Lossy Ad-hoc Networks

By Brett Paris, Connor Rehbein, and Austin Mease

In networking, an ad-hoc network is a collection of wireless mobile nodes dynamically forming a temporary network without the use of an existing infrastructure or centralized administration. Wireless nodes are free to enter and leave the network at any time causing the network topology to change rapidly and unpredictably. Ad-hoc networks are generally used for peer to peer applications but can also have access to the internet through a node and can use the internet's services. Due to the mobility, unpredictability, and size of the network make designing routing protocols challenging. In this project, we look at the network characteristics of one-hop and two-hop UDP/TCP data transmission in an ad-hoc network and try to pinpoint its advantages/disadvantages.

Most ad-hoc networks used commercially, for example Bluetooth, are a one-hop network. This is because they are more easily understood and are less complex. When you have a network where any 2 users can communicate with each other you must introduce routing protocols. Since the nodes can vary in processing capabilities you want the routing protocols to be efficient while still being done in a timely manner. There are many different design choices involved in creating a routing protocol such as proactive (keeping an updated list of reachable nodes) or reactive (getting a distance vector to a node upon request). The best routing protocol for an ad-hoc network is yet to be found and is an active area for research.

Another challenge in ad-hoc networks is security. Because the routing is done at the node level it Is easy for a malicious user to sabotage the routing of packets. Since there is no centralized authority, authentication to join a network can be hard to implement. Since the network can sometimes consist of very unsecure nodes users are able to hack into less secure nodes to easily gain access to the network. Once a malicious user is in the network it can be very hard for the network to identify this user. This creates security challenges at multiple network layers and is shown nicely in a table taken from the book: *A Hoc Mobile Wireless Networks by Subir Kumar Sarkar T.G. Basavaraju C. Puttamadappa*:

LAYER	SECURITY ISSUES
Application layer	Detecting and preventing viruses, worms, malicious codes, and application abuses
Transport layer	Authenticating and securing end-to-end communications through data encryption
Network layer	Protecting the ad hoc routing and forwarding protocols
Link layer	Protecting the wireless MAC protocol and providing link-layer security support
Physical layer	Preventing signal jamming denial-of-service attacks

While ad-hoc networks have a lot of challenges involved in its implementation it can be a useful solution in certain situations. For example, in a natural disaster where the local infrastructure gets destroyed an ad-hoc network would be able to be set up in a matter of hours to be able to contact people in need of help. Another implementation is in sensor networks. Sensors could be put in various environments and communicate with each other without having to be in a location that has access to

the internet. This could be used to track the conditions of an ecosystem in a remote area. There is also a nice list in the same book of potential applications of an ad-hoc network:

- Community network
- Enterprise network
- Home network
- Emergency response network
- Vehicle network
- Sensor network
- Education
- Entertainment
- Coverage extension
- Commercial and civilian environments

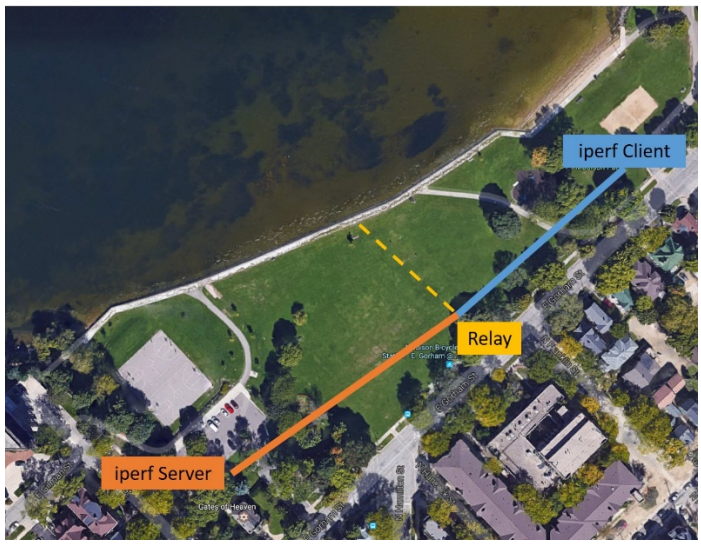
For this project, we were tasked with observing the network characteristics of an ad-hoc network. We looked at the data transmission via UDP/TCP in 1/2 hops by measuring the jitter, packet loss, and speed as well as other factors to try to get a better understanding of how an ad-hoc network functions in the real world.

Collecting the data was difficult because we were trying to use three different operating system, each of which had different security issues. Ad-hocs for Windows could only interface with other Windows. A Mac wouldn't allow it at all. As a result, three of us obtained Ubuntu laptops and developed two scripts (see attached) to use for setting up the ad-hoc networks. In addition, we used iperf –s –u or iperf –s to measure performance of the system, extracting bandwidth, jitter and packet loss from this information. TCP data does not include loss or jitter because TCP will resend lost packets (providing no loss). The data as well as the Matlab script used to analyze the data area attached.

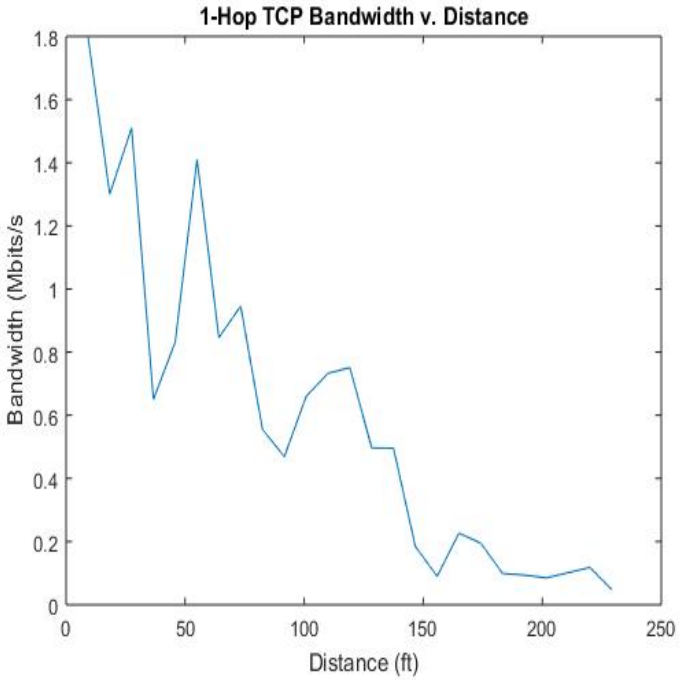
For the physical experiments, an initial challenge was finding locations that allowed us to reach the physical limits of the signal strength. For the 1-hop experiments, we decided to use the Buckeye parking lot. This proved ideal because we found that the signal limit was reached at the end of the lot, at approximately 250 feet. To perform the 2-hop experiments, we needed at least double the distance that we had for 1-hop, since the 2-hop involves 3 points. For this, we chose James Madison Park. This site allowed us to get a large distance without many physical obstacles. The client and server stood 845 ft apart.

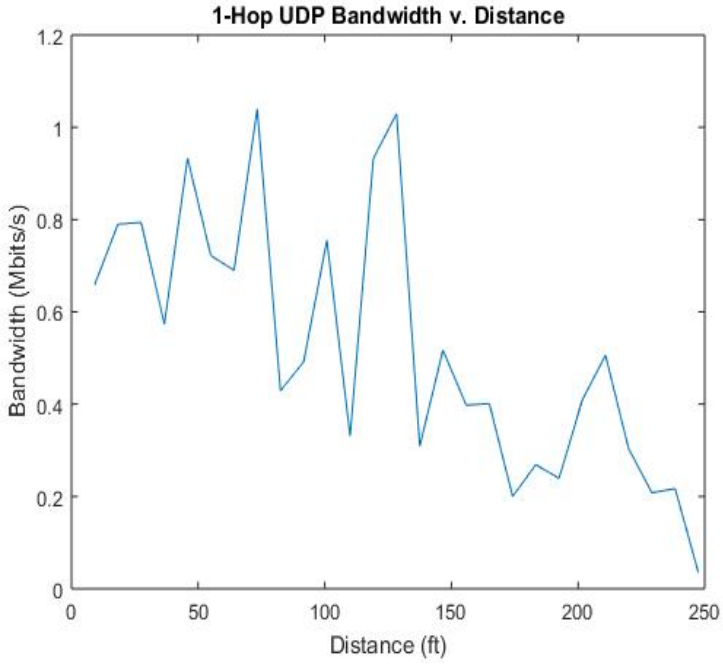
To perform the 1-hop experiment, we had the server stay stationary at one end of the parking lot. The client started next to the server, then moved one parking space distance (approx. 9.167 feet) away from the server. The client continued moving away until signal was lost.

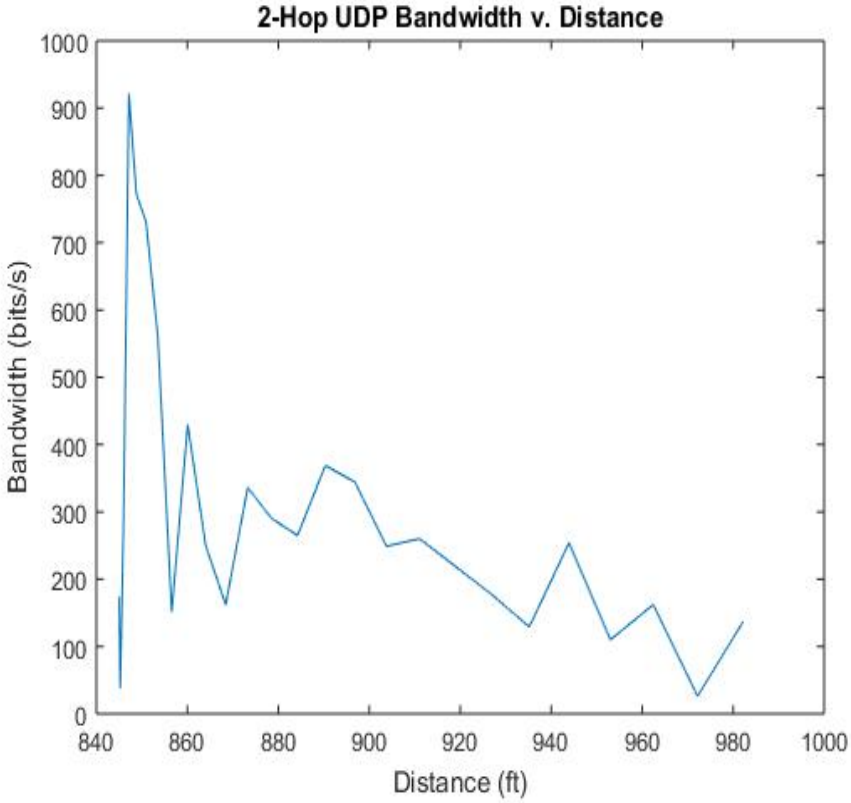
The 2-hop experiment was slightly more complex. The iperf server and client were at either end of the park, far enough away so that they cannot connect to each other directly. The relay, which was connected to the ad-hoc network, but not running iperf, started directly between the client and server. To increase the distance, after each successful collection, the relay would move 10 feet perpendicular (measure via tape-measure) to the server and client. Then the total distance can be calculated using the Pythagorean Theorem. Because of the great distance and low reliability of UDP, we had to decrease our packet size for this transmission method from 1MB to 22KB, otherwise our iperf serve did not record timing statistics.

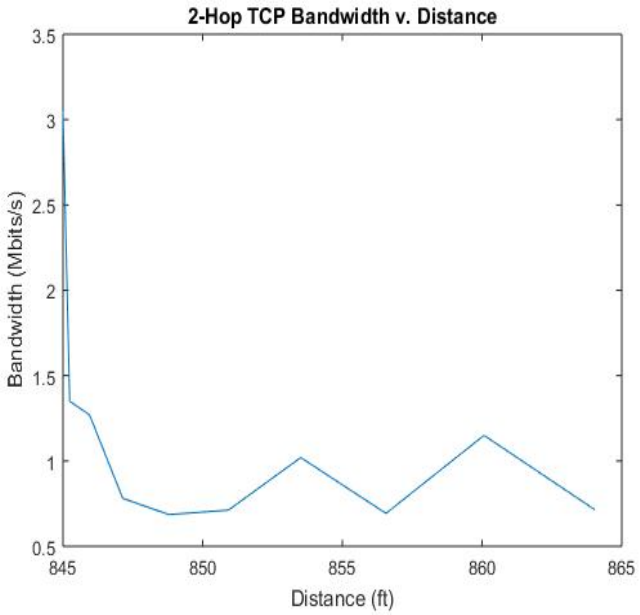


For bandwidth we have:



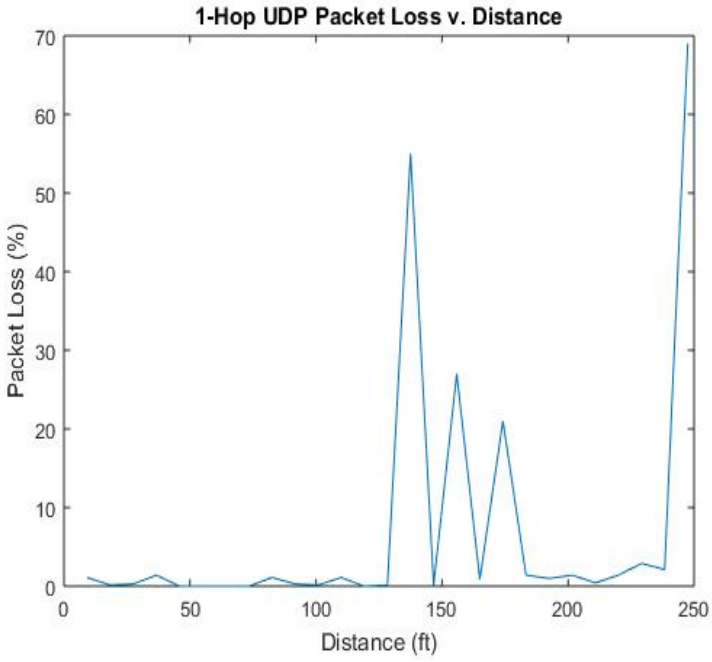

1-Hop UDP Bandwidth v. Distance
Bandwidth (Mbits/s)
Distance (ft)
0
0.2
0.4
0.6
0.8
1
1.2
0
50
100
150
200
250

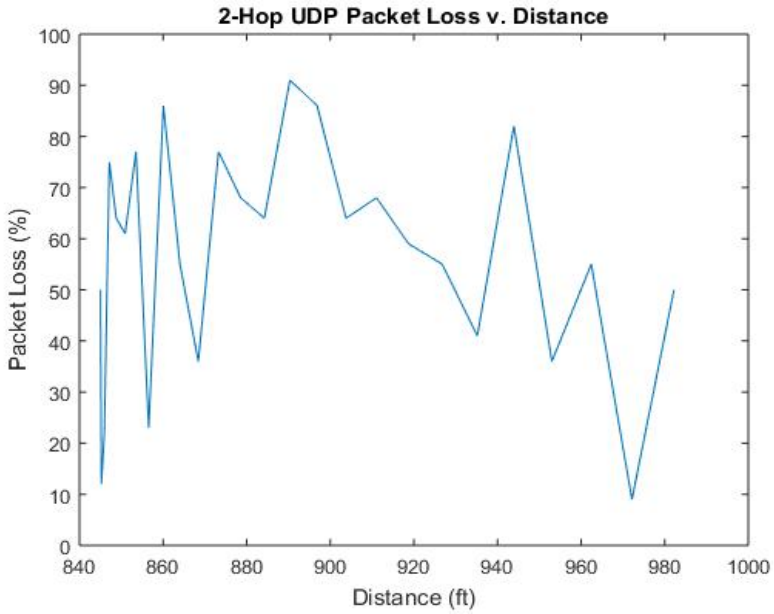

2-Hop UDP Bandwidth v. Distance
Bandwidth (bits/s)
Distance (ft)
0
100
200
300
400
500
600
700
800
900
1000
840
860
880
900
920
940
960
980
1000



As a general trend, TCP will have a better bandwidth at closer distances, but has a significantly quicker decline with respect to distance. This is because TCP will resend all lost data packets, and as we get further away these losses become increasingly more prevalent. This greatly lowers the viability of ad-hoc networks for significant commercial applications. However, lighter-weight services such as communication for emergency services is still viable under these extremely low bandwidth conditions. Bandwidth and throughput show an inverse relationship with each other, as can be seen in the attached time vs distance graphs under our plots folders.

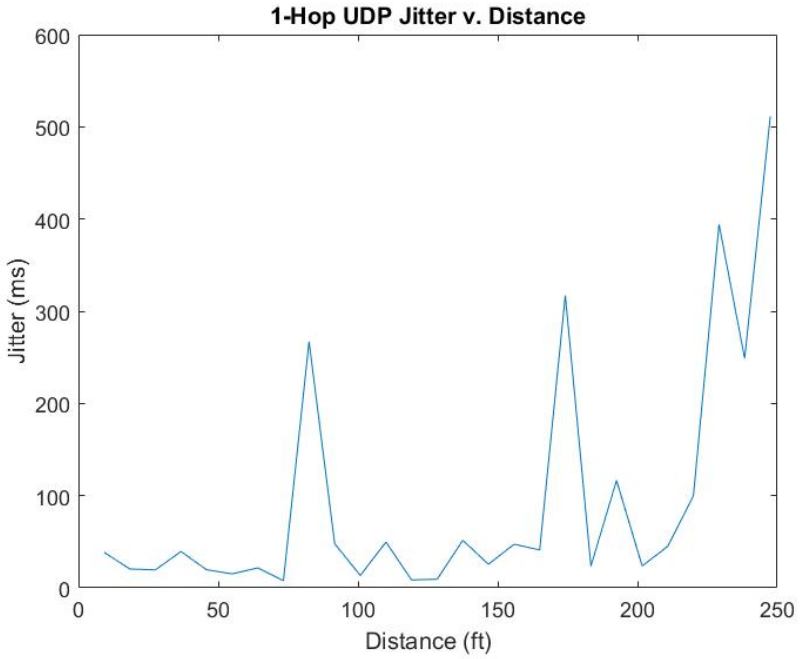
For loss we have:

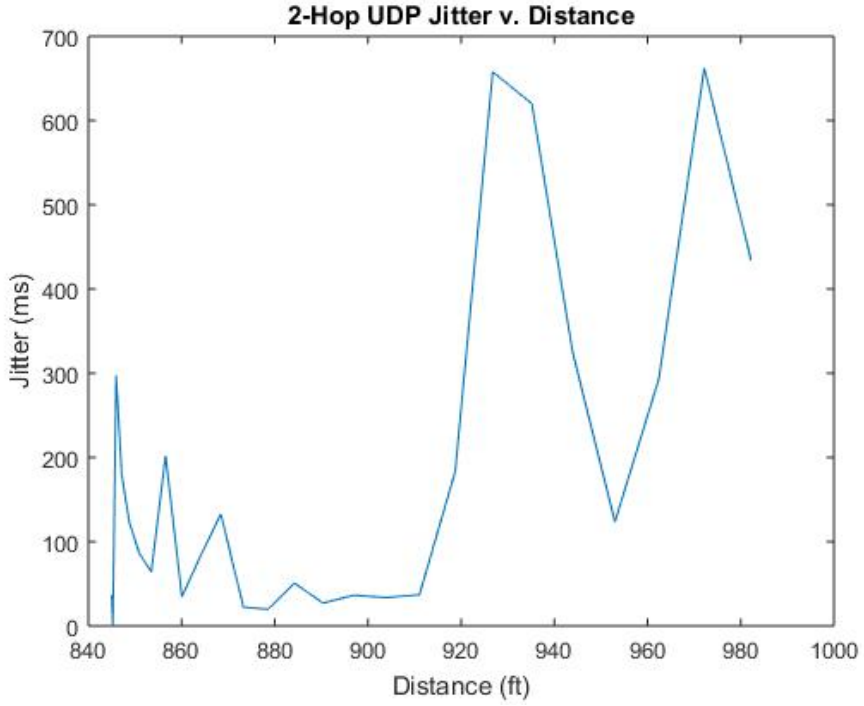




It's extremely fascinating that UDP packet loss seems to be less dependent on distance in the 1-hop case, and is instead prone to random outside forces. In the 2-hop case, it was high to begin with and stayed high until the end. This could be because the packets have to be received and sent twice, giving it twice as opportunity to drop.

For jitter we have:





UDP jitter grows exponentially with respect to distance, with the random peaks increasing in likelihood as distance increases. The instability and unreliability of the long-distance connection cause chaotic jitter.

One thing to note is that when we were taking our measurements for the 2-hop data transmission we noticed that sometimes the source would bypass the relay and try to directly connect to the destination. While we do not know the exact routing protocol in Linux's ad-hoc mode we think that it the source requestor will broadcast an IP that they wish to receive from, then any node that hears this request will try to find that IP. If a node hears this request and is the IP that the request is looking for then the node will respond saying to connect through me. If a node hears this request and doesn't have the IP the request it is looking for then it tries to flood its network with a request for the IP to see if it is within reach. Then if one of the nodes in its network responds then it will respond back to the original requestor saying that it can reach the IP and go through me.

We think that sometimes the destination would hear the IP request and respond to the source faster than the relay would be able to contact the destination and respond. This would bypass the router and even though the destination responded to the request quicker than the relay it is a much spottier connection due to the distance between the source and destination. The complete data transmission would have been smoother to go through the relay because between each individual node there is a shorter distance and there would be less packet loss.

This poses a problem when constructing an ad-hoc network, how does a node choose the best path? Since distance creates a much larger packet loss than speed loss, how do you make a network that can pick up on this and choose the best path?

A proposed solution is using GPS to get the location of a node and then use that in the request/accept message so a node can factor distance into the routing equation. Exhaustive testing would need to be done to have a good understanding of the packet loss at different distances, then what you could do is have a cost based routing function and assign a weight to whatever distance the node is from one another. Combine that with the pinged throughput to have a more accurate routing equation that would produce less packet loss. GPS technology may have to improve because the distances between nodes in ad-hoc networks are relatively small. Also, this adds overhead and would require the nodes to be more power hungry by using GPS. Maybe there will be a new technology that would be able to track relative distances instead of using GPS coordinates.

Appendix

Data:

1hop TCP

Time (s)	Bandwidth (Mbits/s)	Distance (ft)
4.7	1.8	9.167
6.4	1.3	18.334
5.5	1.51	27.501
12.9	0.65	36.668
10.1	0.832	45.835
6	1.41	55.002
9.9	0.846	64.169
8.9	0.946	73.336
15.1	0.555	82.503
17.9	0.469	91.67
12.7	0.66	100.837
11.4	0.733	110.004
11.2	0.751	119.171
16.9	0.497	128.338
16.9	0.496	137.505
45.2	0.185	146.672
93.2	0.09	155.839
36.9	0.227	165.006
42.9	0.195	174.173
84.4	0.099	183.34
89.4	0.094	192.507
98.3	0.085	201.674
70.8	0.118	220.008
175	0.048	229.175

1hop UDP

Time (s)	Bandwidth (Mbits/s)	Jitter (ms)	Packet Loss	Distance (ft)
12.6	0.659	38.615	1.1%	9.167
10.6	0.79	20.496	0.14%	18.334
10.5	0.794	19.433	0.28%	27.501
14.4	0.573	39.419	1.4%	36.668
9	0.933	19.629	0%	45.835
11.6	0.722	15.091	0%	55.002
12.2	0.69	21.735	0%	64.169

8	1.04	7.756	0%	73.336
19.3	0.429	267.046	1.1%	82.503
17	0.492	47.802	0.28%	91.67
11.1	0.755	13.436	0.14%	100.837
25.1	0.331	49.627	1.1%	110.004
9	0.932	8.564	0%	119.171
8.1	1.03	9.369	0.14%	128.338
12.2	0.309	51.52	55%	137.505
16.2	0.517	25.537	0.14%	146.672
15.5	0.398	47.28	27%	155.839
20.7	0.401	41.069	0.84%	165.006
33.2	0.2	317.085	21%	174.173
30.7	0.269	23.606	1.4%	183.34
34.8	0.239	116.448	0.98%	192.507
20.2	0.409	23.647	1.4%	201.674
16.5	0.506	45.127	0.42%	210.841
27.3	0.303	99.809	1.4%	220.008
39.1	0.208	394.172	2.9%	229.175
37.9	0.217	248.831	2.1%	238.342
49.1	0.035	511.311	69%	247.509

2hop TCP

Time (s)	Bandwidth (Mbits/s)	Distance (ft)
2.7	3.05	845
6.2	1.35	845.24
6.6	1.27	845.95
10.7	0.781	847.14
12.2	0.686	848.79
11.8	0.712	850.92
8.2	1.02	853.51
12.1	0.693	856.56
7.3	1.15	860.07
11.7	0.715	864.03

2hop UDP

Time (s)	Bandwidth (bits/s)	Jitter (ms)	Packet Loss	Distance (ft)
269.9	174	37.093	50%	845
314.2	37.4	0	12%	845.24
338	278	297.433	22%	845.95

344.3	922	180.113	75%	847.14
350.2	772	122.73	64%	848.79
354.1	731	86.17	61%	850.92
357.2	560	64.148	77%	853.51
389.6	151	202.05	23%	856.56
519.6	430	34.581	86%	860.07
563.9	250	82.711	55%	864.03
580.9	162	132.817	36%	868.43
594.6	336	22.177	77%	873.27
608.3	290	19.79	68%	878.54
622.3	265	50.766	64%	884.23
637.1	369	27.06	91%	890.34
649.4	344	36.399	86%	896.85
662.2	249	33.695	64%	903.76
677.2	260	36.984	68%	911.05
695	220	183.359	59%	918.73
791.1	178	657.649	55%	926.77
821.5	129	620.122	41%	935.17
834.6	254	325.08	82%	943.92
856.7	110	123.726	36%	953.01
872.2	162	293.233	55%	962.43
902.8	26.1	662.331	9%	972.17
943.9	137	433.997	50%	982.22