A search for the optimal file transfer protocol from surfaced UUVs to UAV relays and beyond

Scott Cutler*[a]
[a]SPAWAR Systems Center Pacific, 53560 Hull St., San Diego, CA, USA 92152

ABSTRACT

While demonstrating the use of an Unmanned Air Vehicle (UAV) to provide a communications relay for surfaced unmanned underwater vehicles (UUVs) at Unmanned Warrior 2016, it became apparent that a closer look was needed to determine the optimal protocol to transfer automatic target recognition image files. Initial efforts with User Datagram Protocol (UDP) were highly unreliable with a very low success rate, despite extensive radio configuration improvements. Afterwards, several protocols were evaluated and UDP-based Data Transfer Protocol (UDT) seemed to provide a good hybrid of the standard transport layer protocols UDP and Transmission Control Protocol (TCP). UDT is an application layer protocol built on UDP with some TCP-like reliability characteristics. This paper documents a careful comparison study of UDP, TCP, and UDT that was performed to determine the optimal protocol for this form of data transfer. Each protocol was used to transfer image files, first in a baseline configuration, then with network emulation packet loss, and finally with real data radios in ideal and then more realistic conditions. Despite that fact that UDT was originally designed to transfer large data sets over high-bandwidth networks, this study demonstrated that it is also ideal for transporting relatively small data sets in high packet loss environments.

Keywords: UDP, TCP, UDP-based Data Transfer Protocol, UDT, UAV, UUV, image file transfer, communication relay

1. INTRODUCTION

1.1 Initial Problem

During Unmanned Warrior 2016, an international military exercise event, the author was participating with a multi-organizational team to demonstrate the capability of transferring an automatic target recognition (ATR) file from a surfaced UUV to a nearby UAV, then on to a ground station within range. During that event, an image file could not be successfully transmitted under normal operating conditions (i.e., while a UUV was in the water and a UAV was in the air). When this same equipment was tested in the lab, the image file was reliably transmitted on a consistent basis. A test was also done with the UAV in the air and the UUV sitting on the beach next to the water. Even in that scenario, the image file was successfully transmitted on a consistent basis. However, once the UUV was bobbing in the water, the 8 (kilobyte) KB image file would not make it all the way through. Parts of the file would come through, and the smaller 1 KB detection file came through a few times, but the image file itself simply could not make it during the available in-water testing time (50-100 attempts were made). As a final test, the radio payload was placed on the top of a stationary panel truck. At that time, the image file came through one time out of roughly 20 attempts.

The radios in use on the UUV and UAV and ground station were commercially available 900 Mhz spread spectrum data modems capable of data rates up to 1.2 Mbps, transmit power output up to 1W, and an advertised range greater than 10 miles. As the event was outside the U.S., the radios used were approved for use internationally (they were not restricted for use outside of the U.S. by the International Traffic in Arms Regulations (ITAR)). The radios were configured with the master radio (through which all data flowed) onboard the UAV payload, and two client radios: one onboard the UUV, and one connected to a ground control station on the beach. The UAV acted as a relay to allow greater standoff of the control station from the UUV. In the demonstration scenario, the UUV completed an underwater mission, then surfaced approximately 500 yards from the ground station. Direct radio contact between the ground station and the UUV could not be established, so a UAV was launched to extend the radio range.

*scott.h.cutler@navy.mil

Some basic telemetry and health information was exchanged when communications were established, and the ground station operator was notified that ATR image files were available for transfer. The operator then manually initiated the file transfer, which was performed using UDP as the transport layer.

The observers were surprised to see repeated image transfer failures in the live scenario, given that the same process had worked flawlessly in the lab using the exact same equipment earlier that day. For the next two days, nearly every radio setting was tested with some notable performance increases, but none big enough to produce even a single successful file transfer in the operating environment. The team decided to investigate using a different file transfer protocol to see if performance could be improved without changing any of the existing equipment.

1.2 File Transfer Protocols

As stated, all attempts at transmitting the 8 KB image file in question were initially done using UDP. UDP can be considered a fire-and-forget transport layer protocol, where the sender makes no effort by default to ensure that the receiver has received the data. It is also known as a connectionless protocol because there is no handshaking between endpoints. This allows data to always flow through in a timely manner and is ideal for situations where timing is more important than reliability.[1] Inevitably some of the data will be lost along the way with UDP, even when transmitting a file between one process and another on the same computer. Testing described in the sections below shows that transmitting a file larger than about 100 KB between two computers on a Gigabit network is unreliable (less than 100%) with UDP. When more packet loss is introduced, reliability of transmitting even small files is diminished.

The other major transport layer protocol used in modern networking is Transmission Control Protocol (TCP). This protocol is used whenever reliability of data transfer is more important than timeliness. A connection is established between the sender and receiver of data, and they work together to ensure every packet of data is delivered in the correct order and completely. If there is major packet loss, this process can endure up to infinity (a fact partially confirmed during this testing!) On networks with minimal packet loss, TCP is the ideal transport layer protocol to use when sending files from one location to another, and is used commonly to serve web pages, transmit email, transfer files, share peer-to-peer files, and stream media.[2]

The environment described above, where the desire was to send a complete image file from a UUV on the surface of the water, through a UAV, then back to shore, with radios that provided a reasonable tradeoff between range and bandwidth, does not lend itself particularly well to either UDP or TCP. As explained earlier, UDP is highly unreliable in this environment. TCP seems more likely to work, but given the lossy nature of the environment, it is rather possible that TCP would get bogged down infinitely (or at least for the limited flight time of the UAV) and file transfer would never complete. Because of this, additional protocols were discussed which could provide a happy medium. Several TCP-based application layer protocols were discussed, such as SCP, FTP, etc., but they were deemed likely to have all the same issues as TCP. Hybrid protocols, which use UDP but also provide more reliability seemed the most promising, and UDT and TFTP were discussed. Some initial reading led the author to choose UDT for additional evaluation because it was easy to find sample C++ code and was a more modern and robust protocol.

UDP-based Data Transfer (UDT) is an application layer protocol that is built on UDP, but provides reliability mechanisms. Its key advertised feature is that it provides much higher data transfer rates for large data sets on high bandwidth networks.[3] That isn't exactly the application here, but testing showed that UDT has some excellent advantages in high packet loss environments.

1.3 Test Code

The author determined that it would be ideal to transmit the file using C++ code as it could be easily merged into existing code. Additionally, libraries and sample code in C++ are readily available for UDP, TCP and UDT. The test code was written with simplicity in mind, and in most cases the default values were used for the relevant send and receive functions. One major consideration was this: while TCP and UDT will retry until every packet goes through by default, UDP just fires and forgets. This made it necessary to choose a maximum time to wait for each chunk of data on

the receiving end (data was chunked into 1024-byte packets). The author chose two seconds conservatively based on the fact that successful file transfers with UDP always occurred in 50 ms or less during baseline testing.

To facilitate this testing, the author wrote test code that consists of server and client applications for each of the aforementioned protocols (UDP, TCP, and UDT). The client application sends a query for a given file. Then the server application sends back the file. UDP of course does not actually connect, but the query process is still the same. These applications each have command line arguments that allow a port to be specified, and in the case of the client apps, the IP address and the number of times to repeat a given file transfer may also be specified. This "repeat" functionality made testing a breeze (other than waiting for packets to come through!). The client applications also print out the percentage of success and the average time of a successful file transfer. File transfer time was measured on the client side just before the client request was sent and ending when the file was received and written to the disk.

Each of the client apps prompts the user to enter a number between 0 and 13. Each number corresponds to a given image test file (with size ascending as the number ascends). These test files were all generated from the original TIFF image used at the military exercise event described above. The original file is one of the test files (number 1). The other file sizes were generated by using an image editor to resize the image, so each file is a shrunken or blown-up image of the original. TIFF files require the entire file to be present for the image to display, so receiving part of the image is not enough to display even part of it (unlike certain video stream formats, which can tolerate some level of packet loss). The table below shows the exact file sizes used.

Table 1. Test numbers and their corresponding image file sizes

Number	File size
0	4.4 KB
1	8.1 KB
2	29.8 KB
3	59.1 KB
4	158.6 KB
5	244.0 KB
6	375.9 KB
7	457.1 KB
8	865.8 KB
9	1.1 MB
10	6.3 MB
11	9.4 MB
12	12.6 MB
13	38.3 MB

2. NON-RF TESTING

2.1 Baseline Testing

In all scientific studies, it is important to have a control or baseline. The author chose this baseline to match typical ideal network protocol behavior. To accomplish this, two virtual machines (VMs) were run on the same Windows 7 host, one running CentOS 7 and the other running Ubuntu 14. After downloading dependencies, the author compiled all test code on both VMs. The VMs were each set up with a bridged network connection and given IP addresses on the 192.168.1.x network.

The servers were run on Ubuntu and the clients on CentOS 7. The clients were set to repeat each file request 100 times. Each of the 13 files was tested and the success rate and the average total time between client request and client receipt of the file were noted (these were calculated by the test code).

Table 2. Baseline file transfer performance using the UDP, TCP, and UDT test clients and servers running on separate VMs hosted on the same machine.

	UDP		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	100%	0 ms	100%	0 ms	100%	0 ms
8.1 KB	100%	1 ms	100%	1 ms	100%	0 ms
29.8 KB	100%	2 ms	100%	1 ms	100%	1 ms
59.1 KB	100%	2 ms	100%	3 ms	100%	2 ms
158.6 KB	92%	3 ms	100%	4 ms	100%	4 ms
244.0 KB	85%	6 ms	100%	5 ms	100%	5 ms
375.9 KB	75%	9 ms	100%	7 ms	100%	11 ms
457.1 KB	69%	10 ms	100%	9 ms	100%	12 ms
865.8 KB	58%	22 ms	100%	13 ms	100%	24 ms
1.1 MB	36%	32 ms	100%	18 ms	100%	36 ms
6.3 MB	0%	N/A	100%	74 ms	100%	138 ms
9.4 MB	0%	N/A	100%	82 ms	100%	177 ms
12.6 MB	0%	N/A	100%	99 ms	100%	238 ms
38.3 MB	0%	N/A	100%	215 ms*	100%	676 ms

*A strange thing started happening with TCP when testing the largest file 100 times. Quite often the software would just suddenly say “Killed” and stop. Most likely, some network buffer was getting filled, but for the sake of time that individual test was repeated only 30 times to get the results.

Table 2 makes it plain to see that the protocols perform very similarly with ideal network conditions and file sizes of 60 KB or less. However, UDP gets less and less reliable as file size increases, with total failure to transfer 6 MB or larger files. It is also important to note that the UDP failures came from packet loss, not receive timeouts (the application output confirmed this), meaning the two second read timeout in the UDP code did not affect the results. Both TCP and UDT achieved 100% success for all file sizes, but TCP had better performance than UDT as file size increased.

2.2 Network Emulation Packet Loss Testing

Ubuntu and other Debian builds of Linux have built-in network emulation capabilities.[4] One useful capability allows a user to introduce and percentage of random packet loss. The following command line can be used to set random packet loss to 10% (and thereby cause an average of 1 in 10 packets to be lost):

```
sudo tc qdisc add dev eth0 root netem loss 10%
```

The value "eth0" is the name of the interface to add packet loss to. The "add" keyword needs to be "change" after the command is run the first time.

This level of packet loss was set only on the Ubuntu VM, where the servers were run. The full set of tests was attempted with 10% packet loss enabled, but the author quickly learned that latency and failures go way up with this relatively small amount of packet loss. Because of the latency, the testing was limited to only 7 file sizes instead of all 14.

Table 3. File transfer performance with 10% packet loss emulated using netem. Results from baseline testing are included in parentheses for easier comparison.

	UDP*		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	52%	0 ms	100%	26 ms (0 ms)	100%	130 ms (0 ms)
8.1 KB	36%	0 ms	100%	39 ms (1 ms)	100%	180 ms (0 ms)
29.8 KB	0%	N/A	100%	76 ms (1 ms)	100%	300 ms (1 ms)
59.1 KB	0%	N/A	100%	141 ms (3 ms)	100%	414 ms (2 ms)
158.6 KB	0%	N/A	100%	232 ms (4 ms)	100%	808 ms (4 ms)
12.6 MB	0%	N/A	100%	9 s (99 ms)	100%	52 s (238 ms)
38.3 MB	0%	N/A	100%	25 s (215 ms)	100%	194 s (676 ms)

*For UDP, due to the failure rate and two second wait time for each attempt, only 25 attempts were made for each test, rather than 100. For TCP, the average for 12.6 MB was based on only 10 attempts and 38.3 MB was based on only one attempt (for the sake of saving time). For UDT, the average time for 12.6 MB was based on only one attempt and 38.3 MB was based on only one attempt. The author was not keen on waiting more than a few minutes for test results.

It is interesting to see that UDP could not transfer any complete files over 8 KB through with just 10% packet loss on the network. The other protocols still maintained their 100% success rate, but TCP slowed down a bit and UDT slowed down a LOT, at least with larger file sizes. At this level of packet loss, TCP seems to be roughly 4 times faster than UDT.

The author continued the same testing procedure with increased levels of packet loss. The transfer time averages collected were based on a reasonable number of attempts based on the file transfer times. For example, if transfer time was 500 milliseconds or less, 100 file transfer attempts were used. If transfer time was 0.5-2.0 seconds, 25 file transfer attempts were made. If transfer time was between two and ten seconds, 10 file transfer attempts were made. For 10 seconds or more, two file transfer attempts were made. In some cases only one attempt was successful and the other took so long the average time was deemed to be greater than 300 s.

Table 4. File transfer performance with 20-95% packet loss emulated using netem.

		UDP		TCP		UDT	
	Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
20%	4.4 KB	24%	0 ms	100%	75 ms	100%	216 ms
	8.1 KB	8%	0 ms	100%	209 ms	100%	368 ms
	29.8 KB	0%	N/A	100%	396 ms	100%	1130 ms
	59.1 KB	0%	N/A	100%	755 ms	100%	1821 ms
	158.6 KB	0%	N/A	100%	2024 ms	100%	2744 ms
	12.6 MB	N/A	N/A	100%	85 s	100%	250 s
30%	4.4 KB	16%	0 ms	100%	220 ms	100%	423 ms
	8.1 KB	4%	0 ms	100%	353 ms	100%	440 ms
	29.8 KB	0%	N/A	100%	1123 ms	100%	2011 ms
40%	4.4 KB	8%	0 ms	100%	687 ms	100%	721 ms
	8.1 KB	4%	0 ms	100%	895 ms	100%	1206 ms
	29.8 KB	0%	N/A	100%	3500 ms	100%	3919 ms
50%	4.4 KB	0%	N/A	100%	1570 ms	100%	1136 ms
	8.1 KB	0%	N/A	100%	4042 ms	100%	2033 ms
60%	4.4 KB	0%	N/A	100%	7635 ms	100%	2350 ms
	8.1 KB	0%	N/A	100%	9894 ms	100%	3863 ms
70%	4.4 KB	0%	N/A	N/A	> 300 s	100%	3327 ms
	8.1 KB	0%	N/A	N/A	> 300 s	100%	5173 ms

80%	4.4 KB	0%	N/A	N/A	> 300 s	100%	6681 ms
	8.1 KB	0%	N/A	N/A	> 300 s	100%	8238 ms
90%	4.4 KB	0%	N/A	N/A	> 300 s	100%	28 s
	8.1 KB	0%	N/A	N/A	> 300 s	80%	38 s
95%	4.4 KB	0%	N/A	N/A	> 300 s	60%	50 s

It is clear from this chart that UDP does quite poorly with greater packet loss, even with small files. This was expected, but the testing verified it. TCP outperforms UDT up to around 50% packet loss. TCP bogs down to the point of unusability at 70% packet loss or higher.

UDT was impressively consistent all the way up to 95% packet loss. Around 80-90% packet loss it had difficulty making and maintaining a connection, but the reconnection code was improved and files were able to continue transferring through. The nature of UUVs that surface and dive is that they will definitely lose connection and go to 100% packet loss at some point, and even on the surface the packet loss is high, so these results are very applicable to the subject problem.

The author pushed the limits of reasonability with this testing to see exactly where these protocols would fail. In real scenarios, receive attempts would typically be timed out after 5 to 10 seconds, which would effectively cut off file transfers via UDT at 70-80% packet loss or 50-60% packet loss for TCP. One interesting note is that even 10% packet loss is too much to send large files (10 MB or larger) through within a reasonable amount of time (5-10 seconds). This makes the prospect of large data transfer dependent on low loss radio links, which tend to require high bandwidth, high power and physical nearness.

3. RF TESTING

3.1 Baseline Testing with Radios

The specific manufacturer and model of the radios tested are purposely omitted here. However, the radios tested were the same ones initially used at the military exercise described in the introduction of this paper. In the lab, two separate computers were each connected to one radio each, with one acting as client (as the ground station would) and one acting as a server (as the UUV would).

As this configuration differs somewhat from the first baseline, each protocol was tested end-to-end with a single ethernet cable between the two computers just to see if there were any changes from that baseline. A few of the files were spot checked using the same procedure as the baseline network test (for easy comparison, the values from the baseline test are in parentheses):

Table 5. Baseline file transfer performance between two test computers connected via copper ethernet cables.

	UDP		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	100%	1 ms (0 ms)	100%	1 ms (0 ms)	100%	1 ms (0 ms)
8.1 KB	100%	2 ms (1 ms)	100%	2 ms (1 ms)	100%	1 ms (0 ms)

457.1 KB	80% (69%)	25 ms (10 ms)	100%	13 ms (9 ms)	100%	11 ms (12 ms)
1.1 MB	69% (36%)	45 ms (32 ms)	100%	32 ms (18 ms)	100%	30 ms (36 ms)
12.6 MB	30% (0)%	311 ms (N/A)	100%	154 ms (99 ms)	100%	204 ms (238 ms)
38.3 MB	0% (0)%	N/A	100%	412 ms (215 ms)	100%	602 ms (676 ms)

The results here are mildly interesting. UDT and UDP both seemed to perform better in this configuration, which makes sense because UDT uses UDP underneath. TCP performed a little worse, but still had a performance edge over UDT with larger files (> 1 MB). In general, the differences are not significant enough to throw out the baseline data.

Next, the same test was performed, but with radios between the computers rather than copper in order to establish a good radio-to-radio baseline. The radios were approximately 7 feet apart inside an office, with both antennae attached to the radios. The radios were set to use 1.2 Mbps bandwidth with 0 retransmissions set and 30 dBm output power. The ideal network baseline values are in parentheses for easy comparison:

Table 6. Baseline file transfer performance between two test computers connected via radios (no relay).

	UDP		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	100%	120 ms (0 ms)	100%	198 ms (0 ms)	100%	160 ms (0 ms)
8.1 K	100%	160 ms (1 ms)	100%	219 ms (1 ms)	100%	240 ms (0 ms)
29.8 KB	100%	491 ms (2 ms)	100%	551 ms (1 ms)	100%	697 ms (1 ms)
59.1 KB	100%	878 ms (2 ms)	100%	981 ms (3 ms)	100%	1.5 s (2 ms)
158.6 KB	90 (92)%	2.3 s (3 ms)	100%	2.4 s (4 ms)	100%	5.3 s (4 ms)
244.0 KB	100 (85)%	3.5 s (6 ms)	100%	3.6 s (5 ms)	100%	6.0 s (5 ms)
375.9 KB	0 (75)%	N/A (9 ms)	100%	5.6 s (7 ms)	100%	8.9 s (11 ms)
457.1 KB	0 (69)%	N/A (10 ms)	100%	6.5 s (9 ms)	100%	11.8 s (12 ms)
865.8 KB	(58%)	(22 ms)	100%	13.0 s	100%	19.2 s

				(13 ms)		(24 ms)
1.1 MB	(36%)	(32 ms)	100%	16.2 s (18 ms)	100%	27 s (36 ms)
6.3 MB	(0%)	(N/A)	100%	110 s (74 ms)	100%	151 s (138 ms)
9.4 MB	(0%)	(N/A)	(100%)	(82 ms)	(100%)	(177 ms)
12.6 MB	(0%)	(N/A)	(100%)	(99 ms)	(100%)	(238 ms)
38.3 MB	(0%)	(N/A)	(100%)	(215 ms)	(100%)	(676 ms)

The file transfer times increased dramatically with radios in the loop. Somewhat surprisingly, UDP outperformed TCP which outperformed UDT for files up to 59 KB. UDP got flakier at that point. It was also very interesting to see that UDP was just as reliable over the radios as it was on a copper network. This indicates that the radios, with full signal strength, lose very few packets, if any. The radios do add significant latency, appearing to be about 3 orders of magnitude larger than a copper connection. This corresponds quite well with the fact that the copper network connection had a maximum rate of 1 Gigabit per second (Gbps) while the radios were configured for their maximum rate of 1.2 Megabit per second (Mbps).

The radio configuration was then changed slightly to include the third relay radio that was present on the UAV payload during the military exercise originally described. The two computers were connected to client radios and the master radio ran standalone with the task of relaying all data through. Also, to more closely represent the target scenario and save a little time, testing was limited to just the 4 smallest files.

Before testing, the computers pinged each other to ensure end-to-end connectivity. The ping averages matched and were roughly 140 milliseconds. To verify that the master radio was truly acting as a relay, it was powered off while pinging was active. Indeed, the other two radios could not communicate. The regular testing process was then again followed. For ease of comparison, the radio baseline transfer times from Table 6 are included in parentheses.

Table 7. Baseline file transfer performance between two test computers connected via radios with relay.

	UDP		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	100%	242 ms (120 ms)	100%	398 ms (198 ms)	100%	303 ms (160 ms)
8.1 KB	100%	306 ms (160 ms)	100%	448 ms (219 ms)	100%	455 ms (240 ms)
29.8 KB	100%	876 ms (491 ms)	100%	1015 ms (551 ms)	100%	1492 ms (697 ms)
59.1 KB	100%	1796 ms (878 ms)	100%	1757 ms (981 ms)	100%	3.3 s (1.5 s)

These results don't reveal anything surprising. Sending the data through two radios instead of one roughly doubles the time it takes to send it.

To come even closer to the target scenario, the same test was performed, but this time with a realistic level of additional network traffic. This was accomplished by running a UUV simulator on one computer and the control station software on the other. Again, the radios were set about 7 feet apart in an office. Previous testing was done in a different office and the ping times show that there was definitely a difference between the two (one had 150 millisecond pings, and the other 50-75 milliseconds). To account for this, new baseline numbers for the two smallest files were documented with no additional network traffic and are included in parentheses in Table 8. Then the additional traffic was introduced and tests were run again.

Table 8. Baseline file transfer performance between two test computers connected via radios with relay and additional network traffic.

	UDP		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	100%	198 ms (162 ms)	100%	351 ms (317 ms)	100%	381 s (327 ms)
8.1 KB	100%	264 ms (205 ms)	100%	375 ms (358 ms)	100%	436 ms (362 ms)

The additional traffic appears to have caused a 10-20% increase in file transfer times, but success rates remained the same.

3.2 Radio Testing in Non-ideal Conditions

Testing the radio network in non-ideal conditions is most representative of the target scenario. At the original exercise, the 8 KB file easily transferred via UDP in the lab with high consistency, but as soon as the UUV was in the water and the UAV in the sky, the file would not transfer. High packet loss was surely the major factor. The first attempt to simulate this was to put the master radio far away from the others until the ping times were seriously affected. It was difficult to diminish ping times significantly within the test building, so instead the antenna was removed from the master node. This had little effect on pings when the master node was in the room with the other nodes so it was placed about 30 feet away with multiple walls between the master and other radios. The pings continued to go through, but it was obvious that there was more time between pings. With the 3 radios in those positions and no additional network traffic, the tests were run again. The numbers from location-appropriate radio baseline testing are in parentheses for comparison.

Table 9. File transfer performance between two test computers connected via radios with relay and diminished performance.

	UDP		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	44% (100%)	267 ms (242 ms)	100%	1.3 s (398 ms)	100%	1.1 s (303 ms)
8.1 KB	56% (100%)	335 ms (306 ms)	100%	1.9 s (448 ms)	100%	2.7 s (455 ms)
29.8 KB	16% (100%)	940 ms (491 ms)	100%	39.7 s (1015 ms)	100%	7.8 s (1492 ms)
59.1 KB	0% (100%)	N/A (1796 ms)	100%	130 s (1757 ms)	100%	6.8 s (3.3 s)

It is interesting to see some of the variation that occurred here. For example, UDP was more successful when transferring the 8 KB file than the 4 KB file. Also, UDT transferred the 60 KB file faster than the 30 KB file on average. This seems to indicate that the packet loss experience by the radios, while definitely non-zero, is also not constant. A qualitative comparison of this chart to the earlier charts where packet loss was introduced by network emulation indicates this performance corresponds most closely with 50% packet loss, at least based on the transfer times of TCP and UDT. It was somewhat surprising to see the success of UDP at this level of packet loss. One possible explanation is that perhaps the average packet loss was 50%, but it momentary packet loss varied between 0% and 100%. The earlier charts show this UDP performance matches most closely with the 10% packet loss values.

The radio locations in the previous test definitely caused packet loss, but it didn't quite match the target scenario because the UDP success rate was above 0%. Preferably, if TCP or UDT were to replace UDP, they needed to demonstrate success even when UDP was completely unsuccessful. The master radio was moved just a few feet further away and pings worsened by 50-100% and apparent packet loss increased. Only the two smallest files were tested because even the slightly bigger ones took too long to test at this level of network diminishment.

Table 10. File transfer performance between two test computers connected via radios with relay and even more diminished performance.

	UDP		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	8%	440 ms	100%	> 180 s	100%	13.0 s
8.1 KB	4%	279 ms	N/A	N/A	100%	13.1 s

UDP was definitely struggling this time with lots of partial file transfers. This is consistent with what was observed at the live testing described in the introduction. In nearly all transfer attempts, parts of the file would get through, but not the whole thing.

Both TCP and UDT had a difficult time establishing a connection in this test, though UDT could always connect within 5 tries. TCP only connected once in at least 20 attempts. During that one good session, two file sending attempts were made, but only one came through (the author gave up after 5 minutes of waiting on the second one, though it was

apparent from the application output that the server had sent it). For UDT, on the other hand, 10 attempts for each file were used to get a good average.

Comparing this data to the emulated packet loss charts on a copper network, it seems the UDP results corresponded with 40% packet loss while the UDT transfer times corresponded with a packet loss somewhere between 80% and 90%. So packet loss probably averaged around 80% but momentarily got down as low as 40% or less at times.

To further disrupt the radio signals, the master radio was surrounded with metal objects. Again, the master radio had no antenna for this test, but the two client radios still did.

Table 11. File transfer performance between two test computers connected via radios with relay and a third level of diminished performance.

	UDP		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	2%	339 ms	100%	10.3 s	100%	3.3 s
8.1 KB	0%	N/A	100%	> 180 s	100%	4.2 s

The author used the UDP results to determine if this was a lower success environment and thus worth testing. Transferring the 4 KB file with UDP, on 76% of the attempts, part of the file got through. But the full file only got through once. With the 8 KB file, part of the file got through on 52% of attempts, but the full file never got through.

This particular setup seems to be the closest to what was observed at the original military exercise. UDP was able to get a few packets through here and there with almost every attempt, but it could never “seal the deal” and get the whole file through.

The TCP results were been somewhat surprising in these scenarios. It severely underperformed while UDT continued to shine.

UDT was extremely impressive here. Even though UDP performed worse here than the previous setup, UDT somehow sped up and shoved through both files even faster than before.

After that last test with all 3 radios, another test was attempted but there was no throughput at all. The master radio was discovered to be powered off and would not power on. Some subject matter experts believe running a radio without an antenna can cause damage, and that seems apparent in this case.

To round out the testing, however, it was important to capture the same scenario with additional network traffic. The author had not yet realized the correlation between the failed radio and the lack of antenna, so the two remaining radio had their antennae removed for this final test. The radios needed to be 4 feet apart or closer to even connect. A test was run again with no additional traffic (results in parentheses), then the traffic was added in.

Table 12. File transfer performance between two test computers connected via radios with no relay and no antennae.

	UDP		TCP		UDT	
Fize size	Success	Avg Time	Success	Avg Time	Success	Avg Time
4.4 KB	0-40% (40%)	173 (180 ms)	100%	2-30+ s (2.6 s)	100%	1-7 s (1.2 s)
8.1 KB	0-16% (20%)	216 (227 ms)	100%	10-60+ s (> 60 s)	100%	2-9 s (1.2 s)

The additional network traffic again seemed to have little impact. This was a very hard test because the results varied quite a lot from minute to minute. Several batches of files were run through each protocol and results varied widely, thus the need for ranges in the table above. One major issue observed with TCP was that sometimes the files would go through just fine, then it would get hung up on one of them for several minutes. It was an extremely time-consuming to test. UDP was also very hard to nail down with this setup. Sometimes 25 attempts were made and several files received, but other times no files would come through. The packet loss seemed to be constantly varying, which seems to be the very nature of radios (especially ones without antennae). UDT, on the other hand, would nearly always get the file through within a few seconds (one time it took 30 seconds, but the rest of the files transferred in under 10 seconds). In addition, UDT behaved relatively consistently no matter what the most recent UDP and TCP results were.

4. CONCLUSION

It is clear from these results that file transfer with UDP is not ideal in any environment. It works reasonably well with a small amount of packet loss and small files, but is simply not reliable enough for most file transfer situations.

Before doing these tests, the author expected TCP to perform about as well as UDT. TCP indeed seems optimal for use on a copper network with high bandwidth. Unfortunately, it completely breaks down with moderate packet loss, and can hardly make a connection in a high packet loss environment. Its biggest problem appears to be that it can get bogged down endlessly. Read and write timeouts could be implemented, but the testing demonstrated that this would cause it to have a much lower success rate than UDT.

Through this testing, it became apparent that UDT is the right solution, and may be the best way to obtain reliable image file transfer in the aforementioned operational environment. The interference caused by the whizzing UAV and the all-too-close water to the UUV made it a high packet loss environment, and that is exactly the situation where UDT shines. If a connection can be established, UDT can push a file through in a reasonable amount of time, even with very high packet loss.

5. EPILOGUE

The results of this testing gave the author the opportunity to implement UDT file transfer for use in the target environment. At a subsequent military exercise running the same scenario, file transfer was nearly 100% reliable which was a complete turnaround from the 0% success rate at the previous exercise.

REFERENCES

[1] Authors Unknown, "User Datagram Protocol", *wikipedia.org*, 2018, <https://en.wikipedia.org/wiki/User_Datagram_Protocol> , (accessed 6 March 2018).

[2] Authors Unknown, "Transmission Control Protocol", *wikipedia.org*, 2018, <https://en.wikipedia.org/wiki/Transmission_Control_Protocol>, (accessed 6 March 2018).

[3] The UDT Team, "UDT: Breaking the data transfer bottleneck", *udt.sourceforge.net*, 2011, <http://udt.sourceforge.net>, (accessed 12 February 2018).

[4] The Linux Foundation, "netem", *wiki.linuxfoundation.org*, 2016, <https://wiki.linuxfoundation.org/networking/netem>, (accessed 12 February 2018).