Time to submission: 14 days

Instructions: Write the simulation codes in Python, C++ or MATLAB. You need to submit a write-up (word/latex) where you discuss your approach, explain different parts of the codes (and logic behind them) and report your findings from the simulations. At the end of the report you must also add the code and instructions to run that, so that we can run it if needed.

Recall TCP. Here, we shall simulate a simplified version that has the essence of TCP. Using that we shall try to understand the interplay between schedulers like max-weight and TCP. As I had said in the class, analysis of that interplay is often intractable and hence, simulations (and real experiments) are the only way forward.

In these problems, we focus on that side of TCP which deal with timeouts and not dupACKs, since simulating a scenario that results in dupACKs would involve multi-hops and may be a little hard on you. That does not mean that those scenarios are not important.

Q. 1 [8 points] **Interplay between the MAC layer and a TCP like transport layer**

Consider K discrete-time Bernoulli flow of packets arriving at a buffer which is served by a single server whose service capacity is Binomial($N, \frac{K}{N}$), where $N > K$. This means that if $S(t)$ is the realization of Binomial($N, \frac{K}{N}$) at time t, the server can serve $S_i(t)$ number of jobs from flow i while satisfying $\sum_i S_i(t) \leq S(t)$.

Out of K flows, R flows have Bernoulli p arrivals. Remaining flows are also independent Bernoulli, but their rates change in the following way.

Each of the remaining $K - R$ flows maintain average delay in the following way. For flow i, if the average delay up to packet n is $T_n^{(i)}$ and the delay (from arrival to service) of the $n+1$th packet is $D_{n+1}^{(i)}$ then the average delay is updated as $T_{n+1}^{(i)} = 0.9\ T_n^{(i)} + 0.1\ D_{n+1}^{(i)}$.

Suppose the arrival rate for flow i at time t is $p^{(i)}(t)$ and $n^{(i)}(t)$ be the number of packets served so far. If at time t there is at least one un-served packet that arrived on or before $t - \lceil 1.2\ T_{n(t)}^{(i)} \rceil$, update $p^{(i)}(t+1) = \frac{2p^{(i)}(t)}{3}$. If at time t, r packets are served and all of them arrived after $t - \lceil 1.2\ T_{n(t)}^{(i)} \rceil$, then update $p^{(i)}(t+1) = \max(0.9, p^{(i)}(t) + \delta)$ for $0 < \delta < 1$.

Generate an arrival in slot $t+1$ for this flow according to Bernoulli $p^{(i)}(t+1)$. Consider the following service policies.

Processor sharing: If $S(t) > l\ K$ and $S(t) \leq (l+1)\ K$ for some integer $l \geq 0$, then serve l packets from each flow first. Then, serve one additional packet from flows $1, 2, \ldots S(t) - l\ K$.

Water-filling or minimize the max-queue: Let flow i have $Q_i(t)$ number of packets at time t. Serve the flows in such a way that at $t+1$, the gap $\max_i Q_i(t+1) - \min_i Q_i(t+1)$ is minimized. (Think: there is an iterative way of doing it.)

Max-weight scheduling: Check class notes.

- In your report, write a short criticism of this caricature TCP model.
- Pick any of the $K - R$ TCP flows. Compare its throughput (define it) and average delay under the three scheduling policies described above.
- Also compare the server utilization, i.e., the number of packets really served at t divided by $S(t)$. (Take a time average of that fraction.)
- Try different values of p. Pay close attention to p close to 1 and p close to 0.
- Try out the following (K, R) pairs: $(5, 1)$, $(20, 1)$, $(5, 4)$, $(20, 15)$, $(50, 1)$ and $(50, 40)$.
- Try $N = \lceil 1.1K \rceil$, $N = \lceil 1.5K \rceil$ and $N = \lceil 3K \rceil$ (note : ceil function)

Q. 2 [4 points] **Finite buffer** Repeat the previous problem, but with the change that the buffer size is limited. Here, any packet arriving to a full buffer will be dropped. Though a dropped packet will not contribute to the update of $T_n^{(i)}$, but it will contribute to the number of un-served packets that arrived on or before $t - \lceil 1.2\ T_{n(t)}^{(i)} \rceil$.

Simulate for different buffer sizes $\alpha \frac{Kp^2}{1-p}$ for $\alpha = 1.2$, 1.5 and 3. (The formula $\frac{Kp^2}{1-p}$ is for the average queue-length in the infinite buffer case. It comes from Kingman's bound and Little's theorem, in case you are interested to know. We are trying to explore, how big a buffer should be.)

Q. 3 [4 points] If you do the above simulations carefully, you will make some interesting observations. You will be given marks for pursuing one or two exciting directions that the observations lead to. This problem is kept open ended intentionally so that you can get a flavor of research, which I think I had promised. (Note that the overall direction of this assignment is not so well explored. So, if you pursue carefully, you may even get a paper out of it!)

EE5150 Communication Networks ProgAssign 1
Instructor - Avishek Chatterjee

Simulating UDP and TCP links at a server

Pranav Phatak - EE19B105

April 25, 2022

Contents

1 Possible Algorithms for implementation

In the assignment, a couple of algorithms were mentioned. I tried out both, but finally decided to go ahead with some kind of hybrid of both of these. The algorithms mentioned were - Processor Sharing, Max Weight Scheduling.

In Processor Sharing, if the server capacity is n, and number of links is K. then first take $n\%k$, and service these many packets from each link, and then serve 1 each from remaining links (random choice of links) till capacity of the server is exhausted or all queues are empty. In Max Weight Scheduling, the server should check all queue lengths, then service 1 from the largest queue, and keep doing this till its capacity is exhausted or all queues are empty.

I felt we could do something different, the idea of Processor Sharing is right, that the server should first service equally from all links, however instead of randomly serving the remaining, the server can then switch to a Max Weight Scheduling type of algorithm. In this approach, the server is not giving extra priority to any link initially (so it is a fair server), however once it has less than K capacity left, then it cannot be fair to all links in this manner. So now the server looks to optimize the queues on its incoming links by trying to decrease all the large queues, so that these packets also reach their destination without too much unnecessary extra delay. In case that 2 queues have equal packets waiting, then the server will service the queue which has a higher average link delay.

This seems like a fair way of serving the queues, since all links are given equal weightage initially, and the server also looks to then finally make sure that no link has too many packets waiting and that all the links have more or less similar average link delay. In my code submission, I have included the C++ codes that I've written for this implementation using infinite and finite buffer (2 slightly different codes). The 2 codes take no user inputs, the K,R,N,p values can be changed inside the code, these have been declared at the start of the main. I have commented the codes, but if there is some discrepancy please tell me I will try to solve it.

2 General Observations

For the given examples, the server is a binomial distribution of N with probability $\frac{K}{N}$ for which, the server capacity comes very close to K for the 3 values of N given by $\lceil 1.1 * K \rceil$, $\lceil 1.5 * K \rceil$ and $\lceil 3 * K \rceil$ (i.e it has maximum probability in the range of K-2 to K+2). Following are some binomial distributions (all the 3 cases for K=5) with total 10000 trials.

Our Bernoulli arrival rate p is fixed for UDP links whereas adaptive for TCP links. It is also pretty clear that at each moment, a maximum of K packets arrive (1 at each link). Therefore, it looks like we have set up a good problem, since the server capacity and amount of packets arriving is random and in the same range. The goal is to test out different algorithms and see how the system reacts to it in terms of throughput, delay and server utility. In further analysis, I have used the case of K=20,R=15 predominantly and varied the values of arrival rate p and binomial distribution coefficient N.

```
0: 0
1: 9
2: 62
3: 534
4: 2070
5: 4038
6: 3287
```

```
0: 6
1: 48
2: 300
3: 1057
4: 2111
5: 2809
6: 2295
7: 1143
8: 231
```

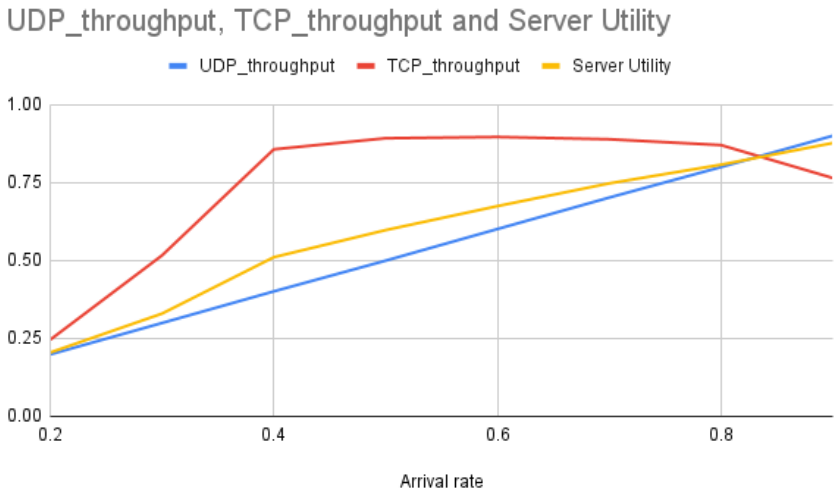
```
0: 23
1: 169
2: 614
3: 1214
4: 1913
5: 2179
6: 1880
7: 1125
8: 570
9: 228
10: 66
11: 14
12: 3
13: 2
14: 0
15: 0
```

3 Infinite Buffer Case

Since buffer size is infinite, at a certain queue, any number of packets can lineup one behind the other if not serviced. Let us analyse by keeping the server capacity same and variating the initial arrival rate for all links.

3.1 Fixing the server capacity and variating arrival rates

It is very clear that server utility will be best when more packets arrive, and hence it rises with the bernoulli rate p in all cases.Similarly, the UDP rate is also fixed, and so the UDP throughput will be close to the arrival rate p, with some average link delay, which will be almost 0 for very small values of arrival rate p, and will increase with p, since higher the rate, more the chances of a delay (we are assuming that server capacity remains same but arrival rate changes)



The above plot is for the case K=20,R=15,N=60.

The above diagram shows how the UDP and TCP throughput vary w.r.t arrival rate p. It also plots the server utility which is the amount it served upon the amount it could serve with full capacity (since actually served = min(total packets available, max serve capacity)).
Clearly, the server utility increases monotonously, since more the packets come, more the server can use itself to its fullest.

The interesting curve amongst these to look at is the TCP throughput curve. Since in the TCP links, we increase the arrival rate p by some delta and take it to a maximum of 0.9, the TCP throughput is more than UDP throughputs, almost everytime till p becomes close to 0.9. What happens near that value is that, since there are some UDP links which are flooding their queues with 0.9 rate, the server is stretched. This eventually causes backlogs on some links, and a backlog on a UDP link won't decrease its rate, however a backlog if it occurs in the TCP link, where packets arrived before $\lceil 1.2 * T \rceil$ is counted as a backlog, leads to a fall in the arrival rate, by changing it to $\frac{2 * p}{3}$ and if even at the next instant, a packet arrives and server capacity limits the server to not serve this packet, a backlog will be recorded at this next moment too, making the Bernoulli arrival rate even smaller.

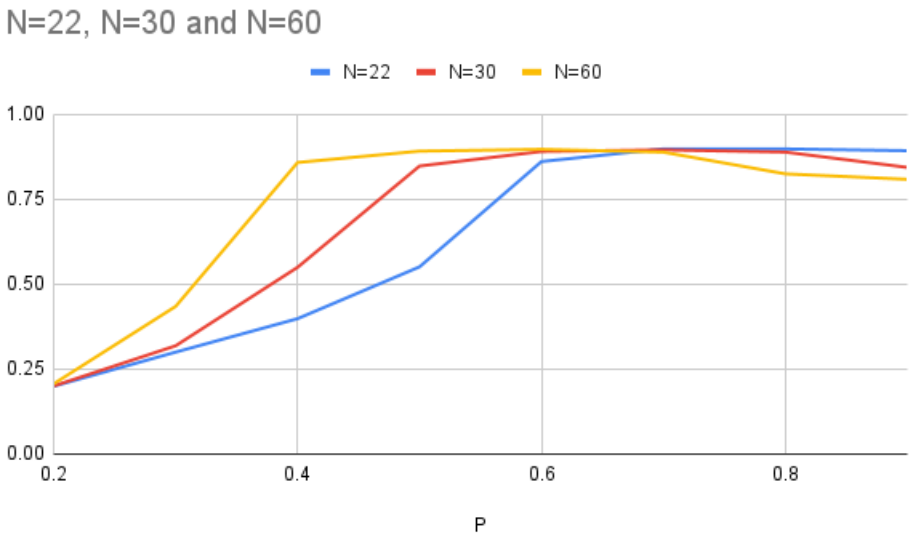
Because of this, the TCP throughput will fall slightly below 0.9 especially in the case when initial probability was close to 0.9 (since UDP links with higher rate will make sure the server is almost always busy and make it tough for the server to clear any backlogs till the number packets arrived at all links at some instant is less than K).

Observation :
If at some moment less than K packets arrive, then due to our good choice of binomial distribution for server, where server capacity is on an average K, this backlog will get cleared out, but for this to happen, the rate at that link will have to fall down sufficiently. Therefore, if a backlog is created, then it will decrease the effective throughput of that link by a noticeable amount (when iterated w.r.t time for around 10^5) .

Clearly, the TCP link is the one interesting thing, and so let us try to analyse this more, now we shall how TCP throughput variates with change in the server binomial distribution.

3.2 Changing the server binomial distribution too and seeing different TCP curves

As we increase N, the distribution becomes slightly more wider with the mean still at K, because of this, server capacity is different from K with more probability, as one can notice from the first image showing different binomial distributions. Therefore, when the initial arrival rate is low, even if capacity comes at K-1 or K-2 and so on, there's a good chance all links are served, and therefore the arrival rates for TCP will increase, however with equal probability, server capacity also becomes K+1 or K+2 and so on, and hence it will cut off the backlog sooner if it is there and won't let the TCP arrival rates fall by much, and hence as N increases, the TCP throughput will increase towards 0.9 faster.



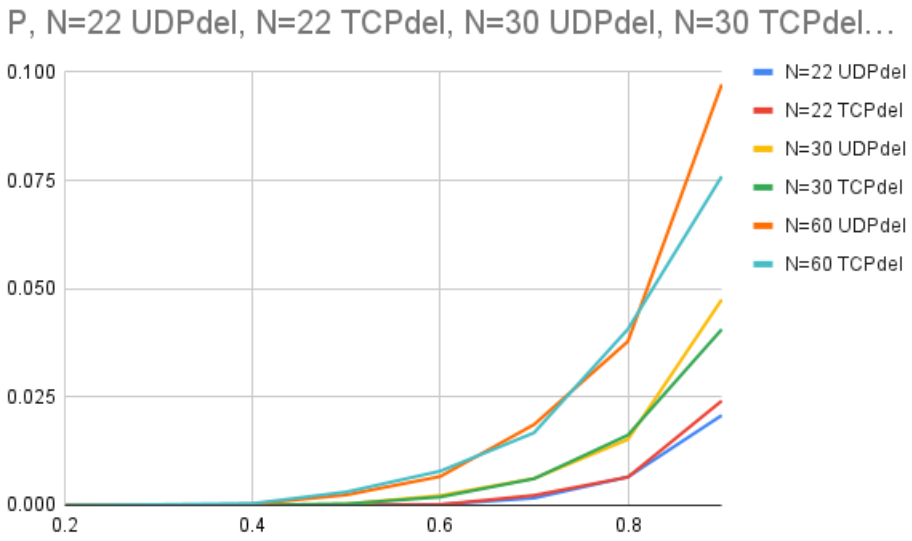
The above is done for K=20,R=15.

However, as we get close to initial arrival rate of 0.9, when the server capacity is K-1 or K-2 and so on, it will cause just as much backlogs as it clears backlogs when the server capacity is more than K and since the arrival rate falls exponentially, whereas it increases linearly, the final throughput will be lesser for higher N values. This same trend is observed in the graph above, which makes good sense.

By now, we have a good idea of how UDP and TCP links work when they are routed through the same server, next we shall look at how the average link delays.

4 Average Link delays as p and N are varied

We know that average link delay will depend upon how many packets were delivered late. This number will be large if the arrivals are more since it increases the chance that some packets were not delivered when it arrived when the server capacity turns out less than K. Also, as N increases, the binomial distribution spread becomes larger, thus it will give server capacity less than K more times than a smaller N would. Any time a packet is not sent the moment it arrived, it adds to the average link delay. Thus, we can note down that average link delay will increase with arrival rate p and the binomial distribution coefficient N.



The above is done for K=20,R=15.

5 Finite Buffer Case

Next we shall take a look at the case when a finite buffer size is used at the links. This shall not affect the average link delay analysis or the UDP throughput since these are pretty independant of buffer size, the main thing to look at in that case will be the packet losses for UDP, TCP and if it can be controlled and if they are related to the throughputs in some way.

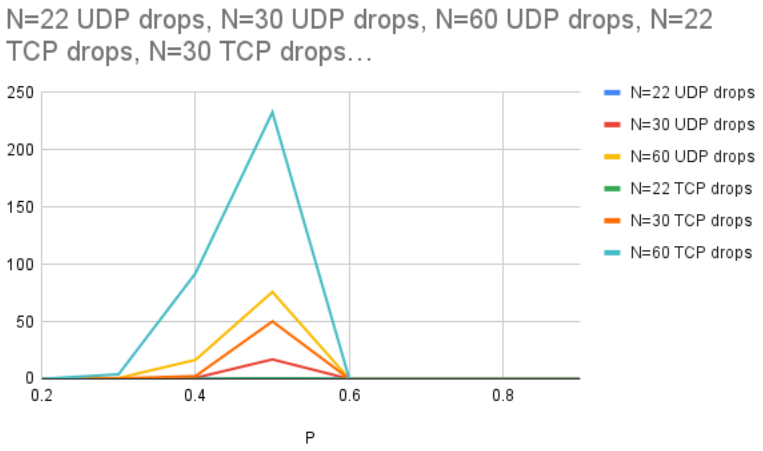
5.1 UDP vs TCP packet losses

In real life scenarios, we will never have infinite buffer (or an extremely large compared to average required memory) since it is a waste of resources, and therefore we will look to find some middle ground. The expectation value of a link with arrival rate p is given by $\frac{p^2}{1-p}$, hence we can multiply this by some factor as a safety net and set that as the buffer size per link. We shall do this experiment by keeping this factor as alpha and change it over a few values and see how the throughput is in these cases and how many packets are lost.

For the range in which we are varying the initial arrival rate, this average queue length $\frac{p^2}{1-p}$ takes the values $0.05, 0.128571, 0.266667, 0.5, 0.9, 1.63333, 3.2, 8.1$ for p from 0.2 to 0.9 respectively, we shall take buffer size as the ceil function of alpha times the arrival rate. (In my codes I have added 1 to this, since the packet in front of the server which is about to be served is not part of the buffer, packets behind it are in the buffer length and my queue length includes the first packet too).

An important observation :
A larger initial arrival rate means we will keep a larger buffer, and if we have a larger buffer then almost no packets will be dropped. Since we are making a buffer that is suited well for the initial arrival rate for each link, we will not see many packet drops in the UDP links (its arrival rate is not adaptive) ; whereas in the TCP links, as it tries to get a better throughput, its arrival rate p increases and therefore it can end up sending more packets than can fit in the buffer assigned to that queue when the connection was made, ending up in more packet losses as compared to UDP.

However as soon as it detects a packet loss, the arrival rate will be decreased, hence the arrival rate will automatically find its optimum value, but there will be packet drops in this link periodically since the arrival rate will increase for a while, then decrease upon detecting loss and this process will be looped leading to finding an optimum throughput value by doing a trade-off with packet losses.



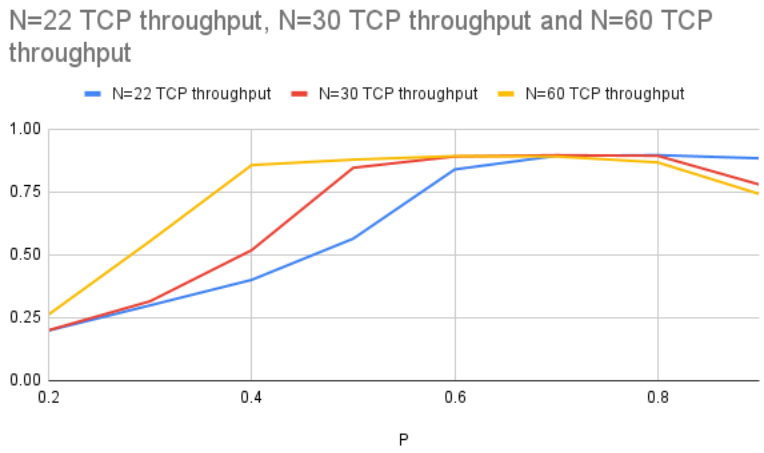
The figure above shows the number of packet drops per 10^5 time averaged over the specific link type.

As was stated earlier, the TCP links have more losses, since it tries to increase its arrival rate, whereas UDP link keeps the rate same. It should be however noted, that if we were to increase the arrival rate of any of the UDP links after making the buffer memory, then the UDP link will have a fixed amount of packet losses per time, and since UDP is not adaptive, it will leak off these many packets which will give heavy losses finally. Comparing this with TCP links, because of its adaptive nature, regardless of what arrival rate is given, it will always find an optimum between the buffer size and throughput in such a way that the best possible throughput is given by keeping packet losses to minimum. Also, as α the multiplying factor to the expectation value of queue length increases, the packet loss decreases, which is pretty straightforward since larger the buffer implies lesser the losses.

5.2 TCP throughput

Another observation which one can make looking at the graphs of TCP packet losses and TCP throughput as N and initial arrival rate are varied is that if TCP throughput reaches the maximum value at lower initial rate value, then the TCP packet loss is very high there, this is because at lower initial rate value the buffer size is low since $\frac{p^2}{1-p}$ is small, and it increases as p increases. Following is a graph of TCP throughput with

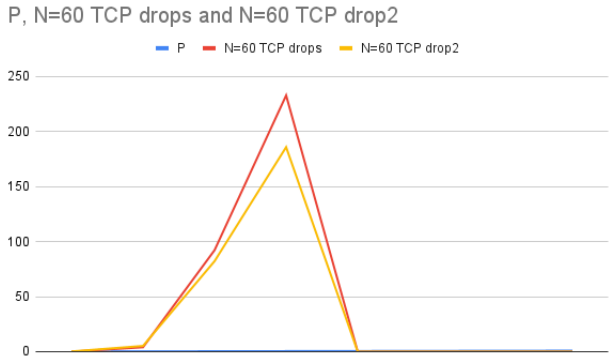
finite buffer for varied N and p.



Similar graph as for infinite buffer size is seen for finite buffer in TCP throughput, the difference is that in this case, the rate decreases by packet loss as well as backlog packets. If the rate peak occurs early, large overflow in buffer occurs which leads to huge packet losses. However, there could be ways to tweak the algorithm and the rate updating formulae to get better curves. I have mentioned one such possible attempt which I tried next.

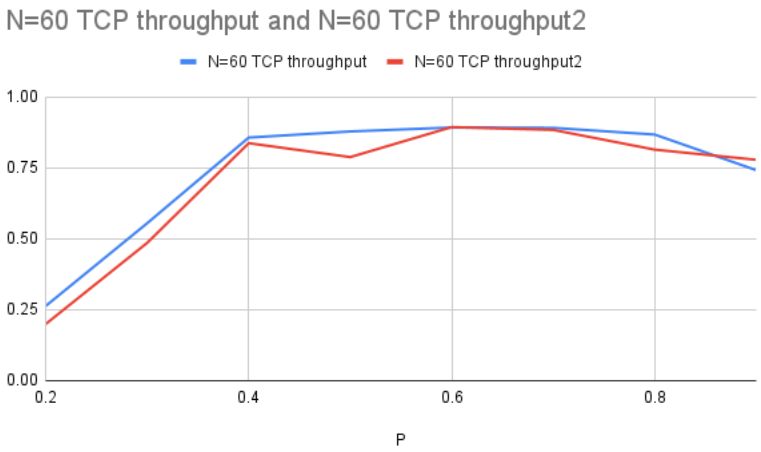
6 Decrease packet loss in TCP by slight change of algorithm

In our current algorithm, we change the rate p to $\frac{2*p}{3}$ when we get a backlog or loss, instead if we reset p to $\frac{p}{3}$ for a loss and the original value for backlog, then we get almost same throughput curve, and better curve in terms of packet loss. One can experiment a bit more and find the optimum on this value too. The following graphs show this specific example that I have mentioned.



The figure above shows the number of packet drops per 10^5 time averaged over the specific link type and the yellow line is the TCP packet loss for the changed formula used to update the rate.

The next graph compares the TCP throughput between the 2 distinct formula used to update the rate, which is more or less unchanged, except that there is a dip in throughput at the value of p where packet losses is maximum in both curves, but the amount of information lost is also less.



7 Conclusion

What we can see from the multiple experiments we tried so far is that coming up with many different algorithms is possible, and each algorithm might have its pros and cons, but overall most follow certain trends with arrival rate p, and N the binomial coefficient. One can however make tweaks as required by the system to get the best out of it in different ways.

Some important trends that were seen:

1)As p increases, the UDP throughput increases linearly, and so does server utility although it is not linear with p.

2)Server Utility is more if more packets arrive at queues, since the mean server capacity is K.

3)Average link delays depend on how many packets got delayed, and therefore will increase as the arrival rate p increases and also as N increases since increase in N gives wider spread of server capacity which implies total more packets with delay.

4)As p increases, the TCP throughput increases and touches a maximum of around 0.9 and falls down at the end due to some backlog due to all links being active at high p (0.9).

5)As N increases, the TCP throughput reaches the peak value of around 0.9 earlier, but falls down to a smaller value as p reaches a high value (0.9) due to larger spread of server capacity.

6)Packet losses are more in TCP once initial rate p is fixed since the buffer is then made for that p and UDP uses it whereas TCP tries to increase throughput by adapting (however, nothing can be done to control the losses in UDP link if the p is increased for those by keeping buffer size same).

7)As α increases, the packet losses decrease, since larger buffer implies less packet losses.

8)Packet loss curve can be brought in control for TCP by tweaking the rate updating formula.