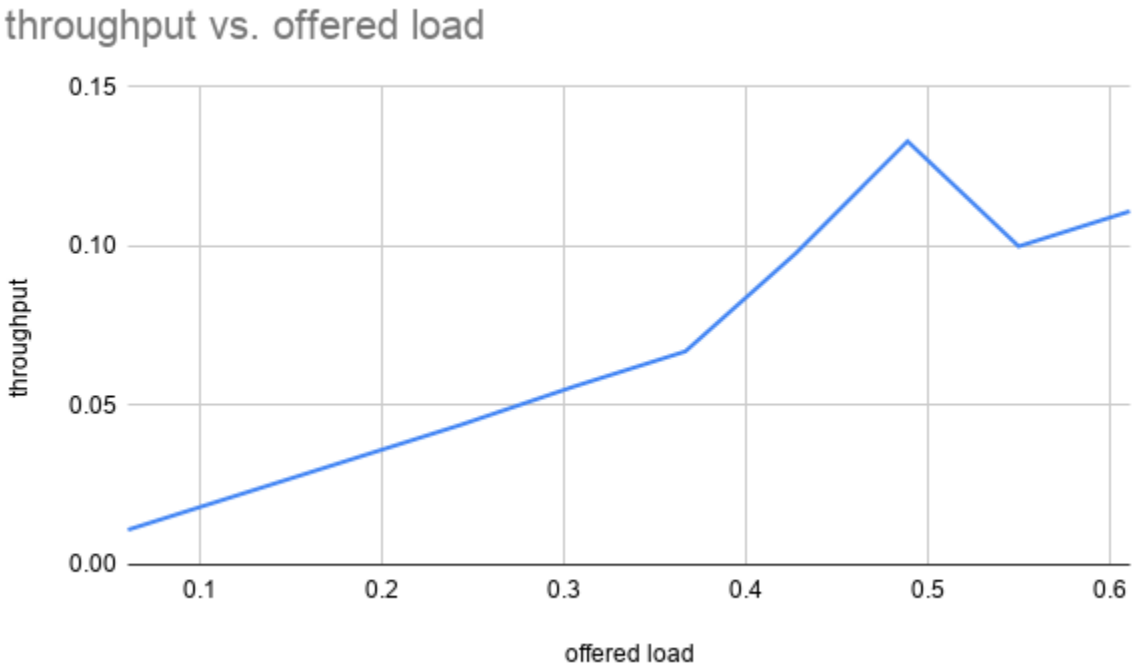
Introduction

Time-division multiple access is a channel access method for shared-medium networks. It allows several users to share the same frequency channel by dividing the signal into different time slots. The users transmit in rapid succession, one after the other, each using its own time slot. One of our aims was to create and implement a TDMA protocol and then to compare our throughput result with an ALOHA protocol.

Script

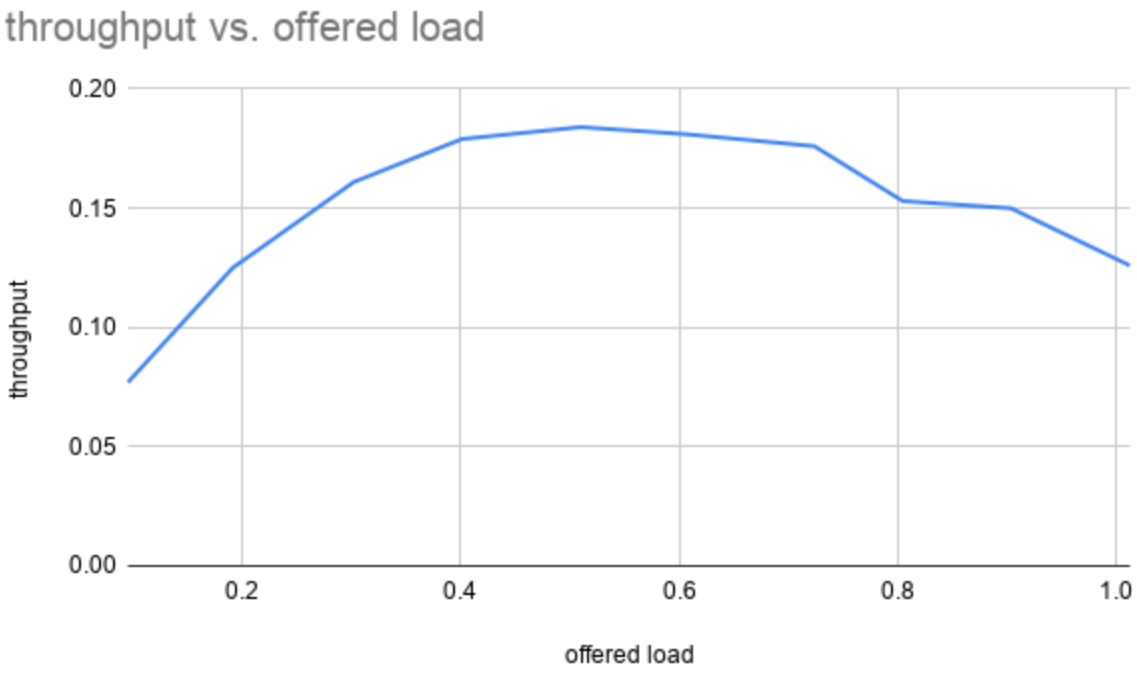
So in this document we see how different “Behavior” models are implemented. Now Poisson Behaviour that randomly repeats a behaviour at a certain interval is used in ALOHA. For our TDMA model we are using another repeating behavior “Ticker Behavior”. We will schedule different nodes to send datagrams at different time slots. And we will send the datagram to 2 from both 1 and 3. We will also send datagrams randomly in the ALOHA protocol and then compare the throughput.

The TDMA Script simulation gave this graph



Now we’ll see what if the nodes randomly send datagrams to the specific node randomly in ALOHA.

This is the graph for ALOHA.



ALOHA has higher throughput. But it peaks around 0.5 while the TDMA script seems to peak and then go back upwards. Further analysis can be made to see which script will be more useful.

Code

A TickerBehavior is run repeatedly with a specified delay between invocations. The ticker behavior may be terminated by calling stop() at any time.

A PoissonBehavior is similar to a ticker behavior, but the interval between invocations is an exponentially distributed random variable. This simulates a Poisson arrival process.

```
println '''
TX Count\tRX Count\tOffered Load\tThroughput
--------\t--------\t------------\t----------'''

for (def load = minLoad; load <= maxLoad; load += loadStep) {

  simulate T, {
                    // simulate 2 hours of elapsed time
    nodes.each { myAddr ->
      def myNode = node "${myAddr}", address: myAddr, location: nodeLocation[myAddr]

      myNode.startup = {                    // startup script to run on each node
        if (myAddr==1){
        def phy = agentForService PHYSICAL
        def arrivalRate = load/nodes.size()   // arrival rate per node
        add new TickerBehavior((Long)(1000/arrivalRate),
        {   // avg time between events in ms
          // drop any ongoing TX/RX and then send frame to random node, except myself
          phy << new ClearReq()
          phy << new TxFrameReq(to: 2, type: DATA)
        })
        }
        if (myAddr==2){
        def phy = agentForService PHYSICAL
        def arrivalRate = load/nodes.size()   // arrival rate per node
        add new TickerBehavior((Long)(2000/arrivalRate),
        {   // avg time between events in ms
          // drop any ongoing TX/RX and then send frame to random node, except myself
          phy << new ClearReq()
          phy << new TxFrameReq(to: 3, type: DATA)
        })
        }
        if (myAddr==3){
        def phy = agentForService PHYSICAL
        def arrivalRate = load/nodes.size()   // arrival rate per node
        add new TickerBehavior((Long)(3000/arrivalRate),
        {   // avg time between events in ms
          // drop any ongoing TX/RX and then send frame to random node, except myself
          phy << new ClearReq()
          phy << new TxFrameReq(to: 2, type: DATA)
        })
        }

      }
```

To calculate the time slot, the arrival rate is load by the number of nodes. So for each node we will have a different time slot based on the number/name of the node. This can always be changed as per the requirements. Then the same message sending script as we've used in all the other protocols.

Improvement

So instead of using the random arrival rate based time slot we can give them a fixed time slot. This will give a throughput of around 0.9 and a fixed offered load.

```
6678            5670            1.060           0.900
6678            5670            1.060           0.900
6678            5670            1.060           0.900
6678            5670            1.060           0.900
6678            5670            1.060           0.900
6678            5670            1.060           0.900
6678            5670            1.060           0.900
6678            5670            1.060           0.900
```

This is much higher than the ALOHA output and also if you keep the time slot value very far apart you can further increase the throughput.
#Experiment: Try to put in different time slots with different differences and see which gives the most throughput.

The Screenshot of the final code used is present here:

```
println '''
TX Count\tRX Count\tOffered Load\tThroughput
--------\t--------\t------------\t----------'''

for (def load = minLoad; load <= maxLoad; load += loadStep) {

  simulate T, {
                // simulate 2 hours of elapsed time
    nodes.each { myAddr ->
      def myNode = node "${myAddr}", address: myAddr, location: nodeLocation[myAddr]

      myNode.startup = {                    // startup script to run on each node
        if (myAddr==1){
        def phy = agentForService PHYSICAL
        def arrivalRate = load/nodes.size()   // arrival rate per node
        add new TickerBehavior((long)(1000),
        {   // avg time between events in ms
          // drop any ongoing TX/RX and then send frame to random node, except myself
          phy << new ClearReq()
          phy << new TxFrameReq(to: 2, type: DATA)
        })
        }
        if (myAddr==2){
        def phy = agentForService PHYSICAL
        def arrivalRate = load/nodes.size()   // arrival rate per node
        add new TickerBehavior((long)(25000),
        {   // avg time between events in ms
          // drop any ongoing TX/RX and then send frame to random node, except myself
          phy << new ClearReq()
          phy << new TxFrameReq(to: 3, type: DATA)
        })
        }
        if (myAddr==3){
        def phy = agentForService PHYSICAL
        def arrivalRate = load/nodes.size()   // arrival rate per node
        add new TickerBehavior((long)(50000),
        {   // avg time between events in ms
          // drop any ongoing TX/RX and then send frame to random node, except myself
          phy << new ClearReq()
          phy << new TxFrameReq(to: 2, type: DATA)
        })
        }

      }
```