Communication versus Storage Trade-offs in Data Distribution Systems

JonLuca De Caro

under the direction of
Dr. Muriel Médard
and
Dr. Viveck Cadambe
Massachusetts Institute of Technology

Research Science Institute
July 30, 2013

Abstract

In data distribution networks, such as those that power Netflix, there often exists a trade-off between local storage capacity and communication demands. In this paper, we devise an approach to characterize this trade-off. The approach assumes that the network can be modeled as a symmetric undirected graph, and aims to characterize the relationship between the storage and communication requirements by defining a function $C(\alpha)$ which measures the communication bandwidth needed for a given amount of local storage per node. We derive an analytical expression for $C(\alpha)$ for the case where the graph is a ring. A set of steps are also created to express the $C(\alpha)$ for any graph, using Dijkstra's shortest path algorithm.

Summary

Large networks that distribute content, such as Netflix and Hulu, are faced with a choice. They must chose whether to store all their information on every server in their network, or to store less information locally and rely more on communication network. The first example imposes a high storage requirement in exchange for low communication requirement — the server could send all the data to the requesting user without the need to access any extra servers, but Hulu would have to install servers with extremely large storage capacity. The second example is a low storage requirement in exchange for a high communication requirement. Hulu would be able to have comparatively small server capacity, but the user would have to connect to every server in Hulu's network to get the information. Understanding this trade-off is important to company budgets. It would be ridiculously expensive to store all their information on every server, so that's not feasible. On the other hand, if they chose to store the very least amount of information per server, that translates to a very high communication requirement, which might lead to slower access from the user, and they will receive complaints and lose their user base. A company such as Hulu must decide what the most optimal way to store their data is. We characterize this trade-off for simple symmetric graphs. We derive equations describing the minimum communication bandwidth consumption for a given storage capacity of the server. This is useful to anyone that has a network that distributes content.

1 Introduction

Consider an online movie provider such as Netflix. Netflix would like for all of its users to be able to access the movies they want at the lowest possible latency. Now imagine that they have servers all over the country that they have to store movies on. Netflix's entire movie catalogue is about 1 petabyte, or 1,000 terabytes [1]. The problem that it encounters is whether it should store all of its movies on every server, or whether it should store a fraction of every movie per server. If it chooses to do the former, it will incur a ridiculously high storage requirement for low communication — a user will only have to query the local server to get the full movie, but each server on the network will need 1 petabyte of space. If Netflix decides to do the latter, then it incurs a low storage requirement for a ridiculously high communication requirement — Netflix would get to have comparatively small server sizes, but a user would have to query a local server which in turn would have to access *every server on Netflix's network* to get the full movie, which would be very inefficient and cause high latency for the user.[2] This problem motivates our research; there needs to be an analytical way to characterize this trade-off. In particular, understanding this trade-off is of increased interest in recent literature in information and communication theory [3]. The main contribution of this paper is to formulate a simplified model of local storage capacity versus communication bandwidth consumption and to quantify it in some special cases.

The problem at hand is a need to analytically characterize the trade-off between local storage capacity and network communication requirements. In some situations communication bandwidth may come at a premium, such as when the nodes are connected via slow links. An example of this would be media entertainment on airplanes — where communication bandwidth is limited, but storage is relatively plentiful. The reverse is also true; in a network of small mobile devices, WiFi might provide ample bandwidth for communication, but the devices' size limitations might not allow for large local storage capacity on each device. It is important for there to be a mathematical model to characterize this trade-off.

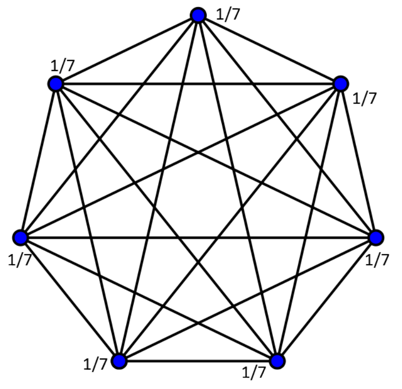
2 Network Models

Our network consists of a set of storage nodes and a set of communication links. We represent our data by a graph, where the nodes of the graph represent the storage nodes and the edges of a graph represent to communication links. In our paper will focus on (simplified) undirected symmetric graphs as the main model. The types of graphs we will focus on are cliques and rings. A clique is a graph in which every node is connected to every other node. A ring, on the other hand, is a graph in which every node is only connected to two other nodes and it follows the properties of a polygon — the graph creates a closed chain. Our graphs are much more simplified models of real-world infrastructures. In a real data distribution system, the graph's edges might be weighted to show different bandwidths, servers might have upper bounds on the amount of information they can store, and the graph might not be symmetric.

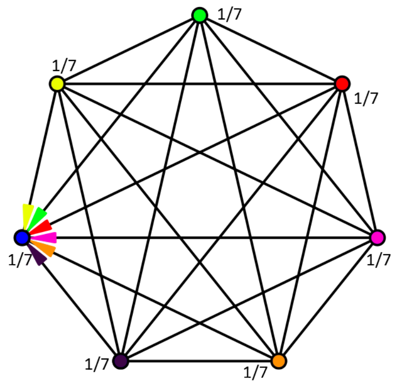
To understand our methods, consider a symmetric graph with N nodes. Suppose there is a file of a fixed size that is meant to be stored on the nodes (such as the movie from the Netflix analogy). We assume that each node's storage capacity is $\alpha * \text{size of file}$ so α represents the ratio of the storage capacity of the server to the file size. If

$$\alpha > \frac{1}{\text{number of nodes}}$$

then there will be redundant information on the graph. Network coding helps make sure that we are able reconstruct the data from any of the nearby pieces, even if some of the data seems redundant. This will be further explained in Section 3.



(a) Example of a 7 node clique graph



(b) Requesting node receiving the data

Figure 1: A 7 node clique

In this example, α is chosen to be $\frac{1}{7}$, as seen in Fig. 1a. Each node can store $\frac{1}{7}$ of the file locally. A node wants to access the full file. Since the node does not store the entire file (it stores only a fraction of the data), it needs to connect to other nodes. The goal of each node is to connect to other nodes in the graph and access their information to end up with the full file. In this particular scenario, each node must only "get" $\frac{6}{7}$ of information from other nodes, since it already starts out with $\frac{1}{7}$ as seen in Figure 1(b).

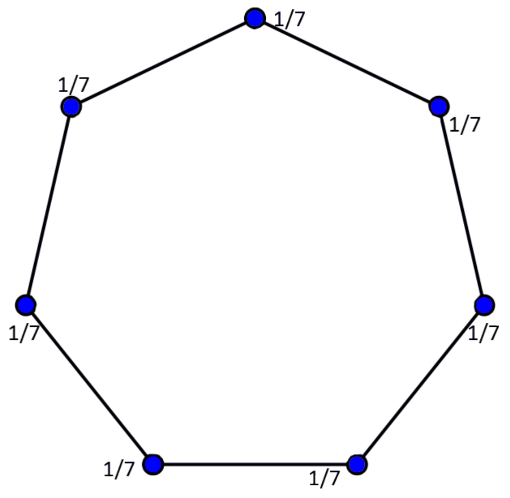
In Figure 1(b), the node received $\frac{6}{7}$ of data from the other nodes for a total of $\frac{7}{7}$, thus making the full file.

2.1 The communication trade-off $C(\alpha)$

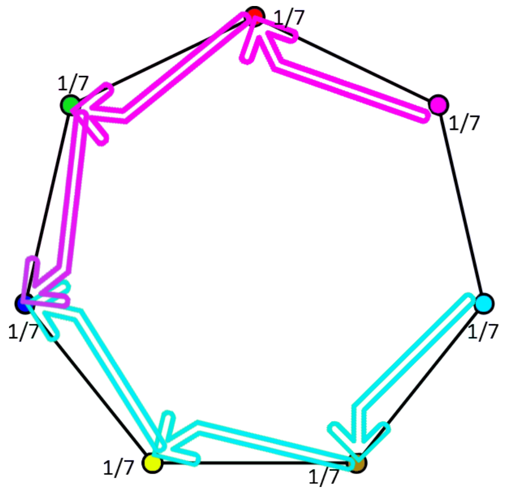
The communication storage trade-off, denoted as $C(\alpha)$, can be most simply described as the relationship between storage capacity per node and the amount of communication needed to make the full file. The $C(\alpha)$ is the sum of all the data sent on every edge in the graph. In the example considered above, the value of $C(\alpha)$ would be $\frac{6}{7}$ for $\alpha = \frac{1}{7}$, There were 6 edges that carried data, and each of these edges carried a total of $\frac{1}{7}$ which results in $\frac{6}{7}$ sent total. Each node is only 1 hop away from every other node, so it incurs a small communication cost — no edge needs to carry more than α. Determining the function $C(\alpha)$ in a graph like that above is simple, since each node is connected to every other node. For $\alpha < \frac{1}{7}$, the total storage capacity of the whole system is less than the actual data to be stored. Therefore, such a scenario would not be feasible since the node cannot access the file no matter how large the communication cost is, so we say that $C(\alpha) = \infty$. If there are 7 nodes, and each node has a local storage capacity of less than $\frac{1}{7}$, then a full file cannot be created.

The example graph above is a *clique*. This also means that the function $C(\alpha)$ is the same for every node, since they are all symmetrical. The problem considered is finding the function $C(\alpha)$ for symmetric graphs beyond cliques.

An example of such a graph is shown in Figure 2a. This is a ring graph. It still has 7 nodes, but each node is only connected to two other nodes. We calculate the $C(\alpha)$ for nodes on this graph.



(a) A ring graph



(b) The light blue and pink nodes have to send their data 3 times through the graph for it to reach its destination

Figure 2: A 7 node ring

Determining the function $C(\alpha)$ for the above graph is more complicated, since the starting node will need to access nodes that are farther than 1 connection away. Some nodes will have to transmit their data to a node closer to the starting node, and then that node sends it to the starting node, as seen in Figure 2b.

In our model, getting data from a farther node is more expensive from a communication perspective. We determine the cost of accessing data by the distance it has to traverse on the graph. Receiving $\frac{1}{7}$ of the file from a neighboring node costs $\frac{1}{7}$. However, from a node that is 2 hops away the cost will be $\frac{2}{7}$, because that information has to travel through two edges. Figure 2b above shows the cost of accessing a node that is 3 hops away. The $\frac{1}{7}$ of the file that is being sent from the node has to travel through 3 edges, thus making its cost $\frac{3}{7}$. Since the function $C(\alpha)$ is just the sum of all the data on every hop, the $C(\alpha)$ for this graph is $\frac{12}{7}$. The total can be measured as $C(\alpha) = \frac{1}{7} + \frac{1}{7} + \frac{2}{7} + \frac{2}{7} + \frac{3}{7} + \frac{3}{7} = \frac{12}{7}$. This will be further discussed in section 4.3.

In Section 3 we describe the aspect that Network Coding plays in the communication trade-off, and how it is applicable to our scenarios. In Section 4 we derive an equation to determine the communication trade-offs for rings. We also define the role that the Dijkstra shortest path algorithm plays, and how it is crucial to discovering the $C(\alpha)$ for any graph.

3 Network Coding

For values of $\alpha > \frac{1}{N}$, there is no guarantee in the graph that the node closest to the starting node will contain unique data. [4] The fraction of the file stored at the first node might be the exact same as the next node. If such redundancy occurs, we would end up with an inflated $C(\alpha)$ (because the requesting node would have to connect to a node farther away to complete the file), or an incomplete file (If some fraction of the file is not stored on the graph, the file will never be complete). We explain this idea in an example below.

Take a four node graph. Suppose $\alpha = 1/3$; that is each node can hold 1/3 of the file. Let each fraction of the file have a representation, in which the first third is A_1, the second third is A_2 and the final third is A_3. For the sake of our example, assume that $A_1, A_2, \text{and} A_3$ are each equal to 1 bit. If we were to lay these out on the graph, one node would have a repeated third, as shown in the figure below. This is labeled as A_k. The methods used here will only work if $A_1, A_2, \text{and} A_3$ are all equal sized binary units.

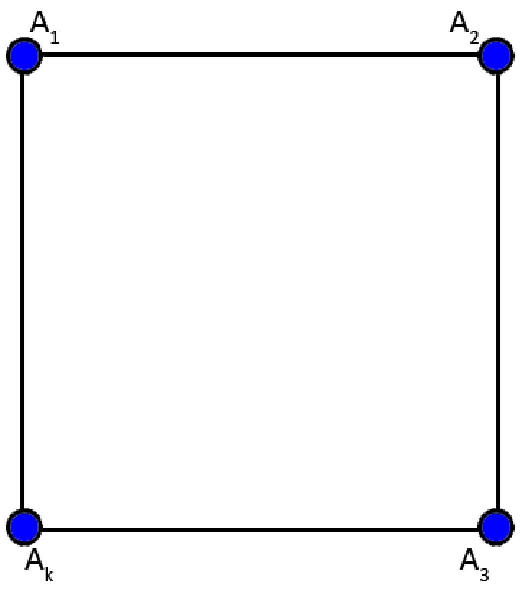


Figure 3: 4 node graph in which A_k is the repeated node

When A_k is replaced with either $A_1, A_2, \text{or} A_3$, there are certain nodes on the graph that will have a higher $C(\alpha)$ than should be necessary.

	$C(\alpha)$ for A_1	$C(\alpha)$ for A_2	$C(\alpha)$ for A_3
A_k=A_1	1	2/3	1
A_k=A_2	2/3	1	2/3
A_k=A_3	1	2/3	1

We can create a strategy that makes $C(\alpha) = 1 - \alpha = \frac{2}{3}$ for all nodes. This strategy involves coding theory. If the last node could be replaced with an XOR of $A_1, A_2, \text{and} A_3$, then the $C(\alpha)$ could be returned to its optimal state, $\frac{2}{3}$. Assume that the last node is v, or $A_1 \oplus A_2 \oplus A_3$ [5]. The node is only left with one value, and that value cannot be re-translated into the full $A_1, A_2, \text{and} A_3$ unless 2 of the other $A_\#$ are there to cancel out. Even though we know that a mysterious value $v = A_1 \oplus A_2 \oplus A_3$ [6], the three $A_{\#}$ have had a logic operation done on them and are now only one value, v. From v the user can get any A if the other two are already known. That is where the nodes come in. If the user were to start at A_1, he could take $\frac{1}{3}$ from A_2. Now the user has A_1 and A_2, and can do $A_1 \oplus A_2$. Then he takes v from the last node and subtracts $A_1 \oplus A_2$ from that. Since v is just equal to $A_1 \oplus A_2 \oplus A_3$, the user is now able to get A_3 while keep $C(\alpha)$ at its most efficient possible value. At its simplest, solving XOR logic functions is just solving a system of equations. The $\oplus$ functions can be done on multiple nodes, to give it its highest efficiency. Network coding generalizes the idea above to ensure that there is no redundancy in the data being accessed from different nodes. As long as a node can access the data from the minimum number of required nodes stored in the graph (including the data stored locally) where $m * \alpha > 1$, the node can reconstruct the entire file."

Network Coding is a candidate solution for solving this problem in p2p and bit torrent networks. One problem that plagues such programs is that the same information is sent multiple times from multiple hosts, when it would be many times more efficient if it were coded correctly and then only sent to those that needed it. It would ensure that everyone that requested a file wouldn't have to wait for the rarer pieces of information.

4 Results

In Section 4.1, we show how to determine the trade-off $C(\alpha)$ for any symmetric graph. Our technique uses the shortest path algorithm to determine the lowest value of $C(\alpha)$. We analytically derive the communication storage trade-offs for cliques and rings in Sections 4.2 and 4.3 respectively. Finally, in section 4.4 we use Dijkstra's shortest path algorithm on graphs of higher degree.

4.1 Shortest path algorithm

To find the trade-off $C(\alpha)$ for any graph, we take the following sequence of steps: Assume that a user wants to access the data from a particular node.

1. Find the minimum # of nodes the user need to access.

   $$m = \lceil \frac{1}{\alpha} \rceil$$

   where m is the minimum number of nodes a user need to access to get the full file. m nodes are sufficient because $m\alpha \geq 1$.

2. Find the minimum possible cost of accessing data from a node in the graph. The cost of accessing α units of data from a node is $\alpha *$ (length of a path between the two nodes)

3. From steps 1 and 2 we now know how many nodes we need and the cost of accessing data from each individual node, and all we need to find out is how to choose the m nodes. To do this we choose the m nodes closest to the user, by sorting the nodes of the graph in ascending order based on distance, and then choosing first m nodes. The distance is calculated via the Dijkstra shortest path algorithm. This approach chooses the m nodes that provide the smallest communication cost from the perspective of the user's node.

4. Let $N_1, N_2 \ldots N_m$ be the m nodes chosen from the previous step. $D(N_k)$ is the length of the shortest path from the user's node to node N_k. The communication cost is determined as $C(\alpha) = \sum_{k=1}^{m} D(N_k)$

The function $C(\alpha)$ for rings and cliques below is derived based on the above process. The shortest path algorithm is used to rank each nodes viability as a low-cost option. The Matlab code for the above steps is in the appendix.

4.2 Cliques

As a simple example consider a fully connected graph; a clique. For clique graphs, the communication trade-off is $C(\alpha) = 1 - \alpha$ (see [7]) for numbers $\frac{1}{N} \leq \alpha < 1$, where N is the number of nodes in the graph. For values of α where $\alpha < \frac{1}{N}$, the graph is infeasible and $C(\alpha) = \infty$. Cliques are easy to analyze because no matter the size of the graph, the equation is always $C(\alpha) = 1 - \alpha$, since the shortest path will always be 1. These results were previously known and have been done by Dr. Viveck Cadambe ([7]).

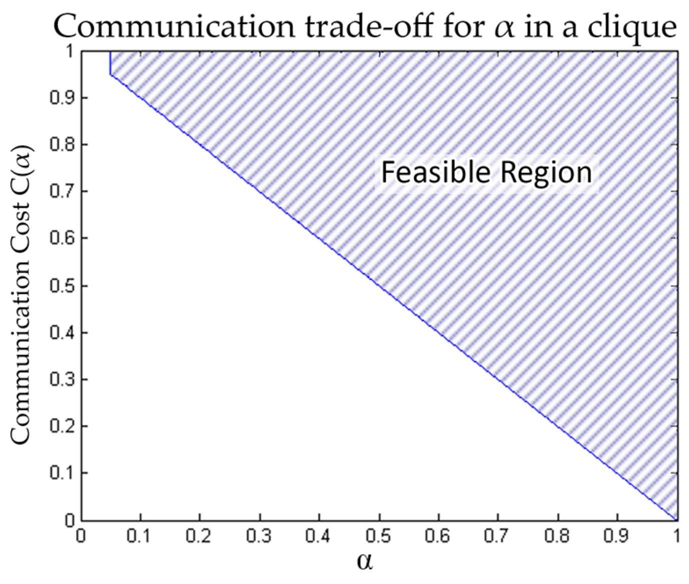


Figure 4: Plot of the communication trade-off in a clique

4.3 Rings

For a regular n-gon, $\alpha \geq \frac{1}{N}$ always has to be true, or the graph is unsolvable. In a square, the lowest α can be is 1/4, in a pentagon 1/5, in a hexagon 1/6, etc. The general set of equations for a symmetric n-gon are $\frac{1}{N} \leq \alpha \leq 1$ and while $1 > \alpha \geq \frac{1}{3}$ then $C(\alpha) = 1 - \alpha$ and while $0 < \alpha < \frac{1}{N}$ then $C(\alpha) = \infty$. For numbers between $\frac{1}{N} \leq \alpha \leq \frac{1}{3}$ the equation differs. For example, in squares $C(\alpha) = -4\alpha + 2$ for $\frac{1}{3} < \alpha \leq \frac{1}{4}$. For pentagons $C(\alpha) = -4\alpha + 2$ for

$\frac{1}{3} \leq \alpha \leq \frac{1}{5}$ For hexagons $C(\alpha) = -9\alpha + 3$ for $\frac{1}{5} \leq \alpha \leq \frac{1}{6}$. For heptagons $C(\alpha) = -9\alpha + 3$ for $\frac{1}{5} \leq \alpha \leq \frac{1}{7}$.

There exist inequalities in this mathematical model because α gets smaller, it needs to access nodes that are farther and farther away to get the full file. We plot different values of α and then use those to derive the inequalities. We can then plot every inequality, and then we get a graph that characterizes the communication trade-off for that specific graph. An example of such a trade-off is Figure 6.

Assume that

$$m = \lceil \frac{1}{\alpha} \rceil$$

m is equal to the minimum number of nodes that are needed to store the full file for that specific value of α. m is the ceiling of the of $\frac{1}{\alpha}$, because the number of nodes must be an integer. For an n-gon where m is even, the function $C(\alpha)$ is, in fact

$$C(\alpha) = -(\frac{m}{2})^2\alpha + \frac{m}{2}$$

For an n-gon where m is odd, the function $C(\alpha)$ is, in fact

$$C(\alpha) = -(\frac{m-1}{2})^2\alpha + \frac{m-1}{2}$$

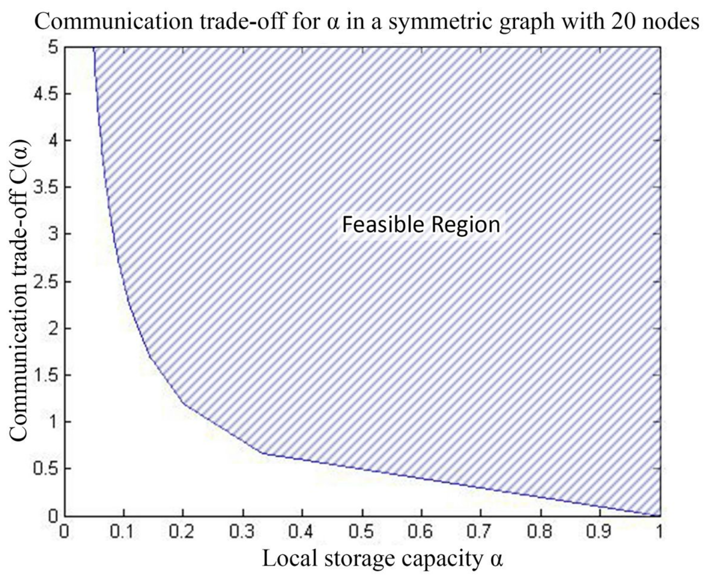


Figure 5: Plot of the communication trade-off in a 20 node graph

4.4 Graphs with different degree

In Figure 7, we plotted the function $C(\alpha)$ of the hexagon of degree 3 seen in Figure 6. We used an adjacency matrix as the input to Dijkstra's shortest path, and then found the cost using the steps detailed in section 4.1

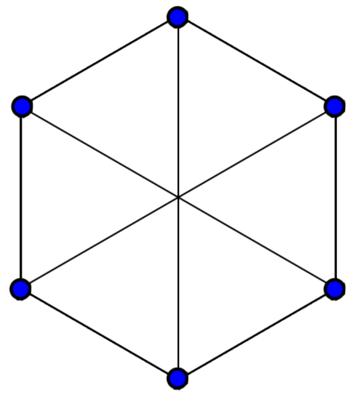


Figure 6: Hexagon with degree 3

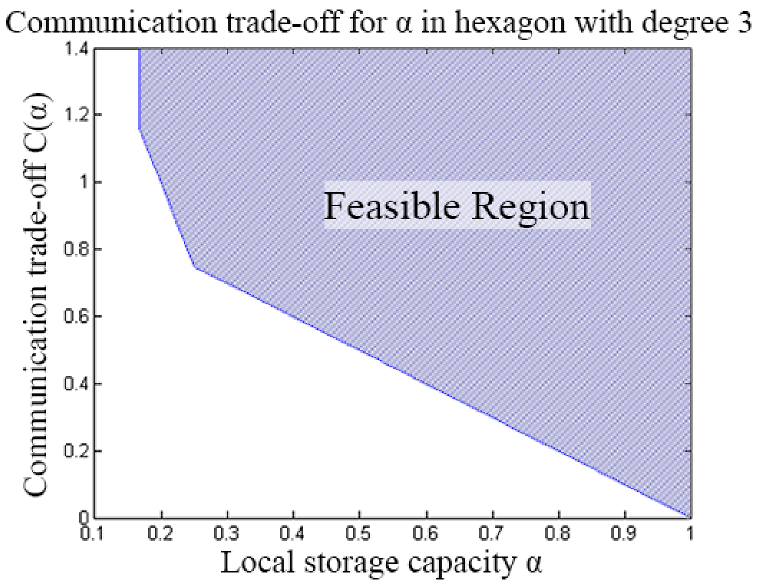
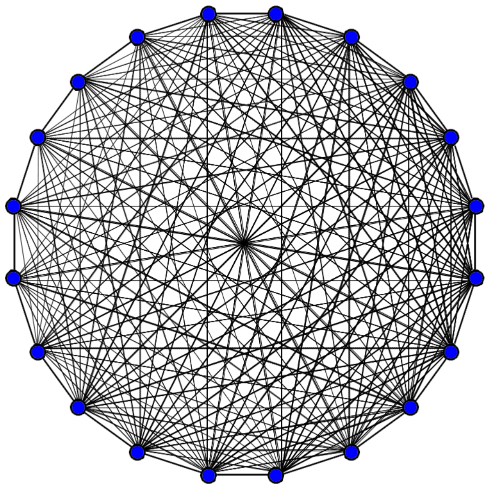
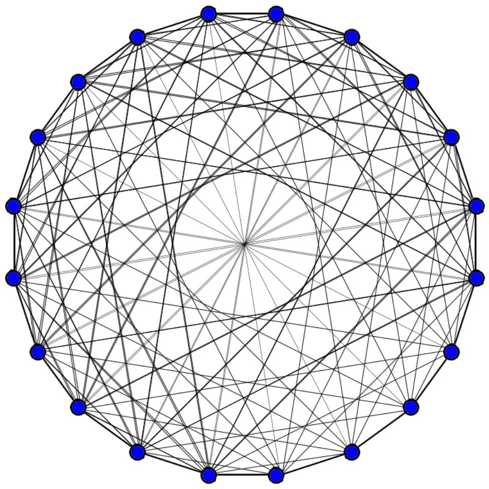


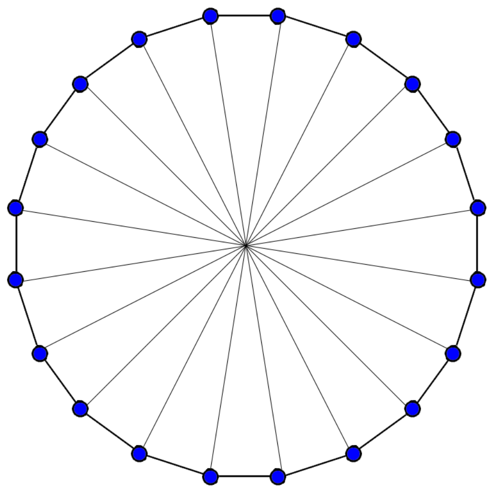
Figure 7: Plot of the communication storage trade-off of the hexagon in Fig. 6



(a) A 20 node clique



(b) A 20 node graph of degree 11


(c) A 20 node graph of degree 3

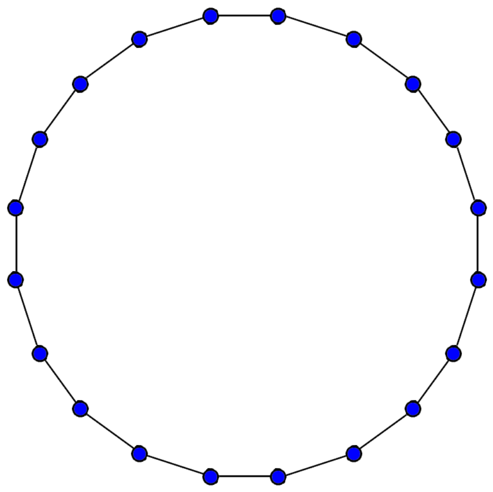

(d) A 20 node ring

Figure 8: A series of 20 node graphs

Shown in Figure 8a, 8b, 8c, and 8d are four different 20 node graphs — a clique, a graph with degree 11, a graph with degree 3, and a ring, respectively. Using the approach outlined in 4.1, we managed to characterize the trade-off, as shown in Figure 9. This approach allows us to calculate the value of $C(\frac{1}{20})$ for every node in any symmetric graph.

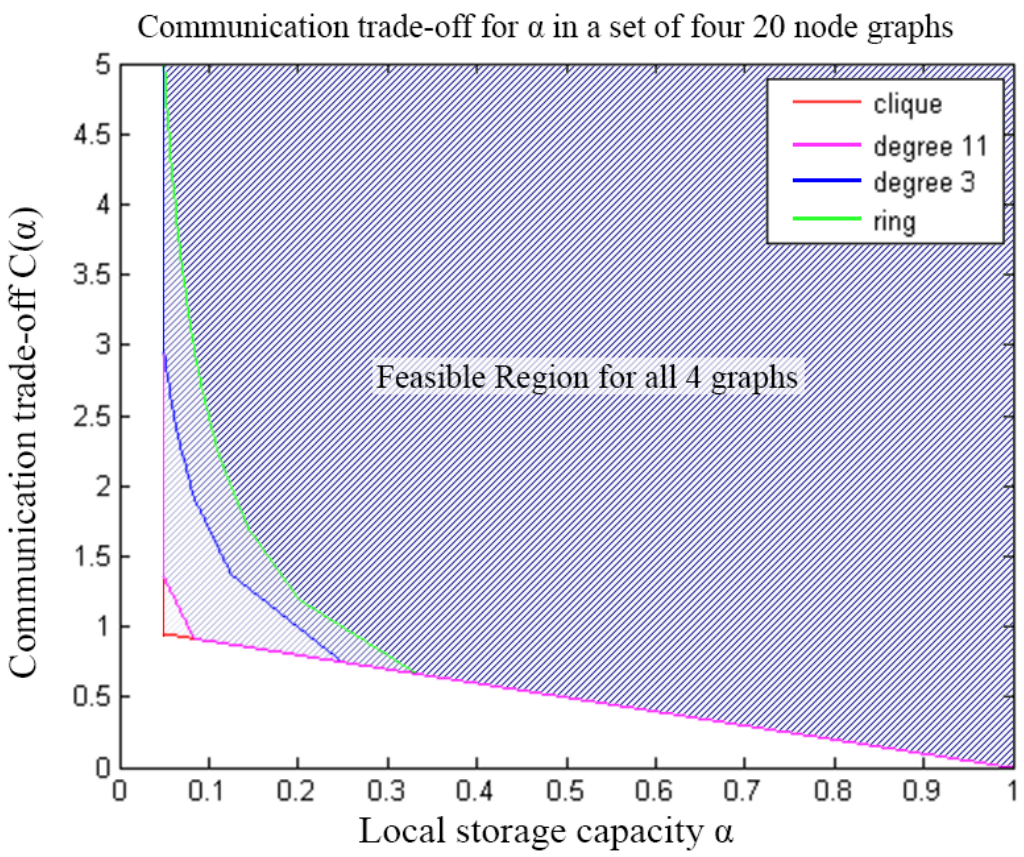


Figure 9: Plot of the communication-storage trade-off for a series of 20 node graphs

5 Application and Importance

The importance of deriving the communication trade-off is for anyone who handles a network that distributes content. [8] For example, a CDN could theoretically implement these algorithms to improve the efficiency of their networks, and make it so that information that is requested more often is on closer nodes and on nodes that can handle more data. We are only trying to capture generalizations on networks, not dealing with any specific product or network in particular.

Another example would be p2p networks, which are, at their simplest, just a giant swarm of nodes connected to each other. These more complicated systems are much more diverse, and suffer from problems that can't be solved in graph theory (i.e. bottlenecks, file availability, discovery of hosts, etc.), but that would still be better understood, and could potentially be improved if the $C(\alpha)$ was considered and optimized. If coded correctly, these p2p networks could actually remove most of their redundancy and problems. The most accessed parts of a file would be stored on nodes with the most bandwidth, and everything would be appropriated to be the most efficient as possible.

6 Conclusion

We described and analyzed a simplified model for the trade-off that exists between the storage capacity of a node α and the amount of communication $C(\alpha)$ it needs to make to receive the full file. Furthermore, we also analytically derived the equations for any value of α in a clique and rings. The implications of the research could be used to understand and improve Content Delivery Networks (CDN) and bit torrent networks. Specifically, the function $C(\alpha)$ can be used to compare two candidates for the network graph, and enable a CDN provider to make a more informed choice.

Some open problems in determining the communication cost $C(\alpha)$ are understanding graphs that are not symmetric, and finding analytical insights for graphs that are more complicated than cliques or rings. Another research problem is determining the best graph of certain degrees and values of α. That just means the graph with the smallest $C(\alpha)$ for each node.

7 Acknowledgments

I would like to thank Dr. Viveck Cadambe for his mentorship and help during the research, Mr. Sam Spencer for the help in refining the paper, and CEE, RSI, and MIT for providing the opportunity to do this research. My research also wouldn't have been possible without the sponsorship of Dr.Laura Adolfie, Dr. Reginald Brothers, Ms. Kim Day, and Dr. Christine Hill of the Department of Defense Education Activity. I am also grateful to Mr. John Yochelson, of Building Engineering and Science Talent. I would also like Prof. Médard for her input and advice throughout the project.

References

[1] D. Goldman. Netflix builds its own delivery network. 2012.

[2] S. Borst, V. Gupta, and A. Walid. Distributed caching algorithms for content distribution networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[3] M. A. Maddah-Ali and U. Niesen. Fundamental limits of caching. *arXiv*, 2012.

[4] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *Information Theory, IEEE Transactions on*, 52(10):4413–4430, 2006.

[5] S.-Y. Li, R. W. Yeung, and N. Cai. Linear network coding. *Information Theory, IEEE Transactions on*, 49(2):371–381, 2003.

[6] R. Koetter and M. Médard. An algebraic approach to network coding. *Networking, IEEE/ACM Transactions on*, 11(5):782–795, 2003.

[7] V. Cadambe. Personal Communication.

[8] M. Yu, W. Jiang, H. Li, and I. Stoica. Tradeoffs in cdn designs for throughput oriented traffic. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 145–156. ACM, 2012.

A MATLAB code from Bharat Patel for Dijkstra shortest path (http://bit.ly/1cl6Ar7)

function [costs] = mdijkstra(A,C)

```
n = length(A);
costs=single(A);
costs(costs==0)=Inf;
for k = 1:n
    disp(sprintf('%d/13-%d/%d',C,k,n));
    w_col=single(costs(:,k));
    if C==1
        mx=max(w_col(find(w_col~=Inf)));
        pmx=0;
        while(mx~=pmx)
            cols=find(w_col==mx);
            tmp=min(costs(:,cols),[],2);
            tmp=tmp+mx;
            tmp1=[w_col tmp];
            w_col=min(tmp1,[],2);
            pmx=mx;
            mx=max(w_col(find(w_col~=Inf)));
        end
        costs(:,k)=w_col;
    elseif C==2
        m1=(w_col(find(w_col)));
        m=sort(unique(m1),'ascend');
        for j=1:length(m)
            mx=m(j);
            cols=find(w_col==mx);
            tmp=min(costs(:,cols),[],2);
            tmp=tmp+mx;
            tmp1=[w_col tmp];
            w_col=min(tmp1,[],2);
        end
        costs(:,k)=w_col;
        costs(k,:)=w_col';
    end
end
for k=1:n
    costs(k,k)=0;
end$$
```

B MATLAB code from JonLuca De Caro on the cost for a hexagon

```
N=6
A=zeros(N);
for i=1:N;
    for j=1:N;
        if rem(i-j+1,N)==0
            A(i,j) = 1;
        end
        if rem(j-i+1,N)==0

            A(i,j) = 1;

        end

        if i-j==N/2

            A(i,j) = 1;

        end

        if j-i==N/2

            A(i,j) = 1;

        end

    end

end


A;




L=mdijkstra(A,2);

M=sort(L(1,:));


alpha=[1./N: (1-1./N)/1000: 1];

for V=[1:length(alpha)];

D(V)=(sum(M(1:(ceil((1./alpha(V))-1)))))*alpha(V);


D(V)=D(V)+(M(1,(ceil(1./alpha(V))))).*(1-alpha(V).*ceil(((1./alpha(V))-1)));


end

plot(alpha, D)
```