Windowed backoff algorithms for WiFi: theory and performance under batched arrivals

Project in computer networks

Spring 2023

Team members:

Musa Eisa

Saadi Saadi

Instructor: Eran Tavor

Contents

Project overview:

Binary Exponential Backoff (BEB) is an old method used to coordinate how devices share a communication channel. This is important in modern networks like WiFi for wireless communication.

In this project we want to see how well BEB works when a group of data packets all try to use a wireless channel at the same time. We tested it alongside newer methods, using a simulation tool called Network Simulator 3.

After that we want to present our new algorithms and compare them with the other algorithms using ns3.

NS-3 is a free open source project aiming to build a discrete-event network simulator targeted for simulation research and education.

BEB overview:

This is a small explanation of how BEB algorithm works:

1. Initial Transmission Attempt:

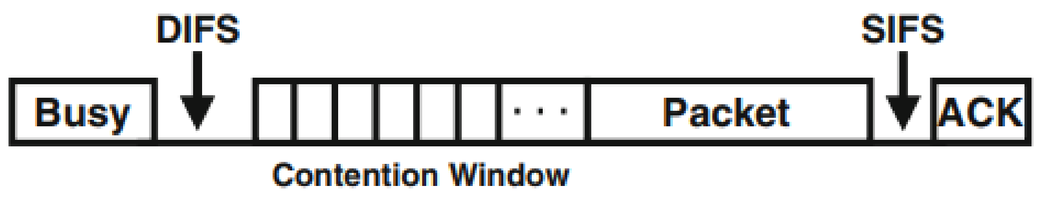
- When a station wants to send data over a shared channel, it first checks if the channel is not being used by other stations. If the channel is idle, the station proceeds to transmit its data.
- Before transmitting, the station waits for a Distributed Inter-Frame Space (DIFS) duration. DIFS is a fixed period of time, acting as a kind of "wait and listen" interval.

2. Contention Window and Slot Selection:

- Stations have a contention window (cw) which represents a range of possible waiting times. Each station begins with a specific cw size.
- If the channel is busy due to another station transmitting when the station wants to send data, the station waits until the ongoing transmission is completed.
- After the DIFS period, the station selects a random waiting time, known as a "slot," between 0 and (w-1) slots, where 'w' represents the current size of its contention window.
- This randomness helps reduce the chances of collisions between stations trying to transmit simultaneously.

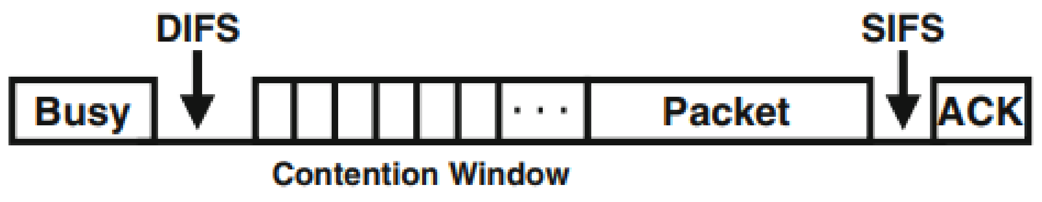
3. Handling Collisions:

- In the event of a collision, where two or more stations attempt to transmit data at the same time, the station detects the collision.
- Upon detecting a collision, the station must wait for a DIFS duration again.
- After DIFS, it chooses a new random slot between 0 and (w-1) slots, where 'w' is the new size of the contention window.
- In the BEB algorithm, the contention window size typically doubles after each unsuccessful transmission attempt. This is referred to as "exponential backoff." For instance, if the previous window size was 'w,' it becomes '2w' after a collision.



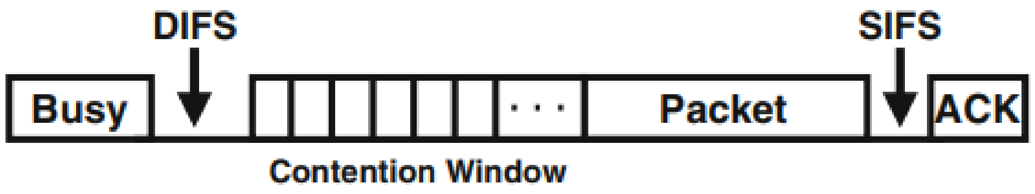
4. ACKnowledgment and Collision Detection:

- After a station successfully transmits its data, it awaits an acknowledgment (ACK) from the receiving station. The ACK signals that the data was received without issues.
- If the transmission was successful, the receiving station waits for a short inter-frame space (SIFS) duration before sending the ACK. SIFS is shorter than DIFS and is used to quickly respond to successful transmissions.
- Upon receiving the ACK, the sending station learns that its transmission was successful.
- However, if the station doesn't receive an ACK within a specific time frame known as the ACK-timeout duration, it assumes that a collision occurred.



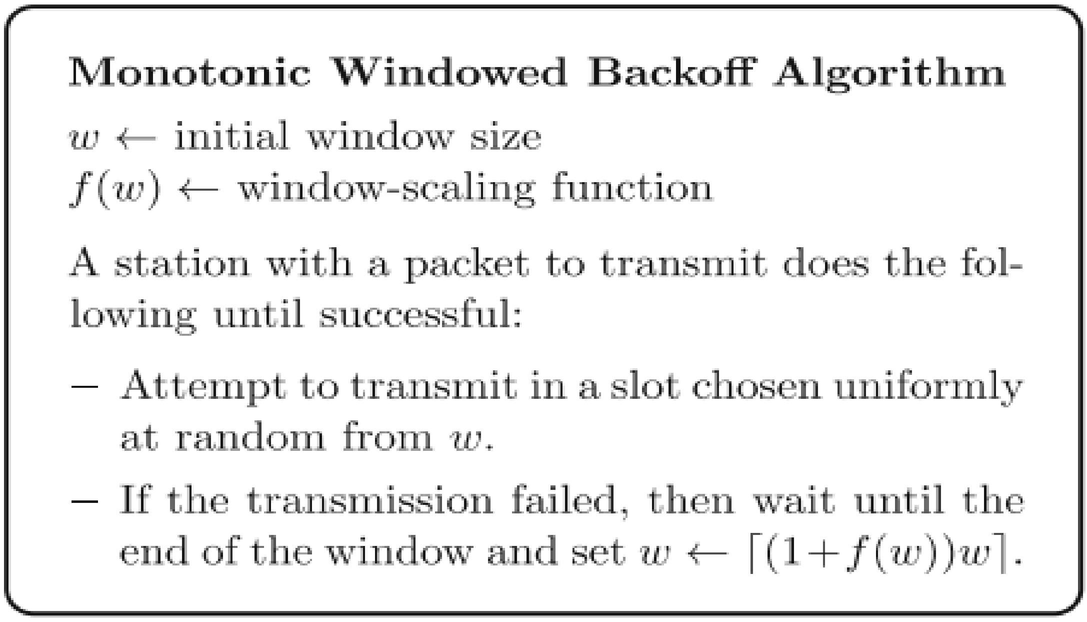
Finally, RTS/CTS (request-to-send/clear-to-send) is an optional mechanism. Informally, a station will send an RTS message and await a CTS message from the receiver prior to transmitting its data. Due to increased overhead, there is some debate on whether RTS/CTS is worthwhile and, if so, when it should be enabled. typically, it is not. Here, we focus on the case where RTS/CTS is disabled.

In summary, the Binary Exponential Backoff (BEB) algorithm is a method for coordinating access to a shared communication channel. It involves checking for channel availability, selecting random slots to minimize collisions, and adjusting the contention window size after each transmission attempt to improve the chances of successful data transfer. It also employs collision detection mechanisms to handle unsuccessful transmissions.



Monotonic Windowed Backoff Algorithms

This is a family of algorithms where their window size always increases, which is why they are referred to as 'monotonic.' All the algorithms within this family share the same formula depicted in the picture to determine their next window size.



Monotonic Windowed Backoff Algorithm

$w \leftarrow$ initial window size
$f(w) \leftarrow$ window-scaling function

A station with a packet to transmit does the following until successful:

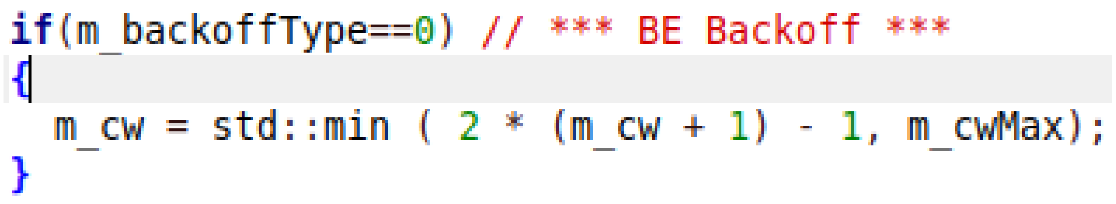
- Attempt to transmit in a slot chosen uniformly at random from w.
- If the transmission failed, then wait until the end of the window and set $w \leftarrow \lceil (1+f(w))w \rceil$.

What sets them apart is the function f(w), which serves as a window scaling function. This function takes the previous size as input and transforms it into a non-negative numerical value. This value is then added to 1, forming the factor by which the previous size is multiplied to determine the new size. This family of algorithms includes BEB, LB, and LBB as its members.

BEB:

For BEB (Binary Exponential Backoff), the window scaling function F(w) is indeed a constant 1, regardless of the previous window size's magnitude. As a result, the factor for BEB remains consistently at 2, meaning the new window size is always double the size of the older one, regardless of the older window's size.

This is BEB method code:

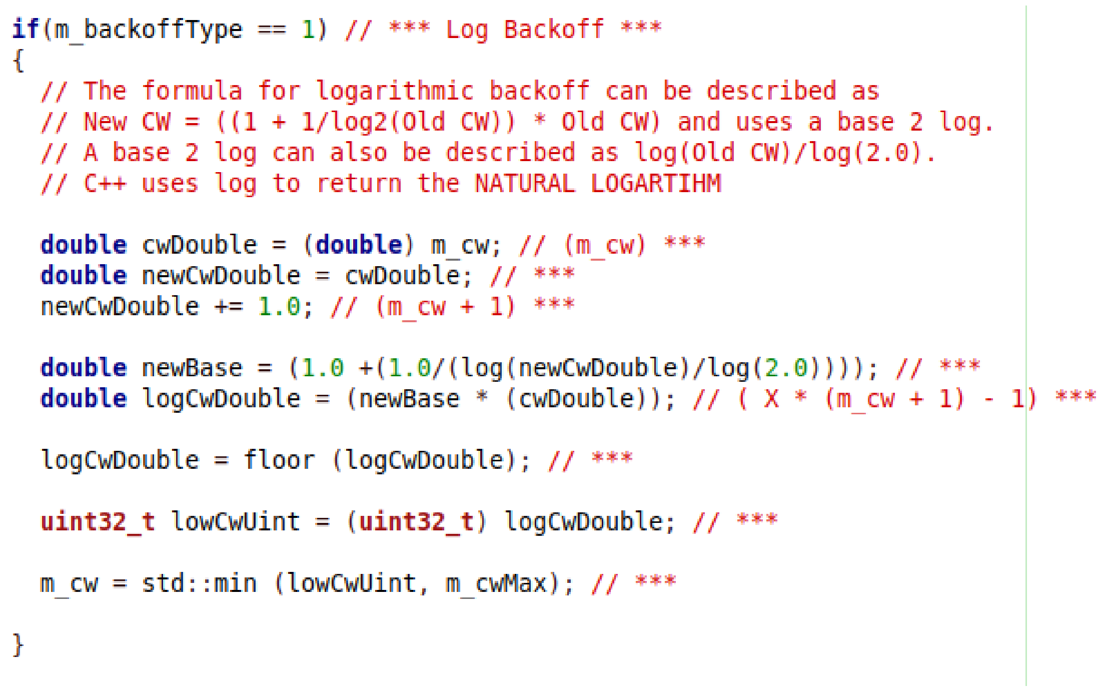


```
if(m_backoffType==0) // *** BE Backoff ***
{
  m_cw = std::min ( 2 * (m_cw + 1) - 1, m_cwMax);
}
```

LB:

For LB (Log Backoff), its window scaling function f(w), is defined as 1/log(w). Consequently, the factor for LB is represented as 1 + 1/log(w), which determines the new window size.

This is LB method code:



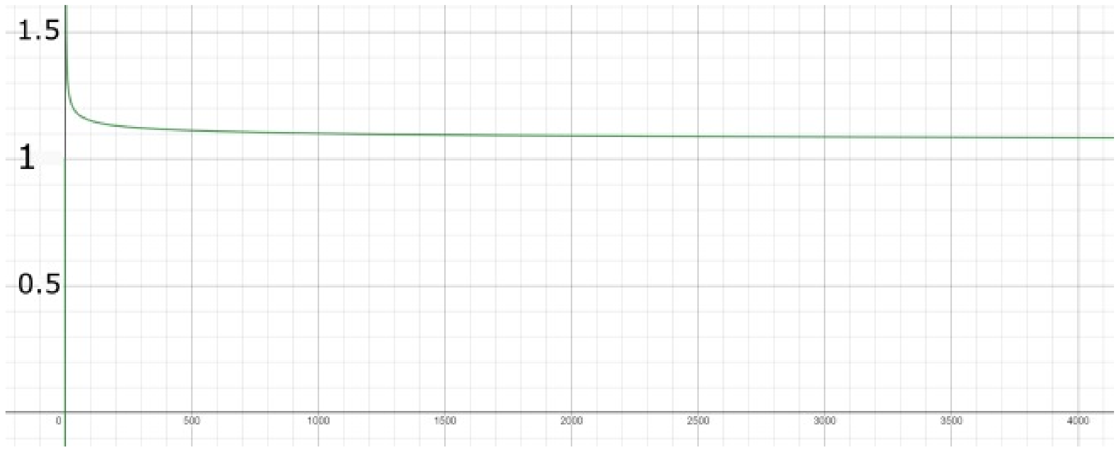
```
if(m_backoffType == 1) // *** Log Backoff ***
{
  // The formula for logarithmic backoff can be described as
  // New CW = ((1 + 1/log2(Old CW)) * Old CW) and uses a base 2 log.
  // A base 2 log can also be described as log(Old CW)/log(2.0).
  // C++ uses log to return the NATURAL LOGARTIHM

  double cwDouble = (double) m_cw; // (m_cw) ***
  double newCwDouble = cwDouble; // ***
  newCwDouble += 1.0; // (m_cw + 1) ***

  double newBase = (1.0 +(1.0/(log(newCwDouble)/log(2.0)))); // ***
  double logCwDouble = (newBase * (cwDouble)); // ( X * (m_cw + 1) - 1) ***

  logCwDouble = floor (logCwDouble); // ***

  uint32_t lowCwUint = (uint32_t) logCwDouble; // ***

  m_cw = std::min (lowCwUint, m_cwMax); // ***

}
```

F(w) is a decreasing monotonic function for LB (Log Backoff) because it is inversely proportional to the logarithm of w. As w increases, the value of F(w) decreases, which results in a larger factor (1 + 1/(log(w))) for smaller window sizes and a smaller factor for larger window sizes. For the minimal window size, F(w) equals1.5, while for the maximal window size, it's around 1.083. This demonstrates the decreasing nature of F(w) with respect to window size.

$$1+\frac{1}{\log(4)} = 1.5$$

$$1+\frac{1}{\log(4096)} = 1.083$$

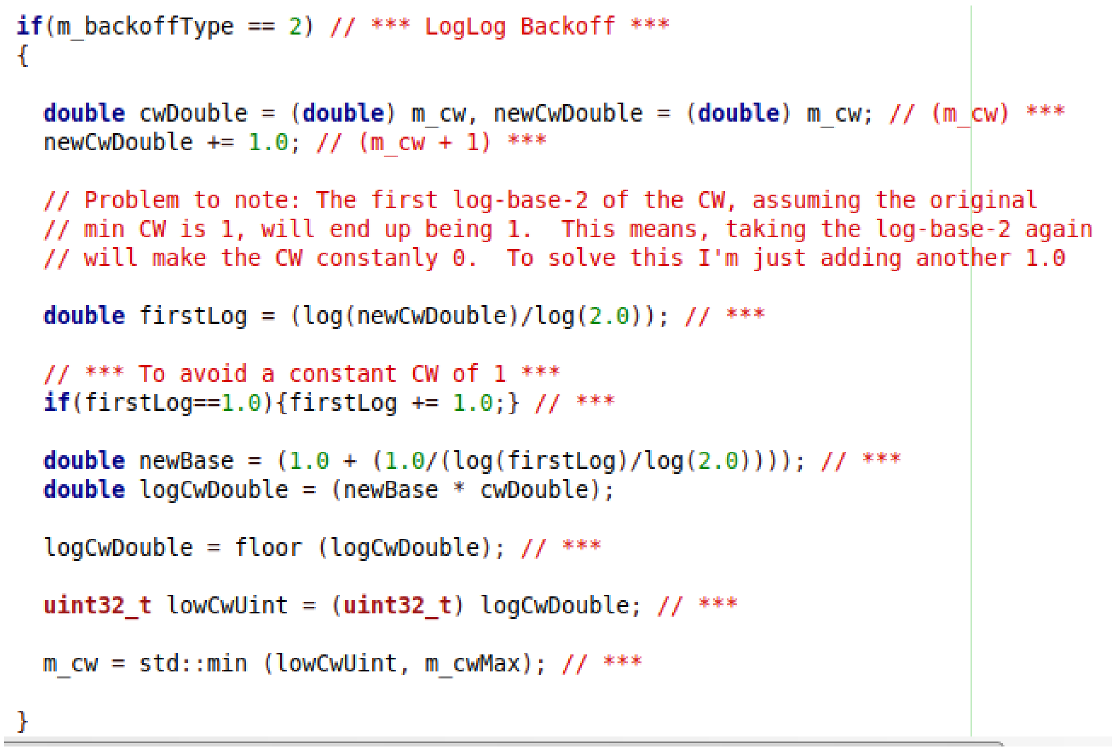
The factor of LB as a function of the older window size:



LLB:

LLB, which stands for Log Log Backoff, is similar to LB (Log Backoff). Its function f(w) is equal to 1/log(log(w)), resulting in a factor of 1 + 1/log(log(w)) that determines the new window size.

This is LLB method code:



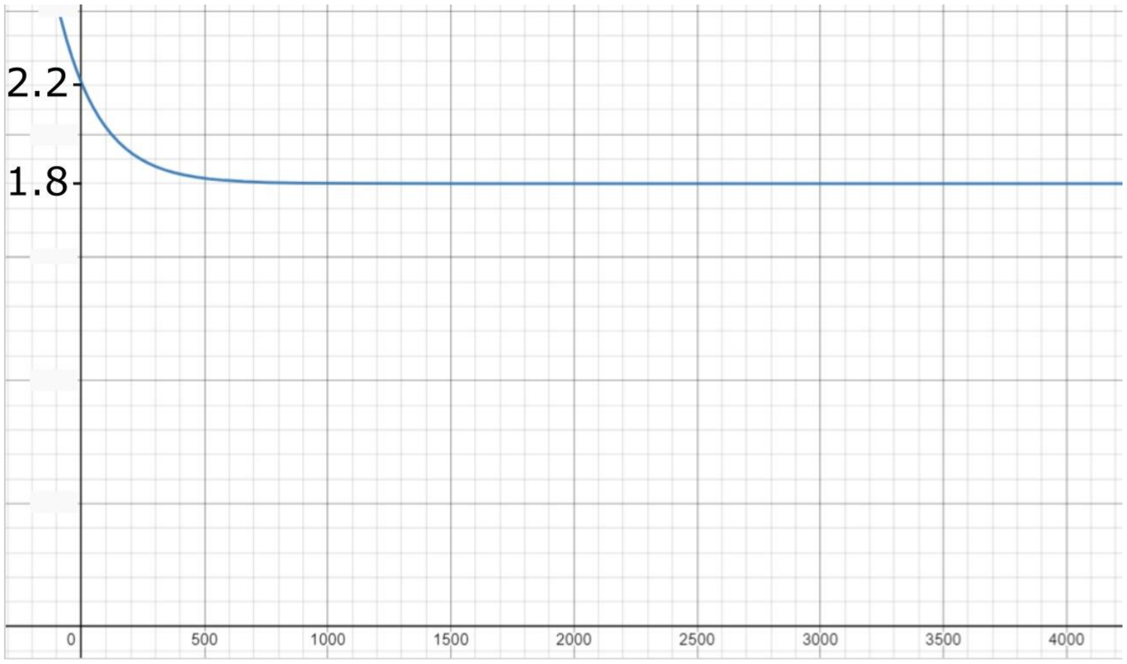
```
if(m_backoffType == 2) // *** LogLog Backoff ***
{

  double cwDouble = (double) m_cw, newCwDouble = (double) m_cw; // (m_cw) ***
  newCwDouble += 1.0; // (m_cw + 1) ***

  // Problem to note: The first log-base-2 of the CW, assuming the original
  // min CW is 1, will end up being 1.  This means, taking the log-base-2 again
  // will make the CW constanly 0.  To solve this I'm just adding another 1.0

  double firstLog = (log(newCwDouble)/log(2.0)); // ***

  // *** To avoid a constant CW of 1 ***
  if(firstLog==1.0){firstLog += 1.0;} // ***

  double newBase = (1.0 + (1.0/(log(firstLog)/log(2.0)))); // ***
  double logCwDouble = (newBase * cwDouble);

  logCwDouble = floor (logCwDouble); // ***

  uint32_t lowCwUint = (uint32_t) logCwDouble; // ***

  m_cw = std::min (lowCwUint, m_cwMax); // ***

}
```

F(w) for LLB is also a decreasing monotonic function due to the fact that log(log(w)) is an increasing monotonic function. This results in F(w) yielding a value of 2 for the minimal window size and approximately 1.278 for the maximal window size.

$$1+\frac{1}{\log\log(4)}=2$$

$$1+\frac{1}{\log\log(4069)}=1.278$$

The factor of LLB as a function of the older window size



BEB, LB and LLB behavior:

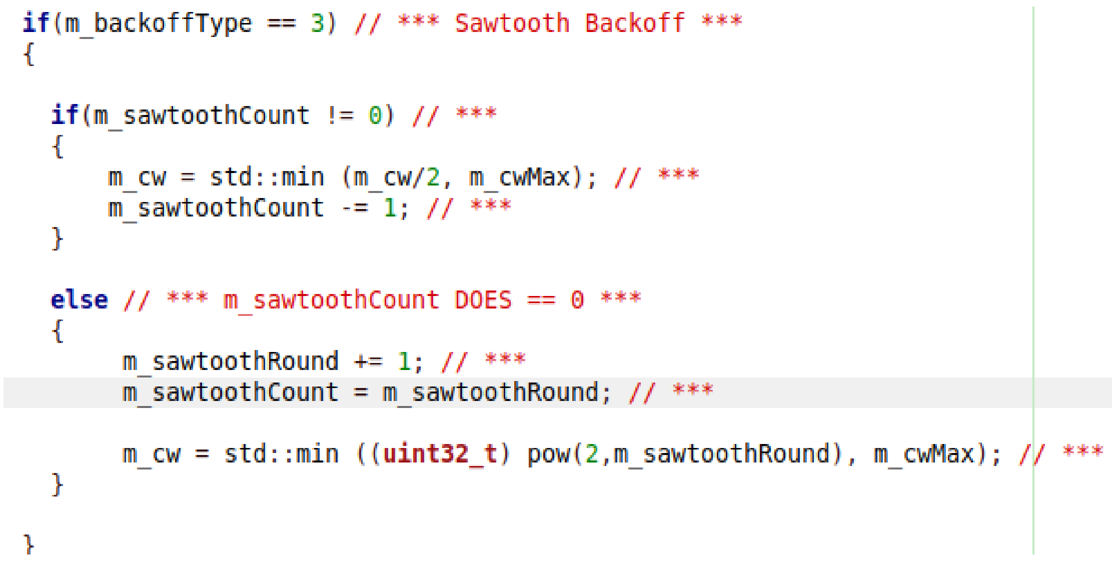
In the BEB method, each time data is resent, the contention window (CW size) doubles consistently, with a constant factor of 2. However, in the case of the other two methods, LB and LLB, when data is resent, the growth of the contention window size depends on its current size. If the window is small, the growth factor is larger, and if the window is large, the growth factor is smaller. While BEB is more efficient in terms of execution time (as we will see), the other algorithms possess the advantage of adjusting the window size rapidly at the beginning, particularly for smaller previous window sizes, and they offer a smaller growth factor for larger previous window sizes.

Despite the superiority of BEB in execution time, it's worth noting that you can still leverage the adaptable window size growth feature of LB and LLB in conjunction with BEB to enhance its performance. This is because BEB continues to double its window size by a factor of 2, even when it may not be necessary, especially for larger previous window sizes.

STB algorithm:

Now we are going to talk about another algorithm that is mentioned in the article, which is STB (sowtooth backoff) , this algorithm is not part of the monotonic family , this algorithm is different and it uses two loops to set the monotonic window the outer loop doubles the older window size exactly like BEB , but we also have an inner loop that goes over all the previous window sizes : this is the sequence of sizes that this algorithm gives : 4 8 4 16 8 4 32 16 8 4…

This is STB method code:



```
if(m_backoffType == 3) // *** Sawtooth Backoff ***
{

  if(m_sawtoothCount != 0) // ***
  {
      m_cw = std::min (m_cw/2, m_cwMax); // ***
      m_sawtoothCount -= 1; // ***
  }

  else // *** m_sawtoothCount DOES == 0 ***
  {
       m_sawtoothRound += 1; // ***
       m_sawtoothCount = m_sawtoothRound; // ***

       m_cw = std::min ((uint32_t) pow(2,m_sawtoothRound), m_cwMax); // ***
  }

}
```

Even though BEB outperforms this algorithm in terms of execution time, it introduces an intriguing concept. It involves a unique approach where, after utilizing a larger window size, it reverts to smaller window sizes for subsequent transmission attempts. The underlying idea is that employing a larger window size might successfully transmit some packets, reducing the overall number of packets in the queue. Consequently, when transitioning to smaller window sizes, there's a better chance of success due to the reduced packet count. To leverage this concept, the inner loop within the algorithm traverses from larger window sizes to smaller ones. We experimented with this feature in one of our algorithms to enhance the performance of BEB

Experimental setup:

In this section, we describe and discuss our simulation tools, which protocols to simulate, the main assumptions and parameters for the simulations.

We used NS3 to run our simulations, NS-3, short for Network Simulator 3, is a widely used open-source software tool for simulating and modeling computer networks. It provides a powerful environment for researchers, engineers, and students to simulate and study the behavior of network protocols, applications, and devices in a controlled and reproducible manner. NS-3 is particularly valuable for testing and evaluating new networking algorithms, technologies, and architectures before real-world deployment. Its flexibility and extensibility make it a valuable resource in the field of network research and development.

Main assumptions:

1. **MAC Layer and Protocol:** Our experiments are conducted at the Medium Access Control (MAC) layer using the IEEE 802.11g standard.

2. **Contending Algorithms:** We introduce modifications to the behavior of the contention window, which varies between a minimum size of 4 and a maximum size of 4096. These changes are influenced by the algorithms under investigation. All experiments utilize IPv4 and User Datagram Protocol (UDP).

3. **Packet Overhead:** Each transmitted packet includes an overhead of 64 bytes. This comprises 8 bytes for UDP, 20 bytes for IPv4, 8 bytes for an LLC/SNAP header, and an additional 28 bytes at the MAC layer.

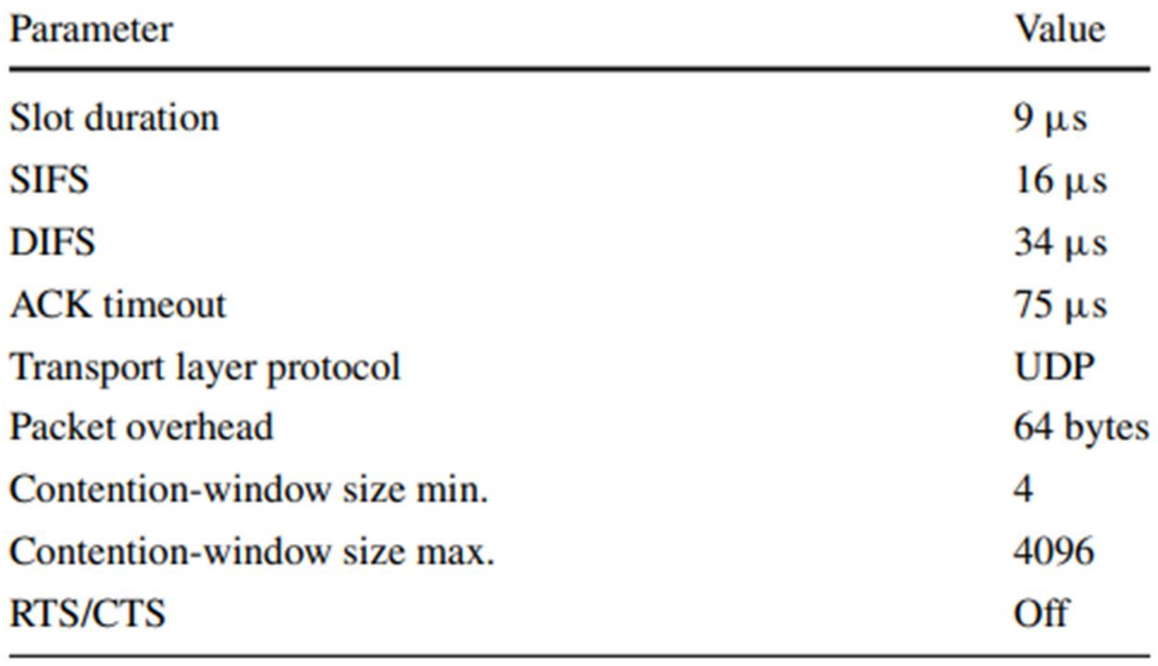
4. **Timing Parameters:** In our experiments, unless specified otherwise, we set the slot time to 9 microseconds (μs), the Short Inter-Frame Space (SIFS) to 16 μs, and the Distributed Inter-Frame Space (DIFS) to 34 μs. The acknowledgement (ACK) timeout duration is 75 μs.

5. **Network Layout:** We deploy 'n' stations within a 40-meter by 40-meter grid. These stations are arranged in a regular pattern, starting from the southwest corner of the grid and moving from left to right in 2-meter increments. When a row is full, the stations move up to the next row. Additionally, a wireless access point (AP) is situated approximately at the center of this grid.

6. **RTS/CTS Disabled:** We assume that the Request to Send (RTS) and Clear to Send (CTS) mechanisms are disabled for the purpose of our experiments.

7. **A single batch:** We examine a single batch of n packets that simultaneously begin their contention for the channel (n stations every station sends a single packet).

8. **Backoff algorithms:** we experiment with the following backoff algorithms: Log- Backoff (LB), LogLog- Backoff (LLB) and Sawtooth- Backoff (STB),and after that our new algorithms.



Parameter	Value
Slot duration	9 μs
SIFS	16 μs
DIFS	34 μs
ACK timeout	75 μs
Transport layer protocol	UDP
Packet overhead	64 bytes
Contention-window size min.	4
Contention-window size max.	4096
RTS/CTS	Off

theoretical guarantees on CW slots:

In theoretical analysis, we quantify time in slots, abstract time units. Theoretical studies often focus on the number of slots required for completing 'n' packets, but this count pertains specifically to contention windows (CWs). For instance, in the BEB method, 'n' packets typically complete in approximately Θ(n log n) CW slots.

In our simulator-based evaluations, we use the term "contention-window slots" (CW slots) to refer to this quantity. The table below summarizes the known with-high-probability (w.h.p.) guarantees regarding CW slots, where the Θ-notation implies both upper and lower bounds on the count of CW slots needed for all 'n' packets to succeed.

Comparing these guarantees, LB, LLB, and STB outperform BEB concerning CW slots. Notably, STB achieves Θ(n) CW slots, which is considered asymptotically optimal.

Algorithm	Contention-window slots
BEB	$\Theta(n \log n)$
LB	$\Theta\left(\frac{n \log n}{\log \log n}\right)$
LLB	$\Theta\left(\frac{n \log \log n}{\log \log \log n}\right)$
STB	$\Theta(n)$

How to run the simulations?

In this section, we will outline the procedure for setting up NS3 and executing our simulations for a range of algorithms, including BEB, LB, LLB, STB, and our newly proposed methods. We will initially cover simulations with client counts ranging from 10 to 150, followed by simulations involving different client quantities.

For our simulations, we utilized NS version 3.25 in conjunction with an environment comprising Ubuntu 18.04.4 LTS, a g++ compiler version 7.5.0, and Python 2.7.

Required files:

First you need to download all the files from this link:

https://github.com/trishac97/NS3-802.11g-Backoff/tree/main

and then download dcf-manager.cc file from our gitlab:

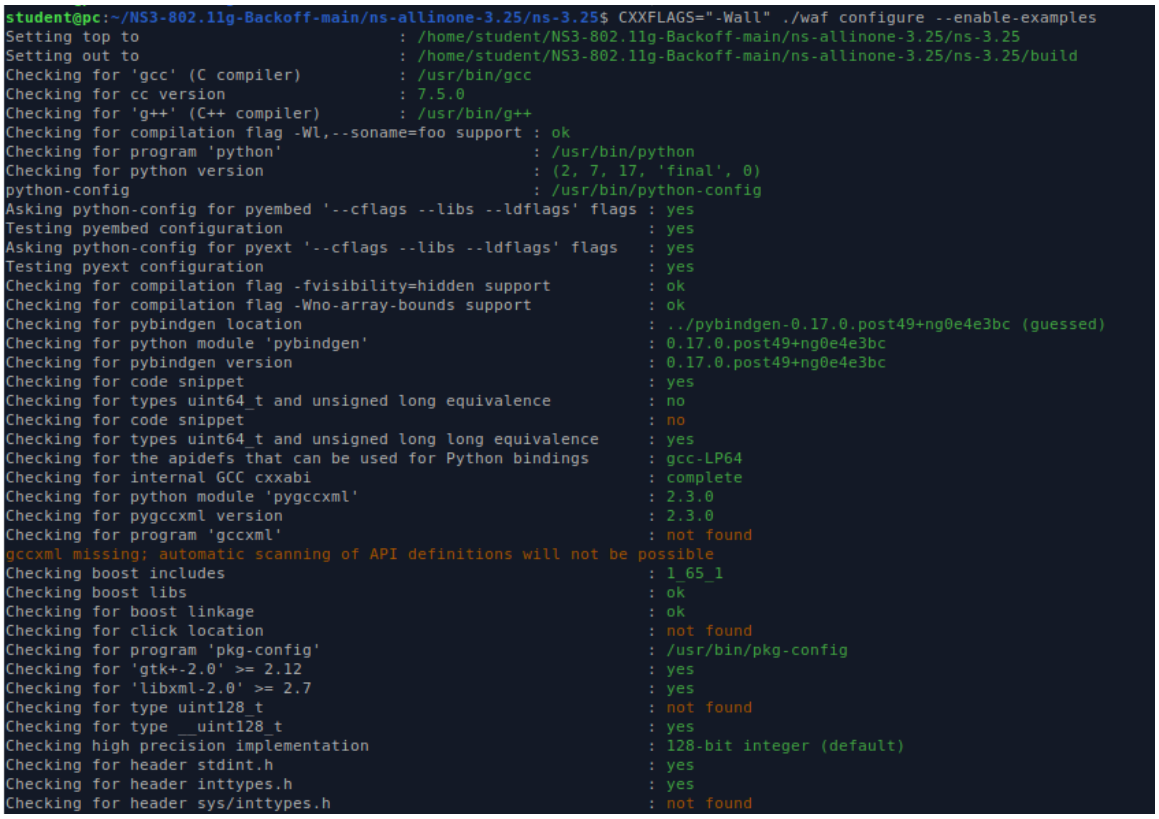
https://gitlab.cs.technion.ac.il/tavran/sp2023_beb_on_burst_traffic

replace this file with dcf-manager.cc that already exist in:
“ns-allinone-3.25/ns-3.25/src/wifi/model/”

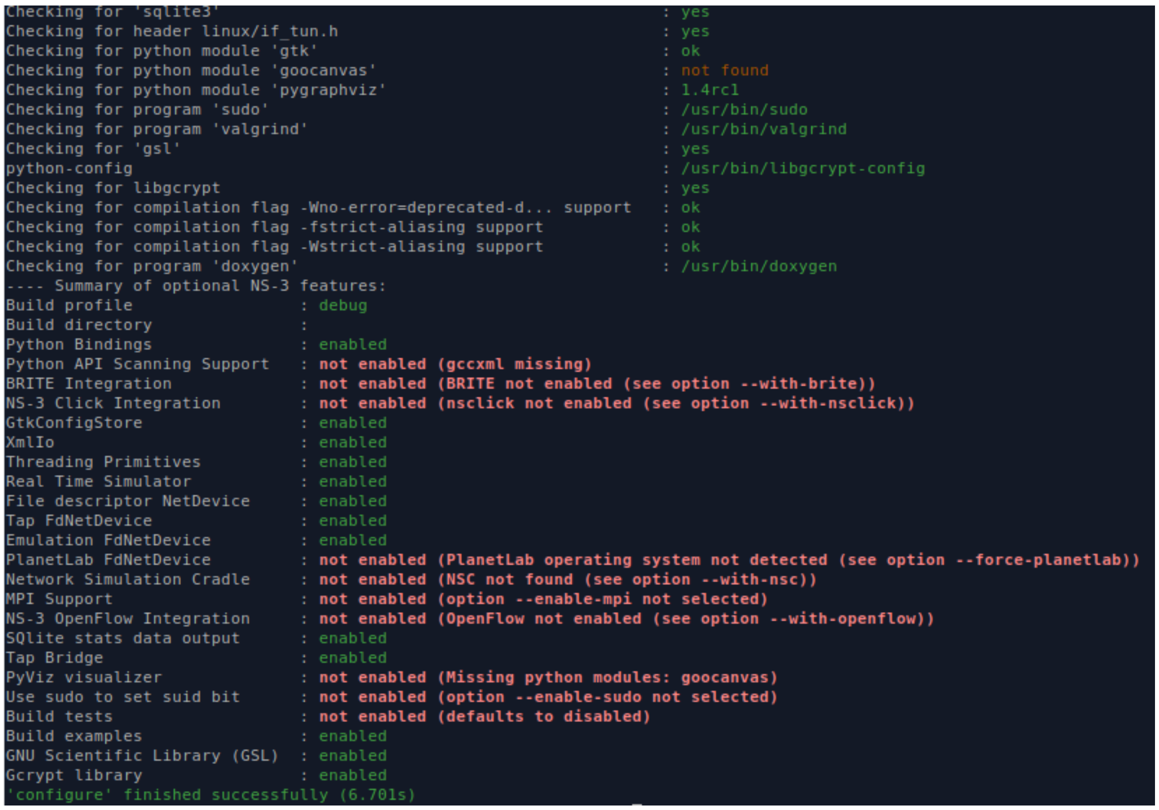
add the file Windowed-Backoff-Algorithms-for-WiFi/NS3-Scripts/nonQOS/nonQOS.cc at location:
~/ns-allinone-3.25/ns-3.25/scratch

Building ns3:

The code for the framework and the default models provided by ns-3 is built as a set of libraries. User simulations are expected to be written as simple programs that make use of these ns-3 libraries.
To build the set of default libraries and the example programs included in this package, you need to use the tool 'waf'.
First from ‘ns-allinone-3.25/ns-3.25/’ type the following command:
CXXFLAGS="-Wall" ./waf configure --enable-examples
You should get something like this:



```
student@pc:~/NS3-802.11g-Backoff-main/ns-allinone-3.25/ns-3.25$ CXXFLAGS="-Wall" ./waf configure --enable-examples
Setting top to                           : /home/student/NS3-802.11g-Backoff-main/ns-allinone-3.25/ns-3.25
Setting out to                           : /home/student/NS3-802.11g-Backoff-main/ns-allinone-3.25/ns-3.25/build
Checking for 'gcc' (C compiler)          : /usr/bin/gcc
Checking for cc version                  : 7.5.0
Checking for 'g++' (C++ compiler)        : /usr/bin/g++
Checking for compilation flag -Wl,--soname=foo support : ok
Checking for program 'python'                          : /usr/bin/python
Checking for python version                            : (2, 7, 17, 'final', 0)
python-config                                          : /usr/bin/python-config
Asking python-config for pyembed '--cflags --libs --ldflags' flags : yes
Testing pyembed configuration                                      : yes
Asking python-config for pyext '--cflags --libs --ldflags' flags   : yes
Testing pyext configuration                                        : yes
Checking for compilation flag -fvisibility=hidden support          : ok
Checking for compilation flag -Wno-array-bounds support            : ok
Checking for pybindgen location                                    : ../pybindgen-0.17.0.post49+ng0e4e3bc (guessed)
Checking for python module 'pybindgen'                             : 0.17.0.post49+ng0e4e3bc
Checking for pybindgen version                                     : 0.17.0.post49+ng0e4e3bc
Checking for code snippet                                          : yes
Checking for types uint64_t and unsigned long equivalence          : no
Checking for code snippet                                          : no
Checking for types uint64_t and unsigned long long equivalence     : yes
Checking for the apidefs that can be used for Python bindings      : gcc-LP64
Checking for internal GCC cxxabi                                   : complete
Checking for python module 'pygccxml'                              : 2.3.0
Checking for pygccxml version                                      : 2.3.0
Checking for program 'gccxml'                                      : not found
gccxml missing; automatic scanning of API definitions will not be possible
Checking boost includes                                            : 1_65_1
Checking boost libs                                                : ok
Checking for boost linkage                                         : ok
Checking for click location                                        : not found
Checking for program 'pkg-config'                                  : /usr/bin/pkg-config
Checking for 'gtk+-2.0' >= 2.12                                    : yes
Checking for 'libxml-2.0' >= 2.7                                   : yes
Checking for type uint128_t                                        : not found
Checking for type __uint128_t                                      : yes
Checking high precision implementation                             : 128-bit integer (default)
Checking for header stdint.h                                       : yes
Checking for header inttypes.h                                     : yes
Checking for header sys/inttypes.h                                 : not found
```



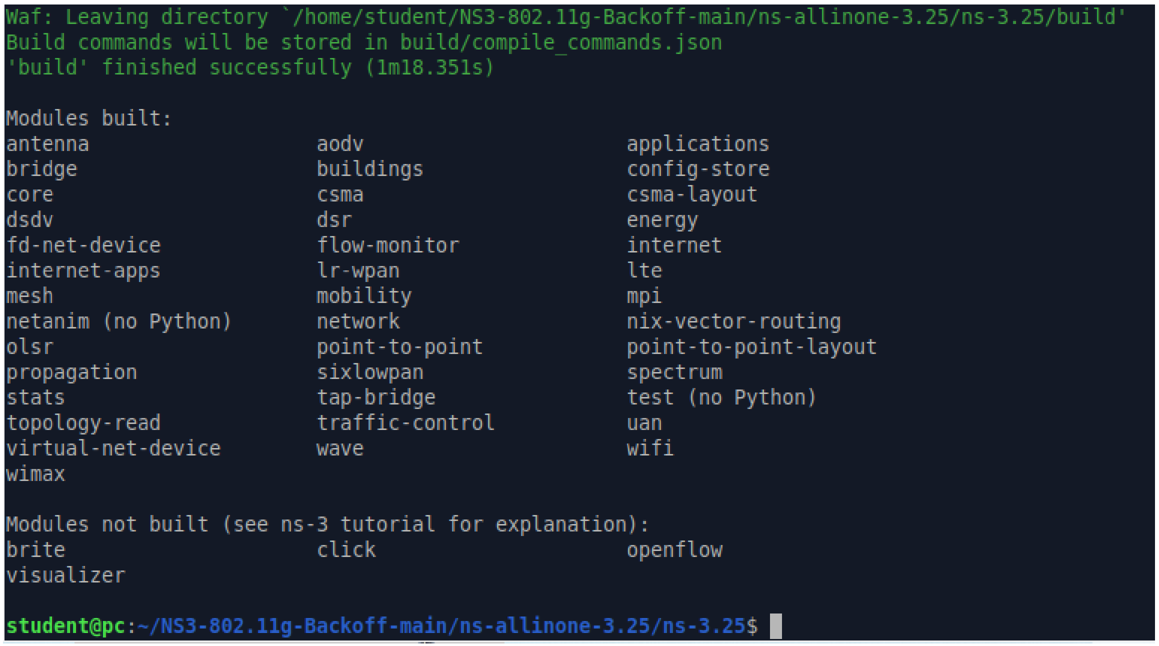
```
Checking for 'sqlite3'                                             : yes
Checking for header linux/if_tun.h                                 : yes
Checking for python module 'gtk'                                   : ok
Checking for python module 'goocanvas'                             : not found
Checking for python module 'pygraphviz'                            : 1.4rc1
Checking for program 'sudo'                                        : /usr/bin/sudo
Checking for program 'valgrind'                                    : /usr/bin/valgrind
Checking for 'gsl'                                                 : yes
python-config                                                      : /usr/bin/libgcrypt-config
Checking for libgcrypt                                             : yes
Checking for compilation flag -Wno-error=deprecated-d... support   : ok
Checking for compilation flag -fstrict-aliasing support            : ok
Checking for compilation flag -Wstrict-aliasing support            : ok
Checking for program 'doxygen'                                     : /usr/bin/doxygen
---- Summary of optional NS-3 features:
Build profile                 : debug
Build directory               :
Python Bindings               : enabled
Python API Scanning Support   : not enabled (gccxml missing)
BRITE Integration             : not enabled (BRITE not enabled (see option --with-brite))
NS-3 Click Integration        : not enabled (nsclick not enabled (see option --with-nsclick))
GtkConfigStore                : enabled
XmlIo                         : enabled
Threading Primitives          : enabled
Real Time Simulator           : enabled
File descriptor NetDevice     : enabled
Tap FdNetDevice               : enabled
Emulation FdNetDevice         : enabled
PlanetLab FdNetDevice         : not enabled (PlanetLab operating system not detected (see option --force-planetlab))
Network Simulation Cradle     : not enabled (NSC not found (see option --with-nsc))
MPI Support                   : not enabled (option --enable-mpi not selected)
NS-3 OpenFlow Integration     : not enabled (OpenFlow not enabled (see option --with-openflow))
SQlite stats data output      : enabled
Tap Bridge                    : enabled
PyViz visualizer              : not enabled (Missing python modules: goocanvas)
Use sudo to set suid bit      : not enabled (option --enable-sudo not selected)
Build tests                   : not enabled (defaults to disabled)
Build examples                : enabled
GNU Scientific Library (GSL)  : enabled
Gcrypt library                : enabled
'configure' finished successfully (6.701s)
```

Notice that may there are missing packages that you need to install to successfully run the simulation but if you get something like in the picture its enough.

After that type:

./waf

And after successfully building ns3 you should get something like this:



```
Waf: Leaving directory `/home/student/NS3-802.11g-Backoff-main/ns-allinone-3.25/ns-3.25/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1m18.351s)

Modules built:
antenna                   aodv                      applications
bridge                    buildings                 config-store
core                      csma                      csma-layout
dsdv                      dsr                       energy
fd-net-device             flow-monitor              internet
internet-apps             lr-wpan                   lte
mesh                      mobility                  mpi
netanim (no Python)       network                   nix-vector-routing
olsr                      point-to-point            point-to-point-layout
propagation               sixlowpan                 spectrum
stats                     tap-bridge                test (no Python)
topology-read             traffic-control           uan
virtual-net-device        wave                      wifi
wimax

Modules not built (see ns-3 tutorial for explanation):
brite                     click                     openflow
visualizer

student@pc:~/NS3-802.11g-Backoff-main/ns-allinone-3.25/ns-3.25$
```

Notice that building ns3 may take a while.

Running ns3:

On recent Linux systems, once you have built ns-3 (with examples enabled), it should be easy to run the sample programs with the following command, such as:

```
./waf --run simple-global-routing
```

That program should generate a simple-global-routing.tr text trace file and a set of simple-global-routing-xx-xx.pcap binary pcap trace files, which can be read by tcpdump -tt -r filename.pcap The program source can be found in the examples/routing directory.
However in this project we will use some bash script files to run our simulations because we want to run 30 simulations for every number of clients and for every algorithm that we want to test.
Files that you should know about to run the simulations:
`dcf-manager.cc` that you copied to :
~/ns-allinone-3.25/ns-3.25/src/wifi/model
this is the file where the algorithms BEB, LOG, LOGLOG, STB, and our new algorithms are implemented. Inside the constructor DcfState::DcfState, you'll find:

```
// *** This variable chooses the type of backoff to be used ***
    // 0 = BE Backoff
    // 1 = Log Backoff
    // 2 = LogLog Backoff
    // 3 = Sawtooth Backoff
    // 4 = our algorithm 2
    // 5 = our algorithm 3

m_backoffType (0), // ***
```

which allows you to select the contention resolution algorithm that you wish to execute.

Inside the function DcfState::UpdateFailedCw, you will find the rules for how the contention window is modified for each algorthm.
`nonQoS.cc` that you copied to:
`~/ns-allinone-3.25/ns-3.25/scratch`
so titled because RTS/CTS is turned off. Inside you will find the code for how clients are placed, their number (taken from the command line which is called inside simScript.bash described below), the packet size, and other parameters that you may wish to set.

The following scripts and processing files are used:

simScript.bash -- executes the algorithm (set in dcf-manager.cc) for a number of packets n=10, 20, ..., 150, and uses 30 trials for each value of n. The resulting log files are written to directory N[n] as log[trial number].txt; the directories N[n] are created by the script.

topParser.cpp -- produces an executable topP that processes the log[trial number].txt file to obtain the total statistics for the execution.

slotParser.cpp -- produces an executable slotP that processes the log[trial number].txt file to obtain the median statistics for the execution.

processData.bash -- copies topP into each N[n] directory, then executes topP to process the log files and concatenate the resulting output to a single file allTrials-[algorithm nam]-P[packet size]-D[distance value].txt in the parent directory.

processMedianData.bash -- same as the above, but for processing the data to obtain median statistics.

resetDirs.bash -- deletes all data in the N[n] directories and recompiles/recopies topP and slotP into each empty directory.

How to execute the experiment?

First Select your algorithm from inside dcf-manager.cc by selecting a number between 0 and 5 in line 75 for m_backoffType, for example if you want to use BEB algorithm then you should choose 0 like this:

```
// *** This variable chooses the type of backoff to be used ***
    // 0 = BE Backoff
    // 1 = Log Backoff
    // 2 = LogLog Backoff
    // 3 = Sawtooth Backoff
    // 4 = our algorithm 2
    // 5 = our algorithm 3

m_backoffType (0), // ***
```

Then the following algorithm will be executed:

```
if(m_backoffType==0) // *** BE Backoff ***
{
  m_cw = std::min ( 2 * (m_cw + 1) - 1, m_cwMax);
}
```

If you want to execute our algorithm 1 then you could simply change the factor ‘2’ to another number, for example if you want a factor of 3 you should simply type that:

```
if(m_backoffType==0) // *** BE Backoff ***
{
  m_cw = std::min ( 3 * (m_cw + 1) - 1, m_cwMax);
}
```

Select your packet size (myClientPacketSize) from line 111 and distance between the clients (myRoomStartX, myRoomStartY, myIncX, myIncY) from line 95 in nonQoS.cc.

From within the directory
~/ns-allinone-3.25/ns-3.25/Data

run the command:
bash resetDirs.bash

which will create new directories N10 to N150 with topP and slotP executables in each one.

From within the directory
~/ns-allinone-3.25/ns-3.25

run the command:

bash simScript.bash

After executing, this produces the log files in the directories N[n] within the subdirectory

~/ns-allinone-3.25/ns-3.25/Data

for n=10 to 150 populated with the log files for each trial. The execution/simulation may take a while, of course.

From within the directory
~/ns-allinone-3.25/ns-3.25/Data

run the command:

bash processData.bash

to grab the total statistics. This produces the file allTrials at location (/home/young/ns-allinone-3.25/ns-3.25/Data/).

How to plot the graphs?

To plot the graph we will use matlab code.

First Rename allTrials to alltrials-[alg name]-P[size]-D[meter].txt

Example: If you're using BEB for 64 Byte packets at distance 2 metres, rename allTrials to alltrials-BEB-P64-D2.txt.

From the directory

~/Windowed-Backoff-Algorithms-for-WiFi/MATLAB-Code/Comparison-Plot/

Run callPlot.m

This function takes as input:

-- packet --> takes packet size as input {'P64','P256','P1024'}
-- xFont --> font for x-axis title
-- yFont --> font for y-axis title
-- lFont --> font for legend
-- gFont --> font for x & y axis scale

This function calls InputData, excludeOutliners, makePlot, pruneNAN internally and creates plots for following metrices: slotTime, packetTime, missedACKtime, totalACKtime.

Create Directory named 'P[size]' and keep alltrials-[alg name]-P[size]-D[meter].txt in it. Ex: P64 directory contains all alltrials-BEB-P64-D1.txt, alltrials-LOG-P64-D1.txt,...,alltrials-BEB-P64-D2.txt, alltrials-LOG-P64-D2.txt...

In line 29 InputData.m, change path to access the correct directory.

Run this command: callPlot('P1024',10,10,10,5)

And the graphs should be plotted.

Notice that you may have to change the matlab files slightly to suit your simulations results files.

How to run the simulations with any number of clients?

simScript.bash is the script that runs the simulations, you could find on it a line like this:

```
./waf --run "scratch/nonQos --numSta=$NUMCLIENTS --numClients=$NUMCLIENTS --myRngRun=$RNGRUN" > "$filename" 2>&1
```

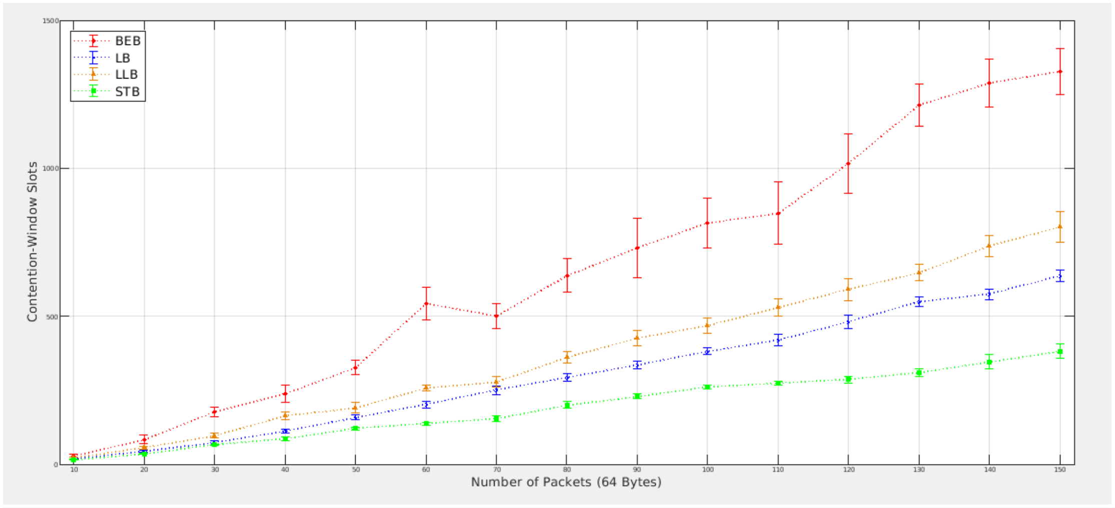
Where NUMCLIENTS is the number of clients that the simulation will run with, so you could simply change the script to run for the number of clients that you want and to put the results where you want.

simScript300-1000.bash is an example of script that does that, it is the script the we used to run our simulations on BEB and our algorithm 3 with number of clients of 10, 20, …, 120, 300, 500, 1000.

Our simulations results:

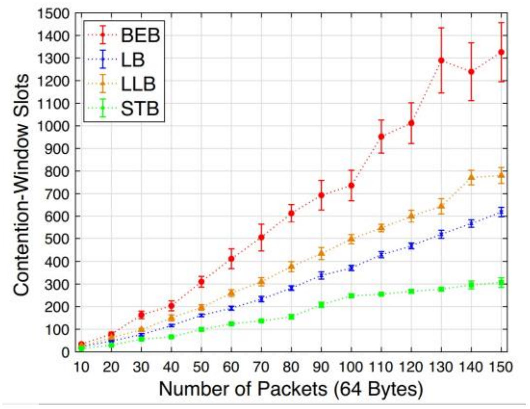
In our simulations using NS3, we test four methods: BEB, LB, LLB, and STB. We have two types of simulations: one for CW slots and one for execution time. We vary the number of packets from 10 to 150, running 30 simulations for each packet count and averaging the results. This creates a graph where you can compare the algorithms' performance for both CW slots and execution time.

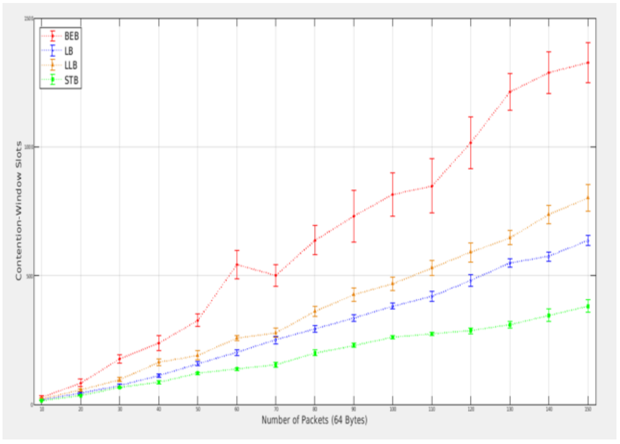
Our graph for CS slots



It's evident from the data that BEB consistently has a higher average number of CW slots across all packet counts compared to the other algorithms. The images at the bottom of the page represent both our own simulations and the simulations from the article. The comparison shows a strong similarity in behavior between the two sets of simulations, with BEB consistently performing worse in terms of CW slots. This alignment in results validates the findings presented in the article.

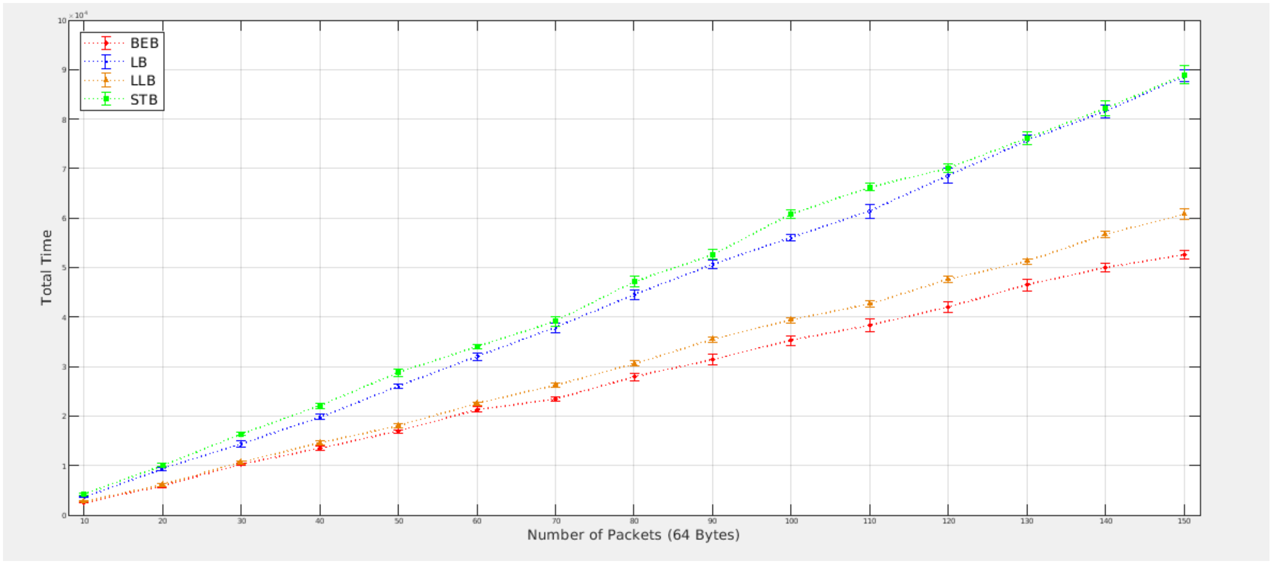
CS slots comparison: our graph on the right and the article's on the left





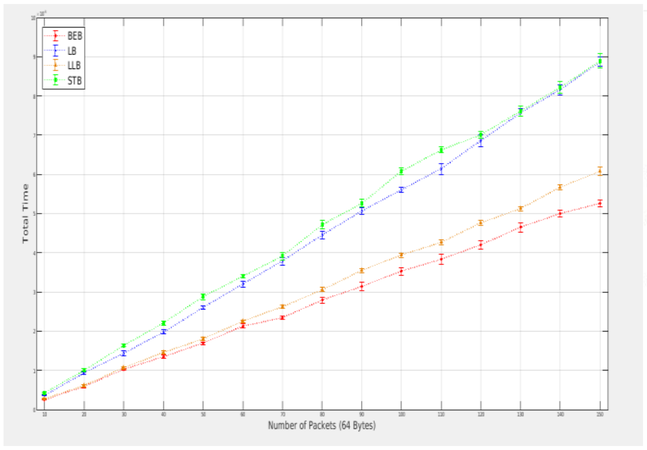
In the total time simulations depicted in the images at the bottom, it's evident that BEB consistently performs the best among the algorithms. This outcome reinforces BEB's advantage in terms of total execution time, as indicated in the simulations.

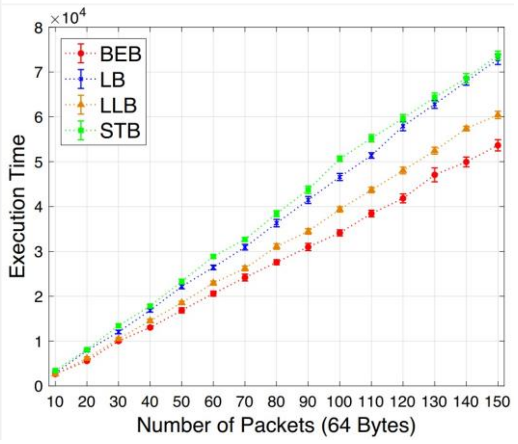
Our graph for total time



Once again, we observe a remarkable similarity between our own simulations and the simulations from the article

Total time comparison: our graph on the left and the article's on the right





Conclusions:

Experiments confirm theoretical predictions that LLB, LB, and STB outperform BEB with respect to CW slots.

In comparison to BEB, the execution time for LLB, LB, and STB is significantly worse.

For algorithm design, optimizing CW slots at the expense of increased collisions is a poor design choice.

Backing off slowly is bad.

New Algorithm Proposals:

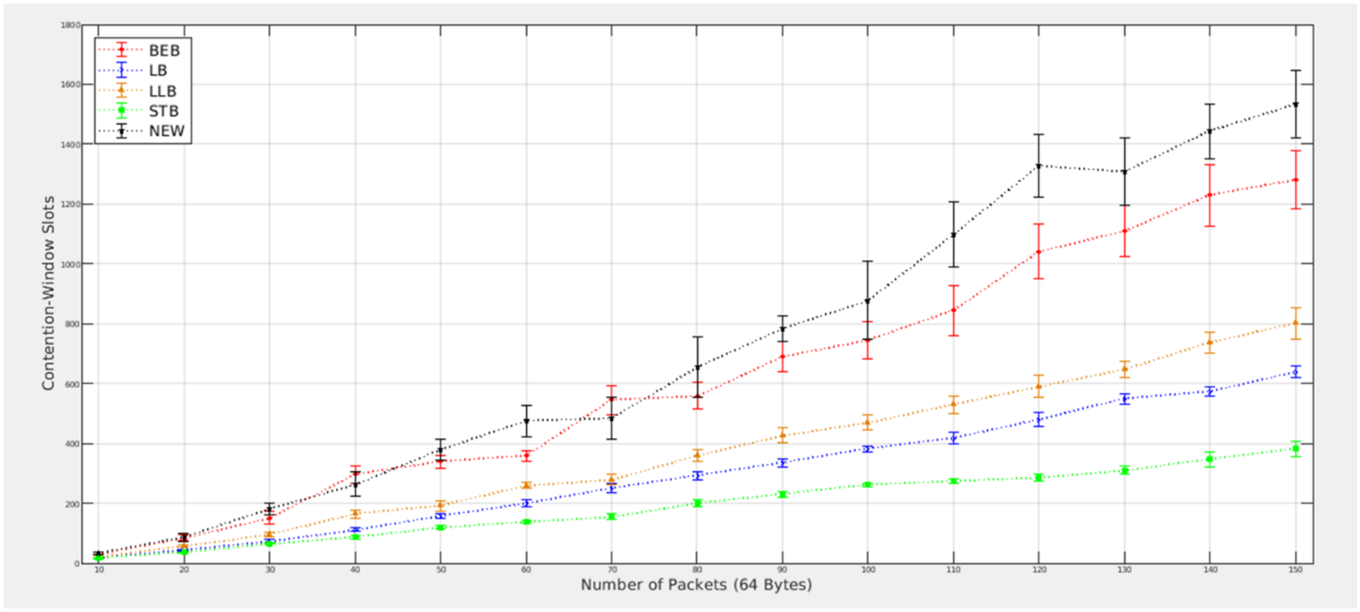
In this section, we introduce novel algorithm concepts that we intend to assess and compare against the algorithms outlined in the article (BEB, LB, LLB, and STB).

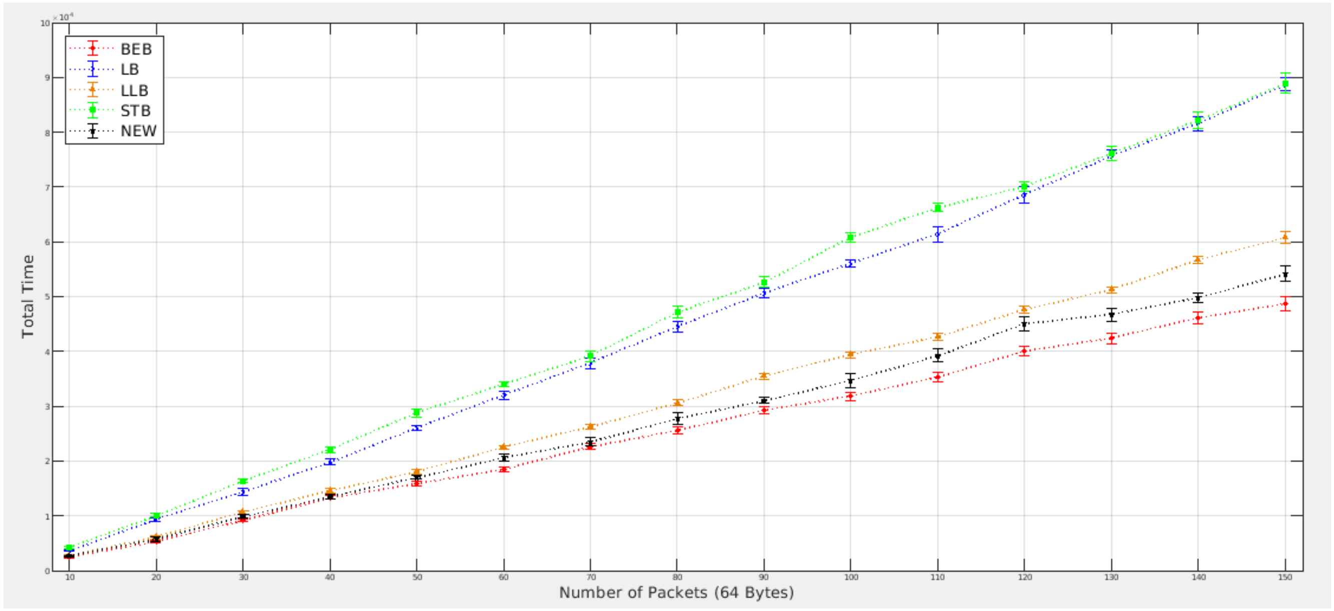
Algorithm 1:

Based on the insights gained from our simulations, we recognize that BEB demonstrates excellent performance, as it is the best-performing algorithm highlighted in the article. Consequently, our initial inquiry focuses on exploring the implications of introducing a slight modification to BEB. Specifically, we examine the scenario where, after each retransmission, the contention window (cw) size is updated by a factor 'f' that is not equal to 2. This adjustment essentially involves testing BEB with a different multiplicative factor. We undertake this exploration to gain a deeper understanding of how variations in the rate of cw size change (whether rapid or gradual) impact the overall execution time.

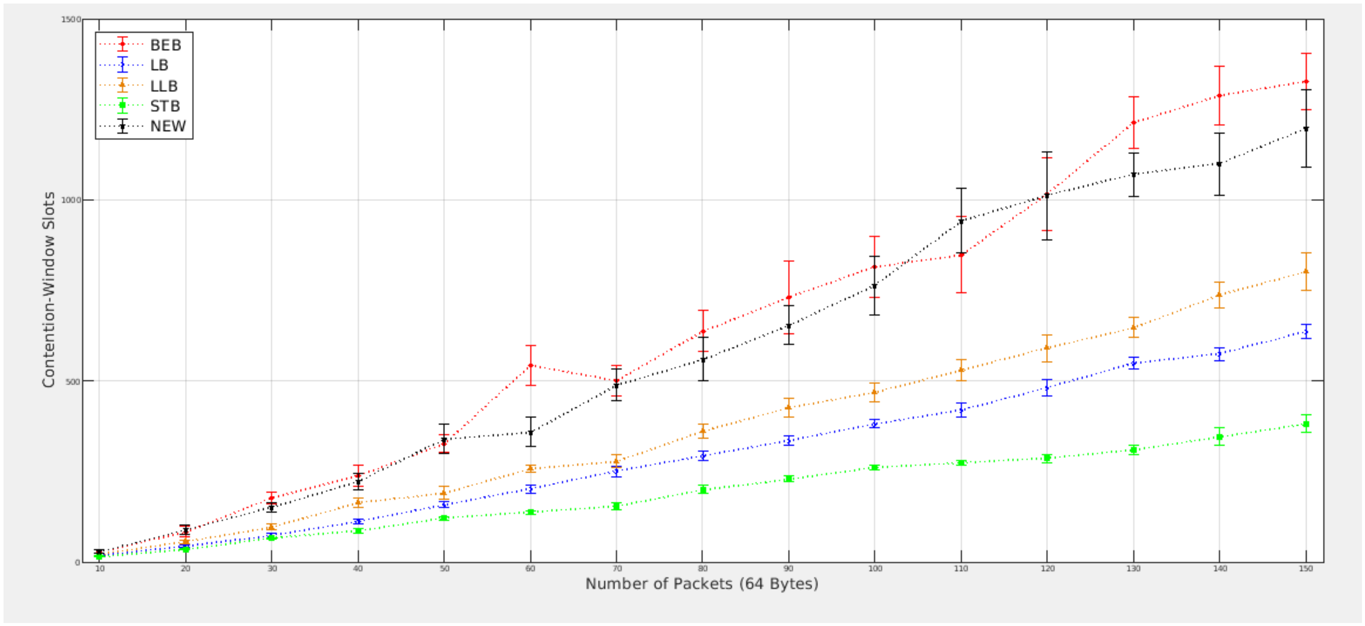
Algorithm 1 simulations results: We conducted numerous simulations employing different 'factor' values, and we present the results of a couple of these simulations as illustrative examples:

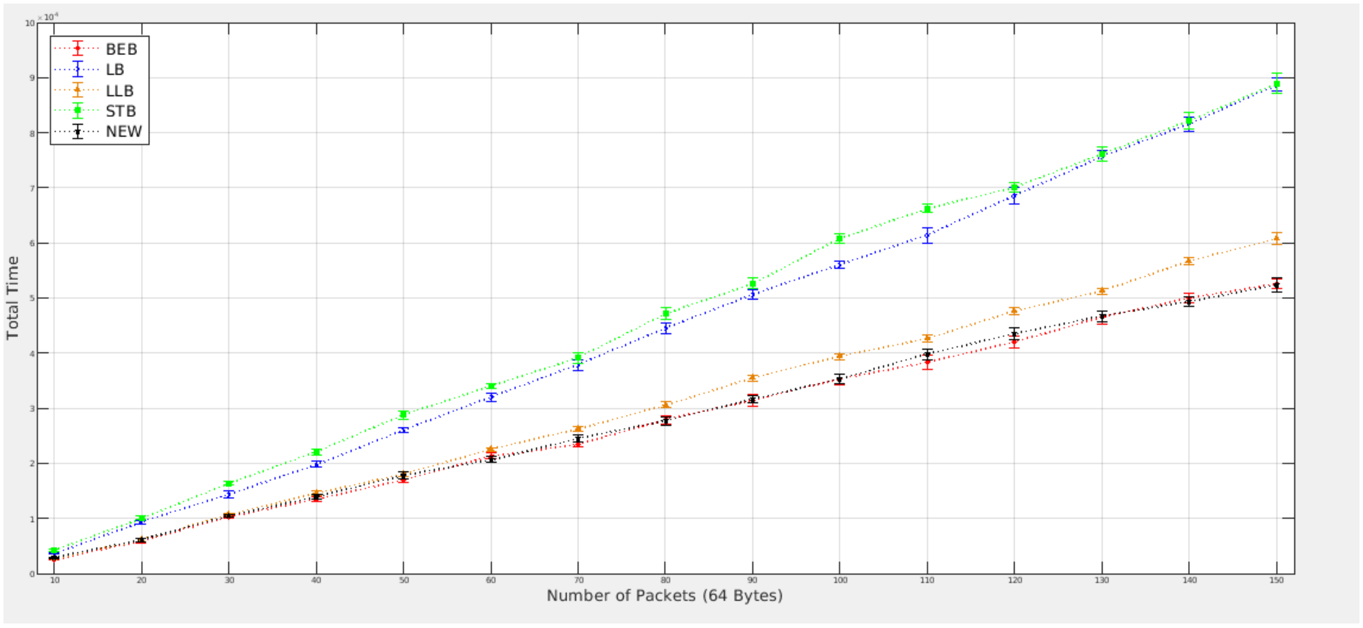
A constant factor of 2.1:





A constant factor of 1.9:





Conclusions on algorithm 1 results:

Our simulations indicate that, in the majority of cases involving varying numbers of clients, the conventional BEB algorithm with a factor of 2 outperforms other configurations. While there are instances where an alternative BEB configuration with a different factor yields slightly shorter execution times for certain client counts, it is essential to note that, on the whole, BEB with a factor of 2 consistently exhibits superior performance when compared to the various factors we examined.

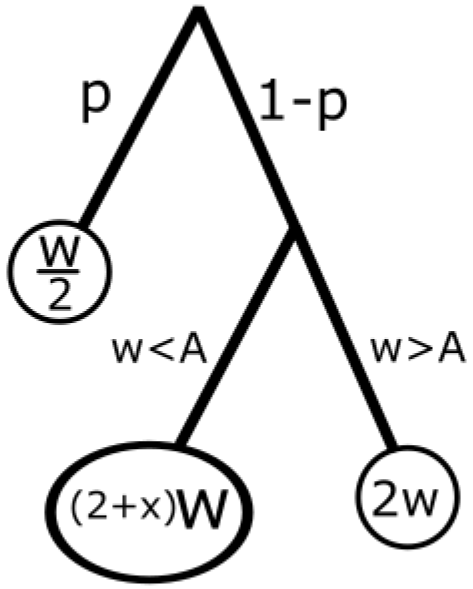
Algorithm 2

When we talked about STB earlier, we explored the idea of using a smaller window size after a larger one. The thought is that a larger window may successfully send some packets, potentially reducing the need for a larger window for the remaining packets. To enhance BEB, we took a different approach. Instead of using two loops, we created a random algorithm.

In this algorithm, we introduced two parameters: 'p' and 'A'. 'p' varies between 0 and 1 and determines whether the window size should be divided by 2 (when 'p' is small) or increased (when 'p' is closer to 1). 'A' is a value between the minimum and maximum window sizes. If the current window size is smaller than 'A', we increase it by a factor of 2 + x, and if it's larger, we simply double it.

This randomization aims to find a balance between enlarging and reducing the window size based on the situation, with the goal of optimizing transmission performance.

Algorithm 2 graph illustration

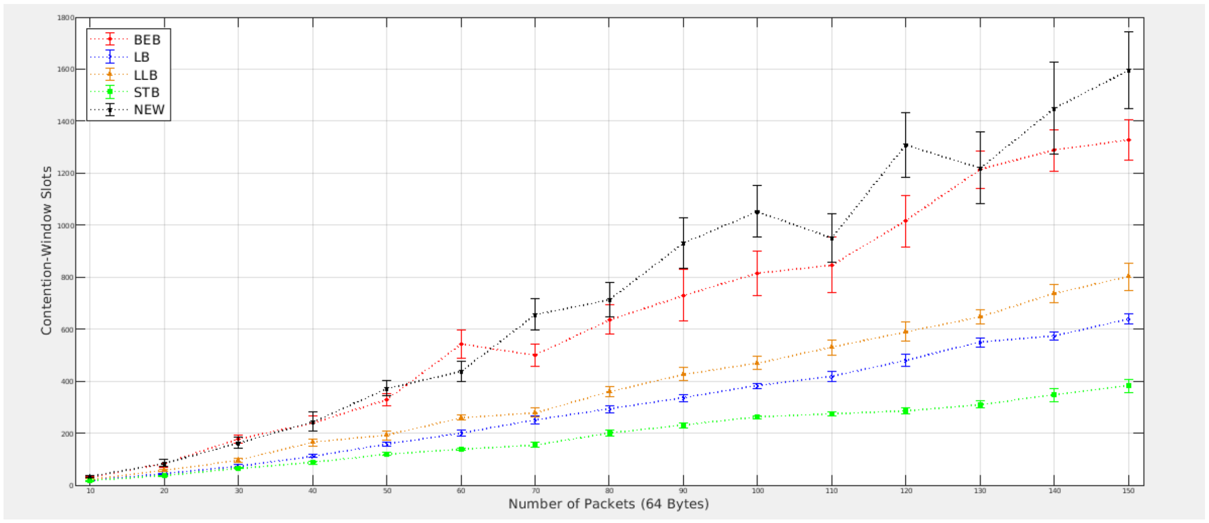


The idea behind this algorithm is to combine the features of STB, LB, and LLB. By doing so, we aim to leverage STB's concept. Additionally, we incorporate the behavior of LB and LLB, which use larger factors for smaller window sizes and smaller factors for larger window sizes. This combination helps us optimize the window size selection, considering both current conditions and past results.

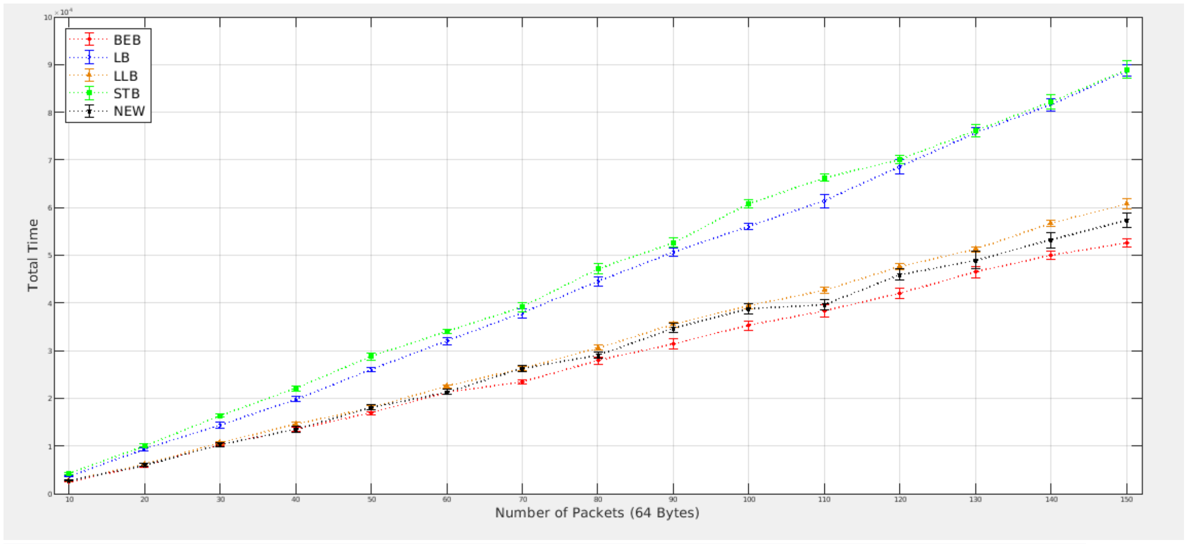
We did one simulation to test this , we chose A = max_window_size / 8 , p = 1/5 , x = 0.8

Algorithm 2 simulations results:

CW slots graph with algorithm 2



total time graph with algorithm 2



As observed, this algorithm appears to perform worse than BEB in many cases, and for various packet amounts.

Algorithm 3:

In Algorithm 3, our goal is to develop a backoff algorithm that amalgamates elements from LB, LLB, and BEB. We aim to create a backoff mechanism where the contention window size smoothly adjusts depending on its current magnitude. When the window is relatively small, we want it to expand more rapidly, whereas if the window is larger, the growth rate should be more gradual. However, taking into account that LB and LLB didn't yield satisfactory results in our previous experiments, our objective is to align our algorithm more closely with the performance characteristics of the BEB method.

To achieve this, we introduce a parameter termed "ratio," which calculates as follows:

$$\text{ratio} = \frac{\text{current\_cw} - \text{min\_cw}}{\text{max\_cw} - \text{min\_cw}}$$

We then transform this ratio using a function that incorporates two factors, alpha (α) and a value we denote as "transformedRatio." The transformed ratio calculation is expressed as:

$$\text{transformedRatio} = 10^{-\alpha * ratio} * (1 - ratio) + ratio$$

We utilize this transformed ratio to determine a factor that will govern the adjustment of the contention window size. This factor is computed as:

$$\text{Factor} = 2 - x + \text{transformedRatio} * 2x$$

Ultimately, the new contention window size (New_cw_size) is determined by multiplying the old contention window size (old_cw_size) by this factor.

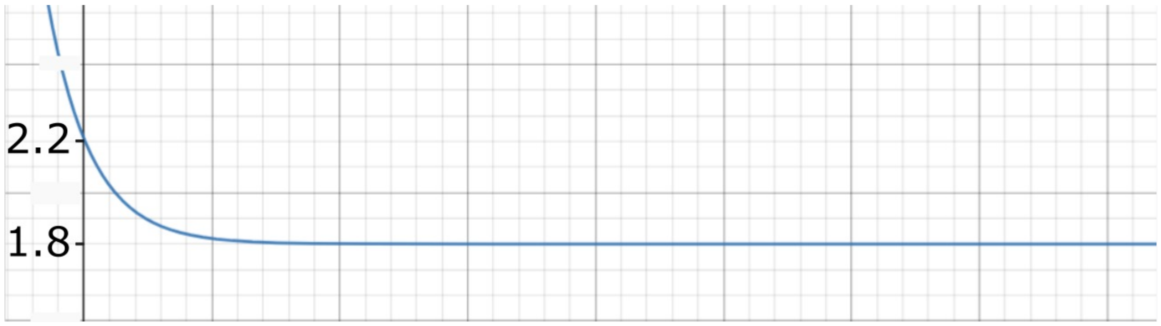
This methodology allows us to obtain the desired behavior in relation to the contention window size while maintaining a close alignment with the value of 2. The factor exhibits varying behavior as the ratio changes, ensuring a dynamic adjustment of the contention window size.

Algorithm 3 - Implementation:

We exemplify the implementation of Algorithm 3 through code. In this code, we have defined parameters α (alpha) and x as 10 and 0.2, respectively, to illustrate the behavior of the factor.

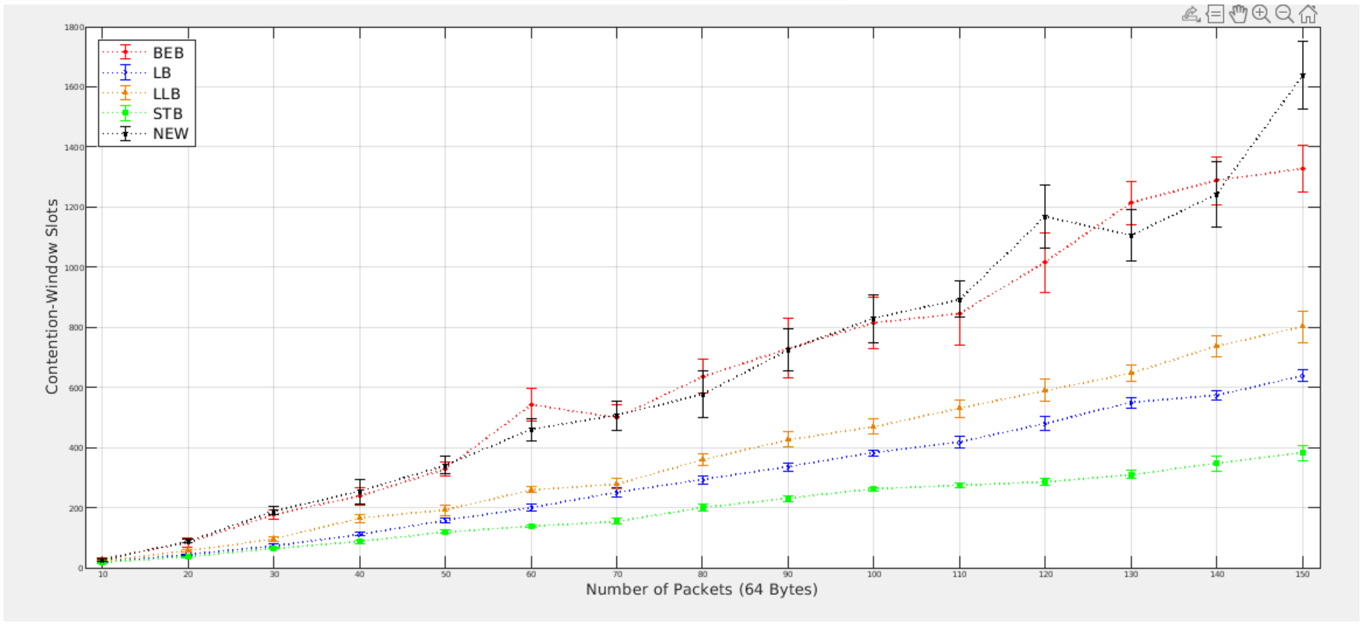
```
if(m_backoffType == 4){//our backoff
      double ratio = (double)(m_cw - m_cwMin) / (m_cwMax - m_cwMin);
      double transformedRatio = std::pow(10.0, -10.0 * ratio) * (1.0 - ratio) + ratio;
      double result = 1.8 + transformedRatio * (2.2 - 1.8);
      m_cw = std::min ( (uint32_t)(result * m_cw + 1), m_cwMax);
```

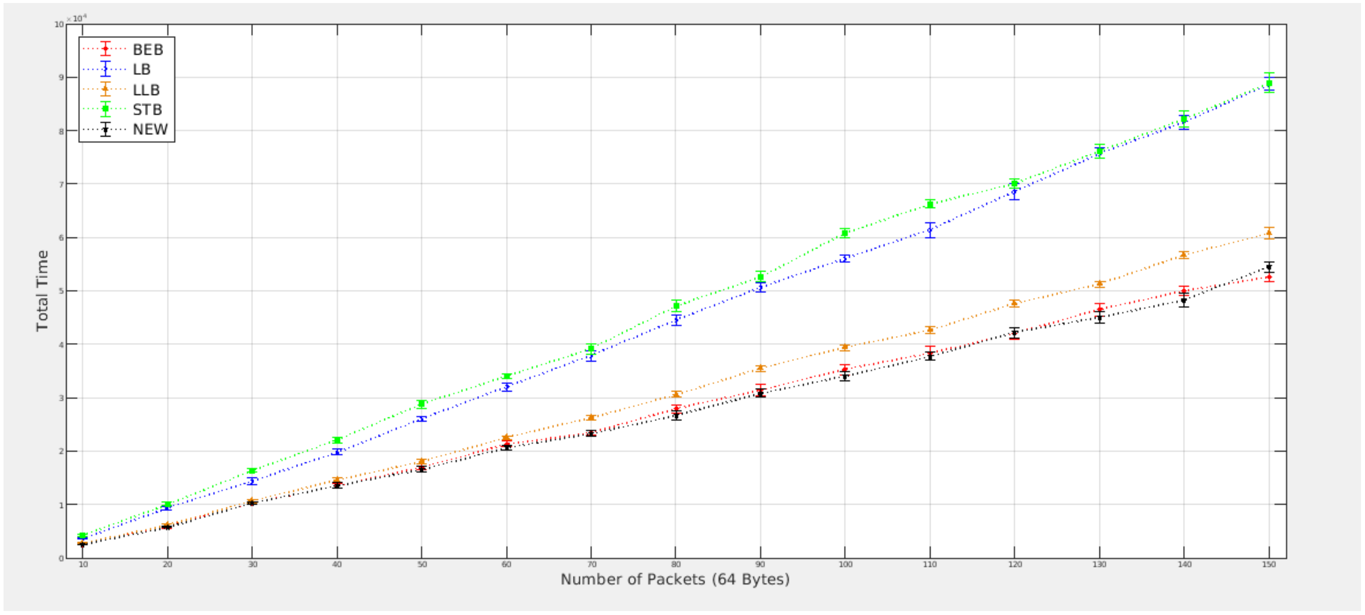
This factor's behavior is graphically depicted, showing how it changes concerning the ratio parameter and ultimately influences the contention window size.



Algorithm 3 simulations results:

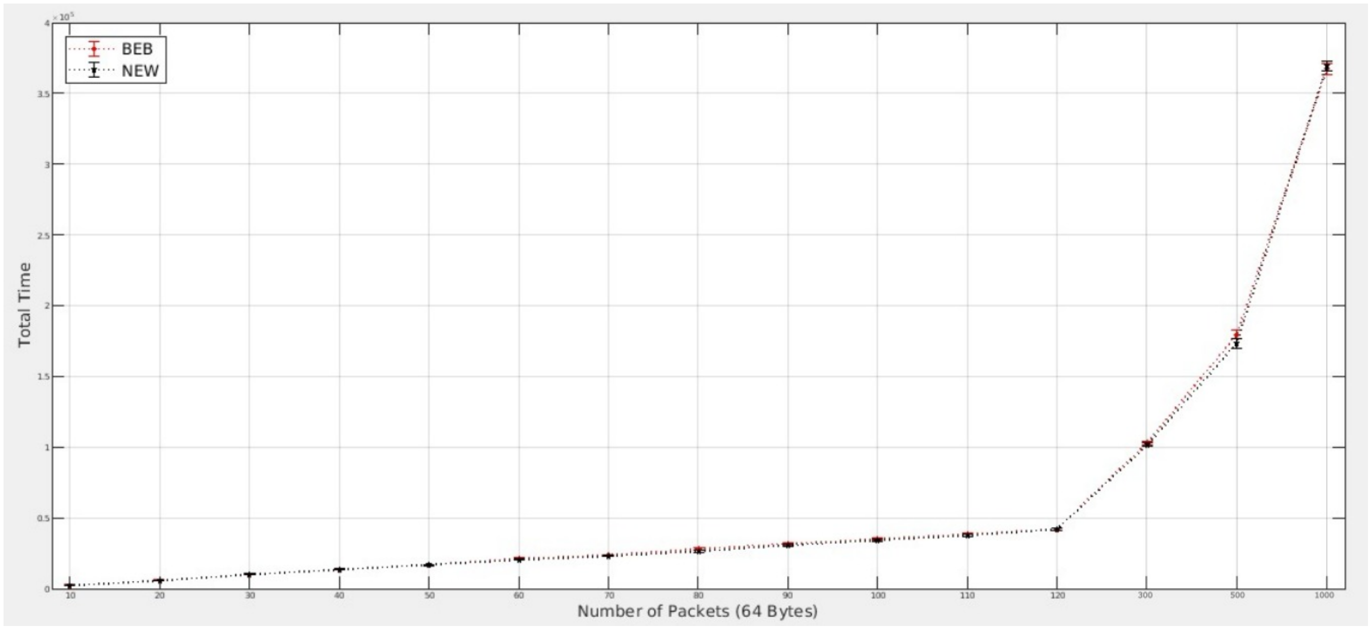
Here are the outcomes of our simulations for Algorithm 3, where we set the values of α and x to 10 and 0.2, respectively:





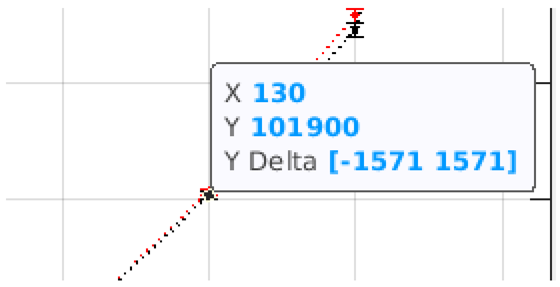
Upon analyzing the results, it becomes evident that Algorithm 3 exhibits a slightly superior performance compared to BEB across various numbers of clients. However, this difference is not substantial. Notably, we observed an interesting trend: when the number of clients reaches 150, BEB outperforms our algorithm. This prompted us to conduct additional simulations, extending our comparison between BEB and Algorithm 3 to scenarios involving 300, 500, and 1000 clients. We aimed to determine if our algorithm's better performance holds primarily for smaller client numbers.

The subsequent simulations encompassed client counts of 10, 20, 30, and so on up to 120, as well as the higher counts of 300, 500, and 1000:

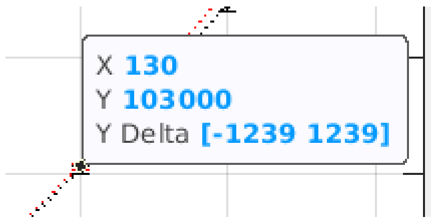


As we see it is not clear from the graph which algorithm has a better performance so we decided to check the numerical values of the total time that is in the graph. in those pictures the Y value is the total execution time in microseconds.

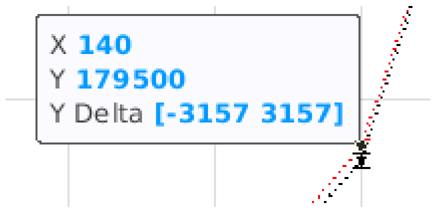
Total time for BEB with 300 clients:



Total time for algorithm 3 with 300 clients:

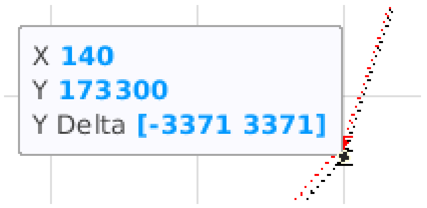


Total time for BEB with 500 clients:



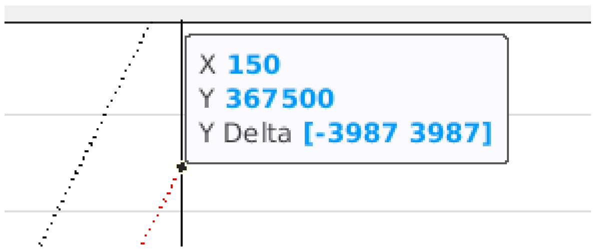
X 140
Y 179500
Y Delta [-3157 3157]

Total time for algorithm 3 with 500 clients:



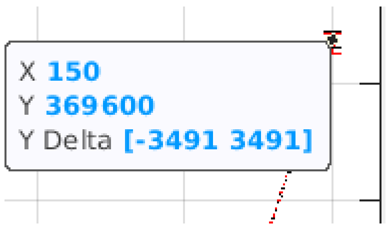
X 140
Y 173300
Y Delta [-3371 3371]

Total time for BEB with 1000 clients:



X 150
Y 367500
Y Delta [-3987 3987]

Total time for algorithm 3 with 1000 clients:



X 150
Y 369600
Y Delta [-3491 3491]

Upon examining the simulation results, we observe that Algorithm 3 outperforms BEB when the number of clients is set at 300 and 500. Conversely, for scenarios involving 1000 clients, BEB demonstrates superior performance compared to our algorithm. This suggests that, in cases with a higher number of clients, the superiority between BEB and Algorithm 3 may fluctuate; sometimes BEB is more efficient, and at other times, Algorithm 3 excels. However, it is noteworthy that even in cases where BEB outperforms Algorithm 3, the margin of difference in total execution time remains relatively small.

Conclusions:

Algorithm 3's performance is somewhat superior to BEB, though the difference is not substantial. Therefore, it's likely more practical to opt for BEB due to its much simpler implementation.

What we have learned from the project and final conclusions:

-we learned a new backoff algorithms like LB,LLB,STB

-We learned how to use the ns3 simulator

-backing of slowly in backoff algorithms is bad

-For algorithm design, optimizing CW slots at the expense of increased collisions is a poor design choice.

References:

https://gitlab.cs.technion.ac.il/tavran/sp2023_beb_on_burst_traffic

https://github.com/trishac97/NS3-802.11g-Backoff/tree/main

https://www.nsnam.org/wiki/HOWTO_build_old_versions_of_ns-3_on_newer_compilers

https://www.nsnam.org/docs/release/3.25/tutorial/html/getting-started.html

https://www.nsnam.org/docs/release/3.25/tutorial/ns-3-tutorial.pdf