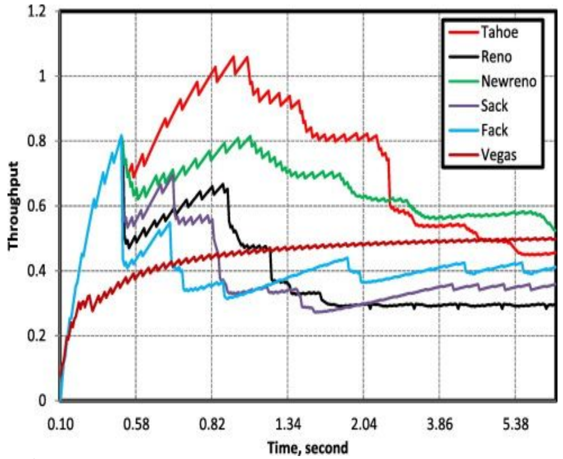
- In aircraft control fast and predictable response is critical.
  - being in an undesirable orientation for any small amounts of time can cause high stress on the aircraft
  - Can cause sudden failure in an extreme case
- It is not enough that control is snappy at high speed
  - Controller needs to be confident that an input will have the desired effect (with a small delay)

Network Stack to Model



- Wifi and IP are inevitable
- IP does not have a lot of logic executed compared to the transport layer
- Physical and Link layer more dominated by Hardware
- However, there is a choice in Transport and higher layers. Two common choices in Transport Layer:
  - UDP -> at most once
  - TCP -> at most once + at least once = exactly once

UDP and TCP



UDP:

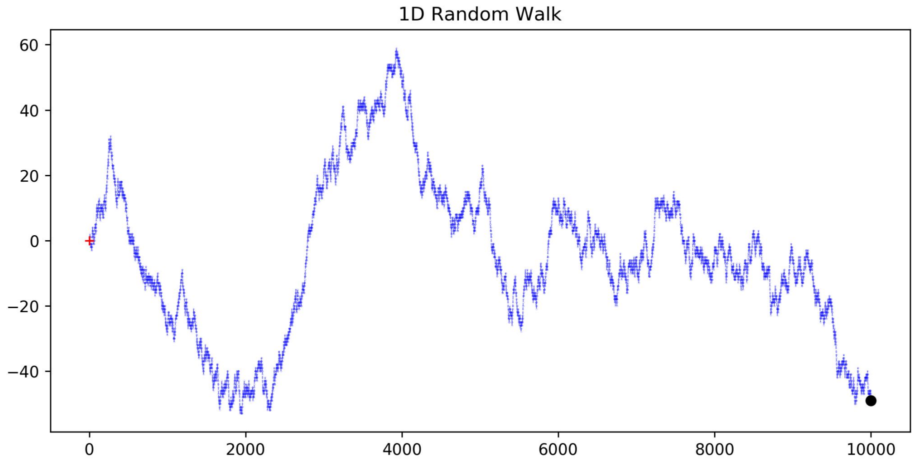
- It is a thin layer on top of IP
- Data sent at a static rate, sender does not know if the data arrived at the destination
- The latency from when a message was sent to the time it is processed is the time a message spends in internal buffers plus the time to physically transmit the data

TCP:

- For every packet transmitted, an acknowledgment is expected. Every packet is delivered to the best of the network's ability.
- The throughput is variable. Congestion Control algorithms are used to determine throughput:
  - For every packet that does need to be re-transmitted rate of transmission is increased by constant additive factor.
  - Rate decreased by constant multiplicative factor for each packet that does need re-transmission
- This creates a Saw Tooth throughput pattern

The Experiment

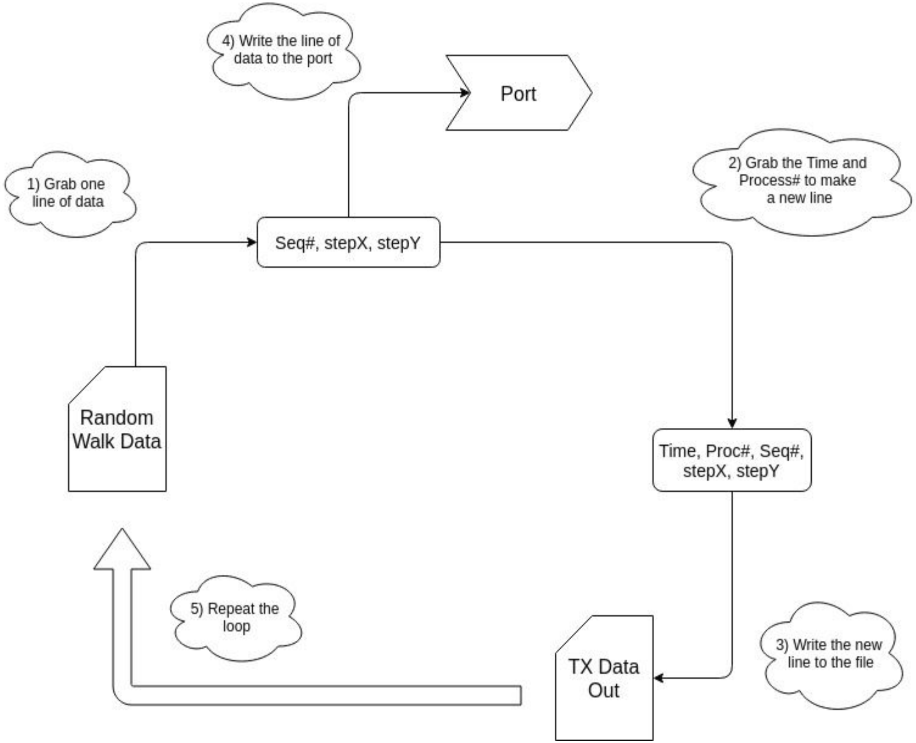
- Transmit a random walk from a transmitter (TX) to a receiver (RX).
- We transmit a 2D random walk (X, Y) but most graphs drawn with just the X coordinate
- Walk generated ahead of time in CSV where each line is (Seq #, stepX, stepY)
- stepX/stepY is -1, 0, or 1, indicating what the next step is
- Seq# is which number step this line is. We start with step 0, then 1, 2, … and so on.



- Random walk simulates input from user to a robot
- We expect that the delay between when TX sends a step and RX receives it, and the variability of this delay, will create a difference between where RX and TX think that the walk is at each point in time.
- UDP and TCP have different transmission characteristics, and we expect to be able to view this by comparing TX and RX walk data.

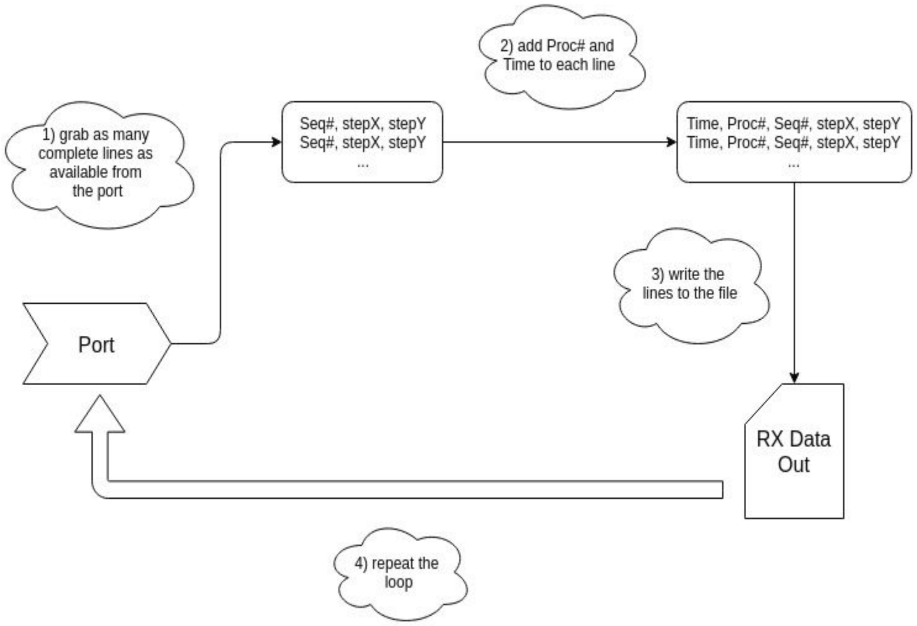
TX Block Diagram

- Random walk data is read one line at a time (one step at a time) from a data file.
- The step is time stamped and given a processing number, then written to an output file.
- The original random walk step is written to the port.
- Random walk data is sent one line at a time to try to encourage the OS to send several small packets to the RX process.
- This process is continued until the entire data file is transferred.



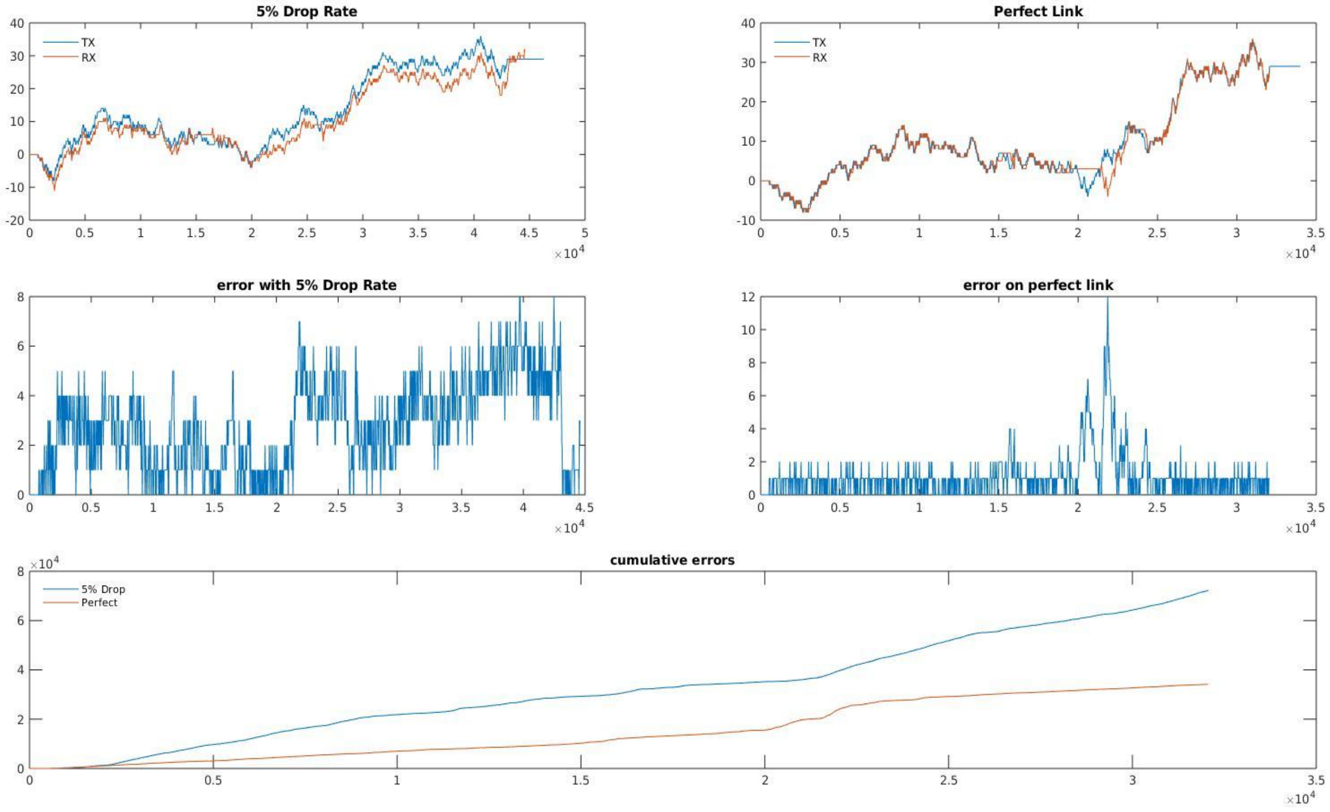
RX Block Diagram

- As many complete lines as are available are read from the port
- Each of these lines are given a process# and a time stamp
- Then all the newly formatted lines are written to the file.
- The loop is repeated until the entire data file is transfered.



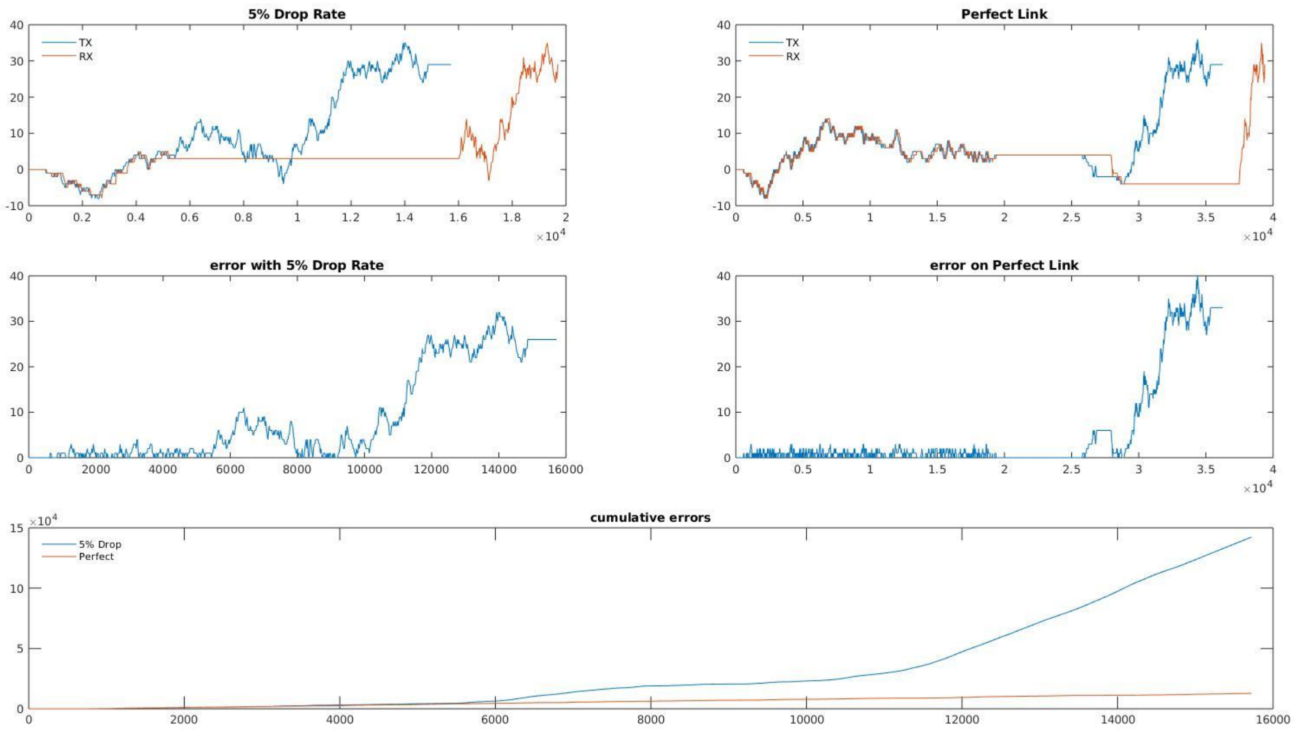
UDP Data

- Low error in both cases
- Predictable lag in output
- More linear cumulative error

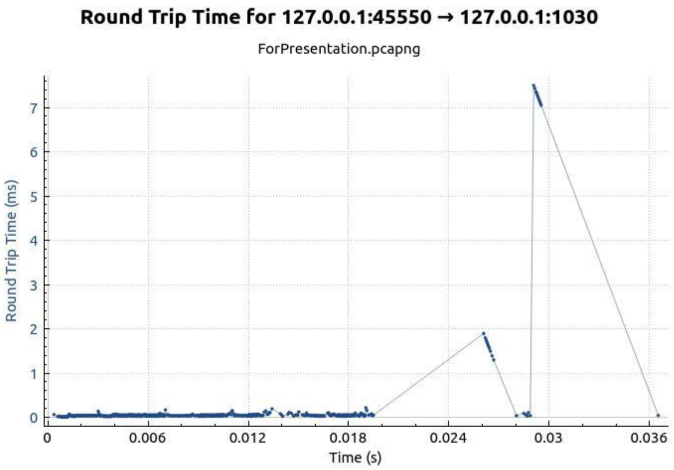


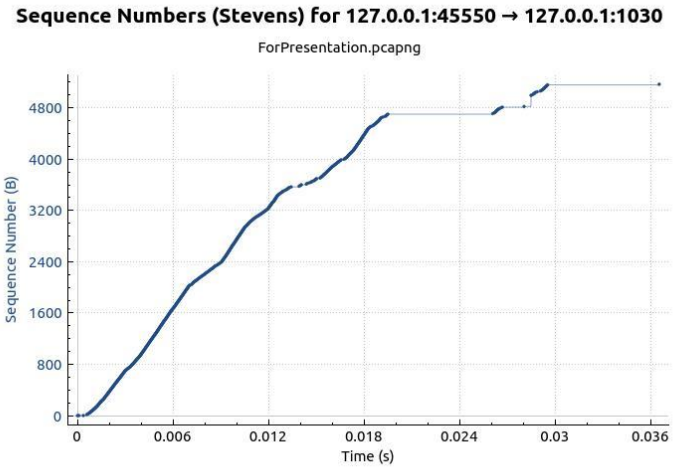
TCP Data

- Long pauses on perfect and imperfect link
- Jumps to new final position after pause
- Greater overall error
- Sudden slope increases in cumulative error



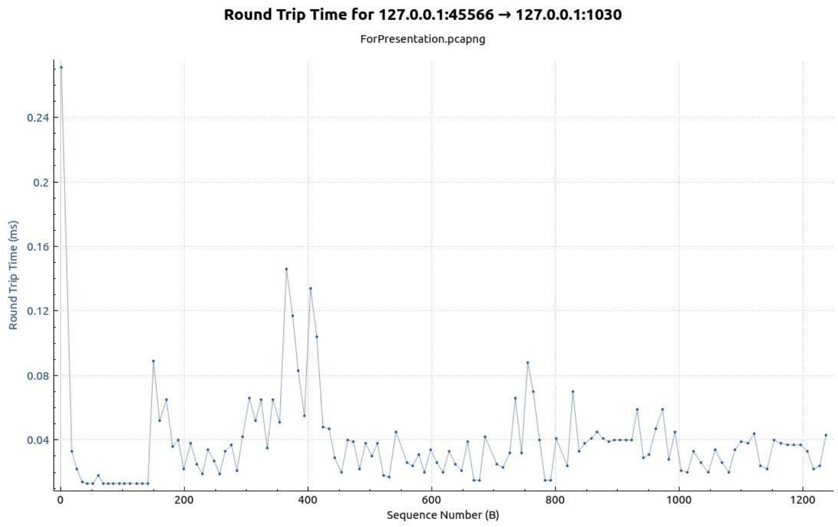
TCP on Wireshark (Perfect Link)

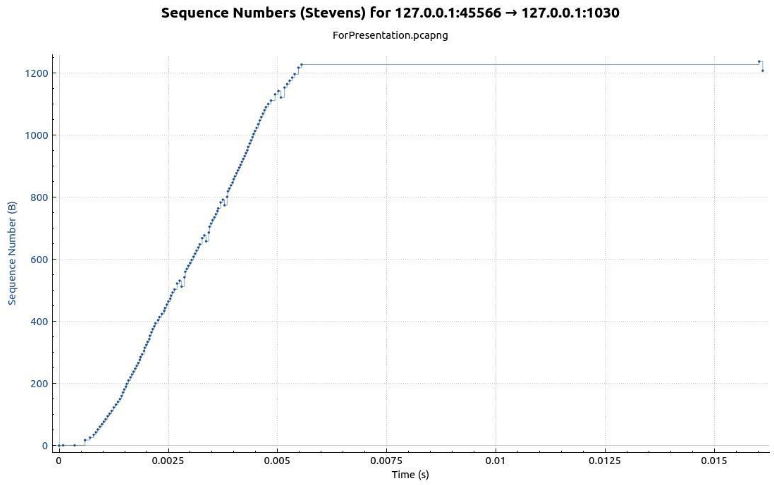




- The jump in RTT is from the RX not sending an ACK back to the TX.
- Application cannot read data until Kernel sends an ACK for the data the application wants to read.
- Most likely : Latency in the Kernel causes a slow down in the application.
    - This chain reaction causes the plateau in the random walk

TCP on Wireshark (5% drop rate)





- Spikes in RTT not necessarily the retransmissions
- The 4 dips in the Steven's graph are the 4 retransmissions
- It is hard to line up the error graphs with the Stevens graph, but:
  - Plateau is probably around first re-transmission
  - Probably caused by something between the application and the lowest levels of the network stack.

What does error mean?

- The definition of error in this context:
  - In this context error is just the difference between what the TX has recorded and the RX has recorded at each point in time
- The input is completely determined ahead of time
  - There is no communication between the TX and RX on what the input should be at any particular point in time
  - The TX strictly sends the data to the in a single stream to the RX (one way communication)
- The input is differential
  - In the sense that each input specifies in which way and how far to move from the current position
  - The position starts at 0
  - The position at any point in time is the summation of all previous inputs

Nagle's Algorithm

- Nagles is an algorithm executed by the TX:
  - The TX waits until there is a reasonable amount of data to send, then sends all the data in one big packet
  - Keeps down the number of packets to reduce data flowing through network
  - And reduce the number of packets to keep track of
- In high speed intermittent transfers this causes an unpredictable lag
- Without Nagles
  - floods the network with packets
  - TX is extremely opportunistic
  - Requires more processing power and memory
- Without Nagles, reduces some of the benefits of TCP

Memory Management and Kernel Operations

- High speed data transfer can be stunted by IO operations
    - Writing to disc will cause a sudden and dramatic pause
    - Application level dynamic data structures may cause unpredictable slowdowns if OS need to step in to manage the memory of these data structures
- Memory copying/moving is slow
- Overall, data structures are important and OS memory management is noticeable
    - Need to be careful how manage your memory
- Threads may help, but the transfer is a very serial task so would not expect a large speed up
- Core of the problem is that interfacing with a network is a lot of work on the part of a *kernel thread*
    - Cannot control these thread directly, and need to be careful on what you ask them to do

RST Flags, and a handshake

- Every TCP connection is started with a handshake to establish a connection
- Handshake is timely
    - Handshake must be completed below network bandwidth
    - Requires three data transfers over the network in series
- A restart flag causes a new handshake, AND a new socket to be allocated
    - This is double whammy:
        - Cannot use the old socket, Kernel must set up a new socket
        - Also need to repeat the handshake for the new socket
    - During tests a few spontaneous RSTs happened occasionally

TCP vs UDP

UDP

- Pro
  - No starting handshake
  - Data available to application soon after it is available to the Kernel
  - More predictable behavior in an unpredictable network.
- Con
  - Cumulative errors cause drift over time
  - Will always send at high speed
    - Does not respect quality of the network

TCP

- Pro
  - Will always eventually be correct (each message processed exactly once)
  - Congestion Control for appropriate data transfer rates
- Con
  - Rate that data is available to an application is more sensitive to network variability.
  - Delay between what is received by Kernel and what is available to the application
    - Can cause inputs to be processed , but very late

Error in the Context of Differential Control

- Tests only look at data transfer, but data transfer is one part of a control loop
    - Input is not completely mapped out beforehand
        - Input can be adapted to predictable error
    - A User can re-evaluate where they are according to where they want to be
    - Harder if cannot judge what result a particular input will have
        - Or if there are possibly a large number of inputs yet to be processed)
- Want environment (network quality) to be a smaller factor in the control loop
    - Don’t want network variability to influence control variability
    - Would rather predictable behavior in a range of conditions
- In the case of UDP and TCP:
    - Eventually getting the correct inputs to a robot will cause Jump and Jitter in control
        - I argue this is worse than a steady drift (increase in error) over time.

Summary

- High speed data transfer requires plenty of memory management and processing power
  - Can be a game of Buffers
    - There is the kernel buffers, and application buffers and how the two interact is critical
    - The network is one link in the chain
  - You are as fast as your slowest link
    - Memory management and efficient processing is as important as the communication protocol
  - Data transfer requires a lot of work on the Kernel's part
- A delay in one component will quickly propagate to others
  - A “clog” in one part of the pipe creates a jam for everyone in the pipe
    - And is expensive to fix (think re-transmissions and RST Flags)
  - Variable delay is worse than consistent larger delay
    - Variable delay can propagate and can cause odd interactions between components
- May be better to use UDP as there are less moving parts