On the Viability and Performance of DNS Tunneling

Tom van Leijenhorst, Kwan-Wu Chin and Darryn Lowe

***Abstract*— DNS tunnels are network covert channels that allow the transmission of arbitrary data using the DNS infrastructure. Users can use such tunnels to hide their communication sessions in order to bypass local security and accounting policies. Hence, it is important that we investigate the viability and performance of DNS tunneling. Our results show that clients can obtain up to 110 KB/s in throughput, and delays as low as 150ms. These results, however, incur very high overheads. In the worst case, clients generate up to 2000% more traffic!**

***Index Terms*— Domain Name System, DNS, Tunneling, Covert Channels**

I. Introduction

Covert channels allow processes to transfer information in a way that violates a system's security policies [2]. These channels do so without raising any alarms; i.e., they function within the specifications of the system they are operating in. As such, they cannot be seen as exploits, as they do not abuse vulnerabilities, but rather use a system to transmit information in a novel way. An example covert channel involves hiding data in the least significant bits of each pixel of a bitmap image. The image looks like the original and is technically indistinguishable. This implementation exemplifies a *storage* covert channel; a means of transmitting information between two processes by having one process directly or indirectly write to a storage location, which is then read by a second process [9]. Another type of covert channels concerns the exchange of data between two processes that alter and observe the timing of system resources [9], so called covert *timing* channels. A simple implementation involves a sending process executing two commands: one that takes a few seconds to complete and another that takes 20 seconds. A receiving process then interprets the former as binary "0"and the later as binary "1". When covert channels are established between processes on different machines, we refer to them as *network* covert channels [13]. Often, network covert channels are used to bypass firewalls. This, however, is not the only reason for using them. Other applications include data hiding for confidentiality (see steganography [8][3]), anonymity or to counter censorship; e.g., to bypass proxies that block certain websites.

This paper studies domain name system (DNS) tunnels, a realization of network covert channels. Thus far, research into DNS tunneling is limited. There are no academic sources that describe how DNS tunnels work. The most detailed documents are from Kaminsky's seminars [5]. Moreover, although there have been studies on detecting and preventing DNS tunneling [7], no researchers have reported their performance. To this end, this paper aims to investigate the viability and performance of DNS tunnels. Our study will be of interest to network administrators who are concerned with users using DNS tunneling to circumvent network security and accounting policies.

We have setup an experimental test-bed involving a client on the University of Wollongong (UoW)'s network and a tunneling server connected to Telstra's broadband network[1]. UoW's firewall does not allow clients to directly connect to the internet. It does, however, allow DNS traffic from local nameservers. We experimented with a DNS tunnel that is routed via the local DNS server, and also one that connects directly to the tunneling server. For both experiments, we collected the following metrics: (i) throughput, (ii) delay, (iii) signalling overheads, (iv) tunnel consistency, and (v) tunnel reliability. A summary of key results, demonstrating the practicality of DNS tunnels, is provided in Table I.

TABLE I
Summary of Results.

Metrics	*No Tunnel*	*Tunneled*
TCP Throughput	145 KB/s	70-110 KB/s
Delay	13-57ms	140-1500ms
UDP Jitter	0.3-2.5ms	8-57ms
DNS Overheads	N/A	200-2000%

The authors are with the School of Electrical, Computer and Telecommunications Engineering, University of Wollongong, Northfields Ave, NSW, Australia 2522, email: tom@vanleijenhorst.net, {kwanwu, darrynl}@uow.edu.au.
ICITA2008 ISBN: 978-0-9803267-2-7

[1]Telstra, http://www.telstra.com, is the national carrier of Australia.

The remainder of the paper is organized as follows. In Section II we explain how DNS tunneling works. After that, in Section III, we present our testbed, experiment details and the tools used to characterize the performance of DNS tunneling. We then present our results in Section IV, followed by conclusions and future works in Section V.

II. DNS TUNNELING

We first provide an overview of DNS tunneling before presenting our experimental setup in Section III. Define S_p^M as a server S of type M using protocol P. C^M is a client C of type M. Moreover, we use the following entities in our example:

- S_{DNS}^L. This is a DNS server on the local network. It is authoritative for a local zone and resolves all queries for both local and remote hostnames. Moreover, all DNS traffic from this server passes through the firewall unimpeded.
- S_{DNS}^T. A DNS server on the Internet that is running a DNS tunneling software. This server intercepts DNS requests containing covert information from clients. Moreover, it is the authoritative server for a domain; e.g., *dnstunnel.org*.
- S_{HTTP}^R. Any web server on the Internet. For our example below, we will use *www.cnn.com*.
- C^T : Tunneling client. The client that communicates with S_{HTTP}^R over a DNS tunnel.

Figure 1 shows how C^T is able to send a HTTP request to *www.cnn.com*, which is hosted on S_{HTTP}^R, using a DNS tunnel. The HTTP request is routed as follows:

1) The tunneling software on C^T intercepts the HTTP request, defined as R, and transforms it into a DNS request. It does so by encoding R using base32, thereby representing R using the letters of the alphabet A-Z and the numbers 2-7. This is necessary because the resulting encoding needs to adhere to the rules concerning valid DNS domain names. If R is too big, it is split into multiple requests.
2) The client then issues a DNS request for the domain *base32*$\{R\}$*.dnstunnel.org*, where base32$\{$R$\}$ denotes the base32 encoded HTTP request R.
3) The request for *base32*$\{R\}$*.dnstunnel.org* is first sent to S_{DNS}^L. Since S_{DNS}^L does not know the answer to the query, it proceeds to resolve the query as per DNS's resolution process. Eventually, the query reaches S_{DNS}^T, since S_{DNS}^T has been configured to be the authoritative server for the domain *dnstunnel.org* and its sub-domains.
4) The tunneling software on S_{DNS}^T decodes base32$\{$R$\}$ to reveal R, which it then forwards to S_{HTTP}^R.
5) S_{DNS}^T then encodes the HTTP response from S_{HTTP}^R using base64, as DNS records are case sensitive. The resulting encoded HTTP response is then returned to S_{DNS}^L in a DNS reply as a TXT resource record.
6) Upon receiving the DNS reply, S_{DNS}^L forwards it to C^T, which then decodes the HTTP response before passing the resulting data to the web browser.

The above DNS communication is simplex in nature. A DNS server must wait for a request from a client before it will undertake any action. Therefore, to ensure bi-directionality, a client has to continuously poll the tunneling server for information. Furthermore, due to the limited size of DNS request and response messages, a single data block may cause the transmission of multiple DNS requests and responses. Note that in Step-3, the DNS resolution process involves "walking" the DNS name hierarchy, which results in significant delays. Fortunately, after the first reply, S_{DNS}^L will have learned the address of the authoritative name server for the domain *dnstunnel.org*. Thus, allowing all subsequent DNS queries to be sent directly to S_{DNS}^T. Measurements on our testbed, see Section III, show that caching reduces delays by several orders of magnitude.

A concern with caching is that S_{DNS}^L may stop a query from reaching S_{DNS}^T since it may have the query in its cache. As shown in the example above, DNS tunneling works by creating "fake" sub-domains whereby data/requests are encoded using base32. Therefore, even though S_{DNS}^L has cached a given sub-domain, the chance that there will be another query bearing the same sub-domain is almost non-existent.

We like point out that the use of arbitrary domain-names could prove to be detrimental to the performance of DNS. For example, if S_{DNS}^L has a limited cache, massive amounts of "fake" sub-domain queries might cause the removal of older, but more relevant, cached records. As a result, S_{DNS}^L will find itself traversing the DNS name hierarchy frequently, thus prolonging the response time of applications such as web browsers.

A. Existing Implementations

To date there are many implementations, ranging from proof of concept to fully working tunnels. The following is a brief description of existing implementations.

- **DNSCat** [10]. DNSCat consists of two small programs, a server and client, written in Java. It is a fast, efficient and highly configurable cross platform

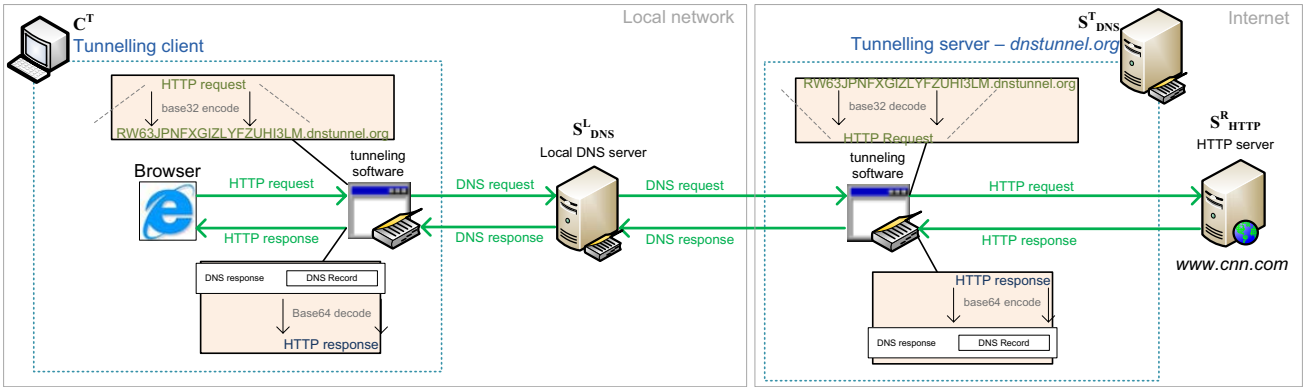

Fig. 1. DNS Tunneling.

implementation. In Linux, full support for tunneling is achieved via PPPd .

- **NSTX** [4]. Name Server Transfer Protocol (NSTX) is a hack to tunnel IP traffic over DNS that uses Linux's TAP/TUN . It requires a server running the NSTX software as well as a second server to administer the tunneling domain. This means it needs one more server compared to DNSCat. It also requires the client and server to have special kernel configurations. For this reason, and because it is less configurable, NSTX is less flexible than DNSCat.
- **OzyManDNS** [6]. This implementation consists of four Perl scripts. Two of them allow users to upload and download files using DNS. The other two form a server client pair. The server imitates a DNS server and listens on port 53 for incoming DNS requests. The client converts input to DNS requests, which are then sent to a given domain. These two scripts can be used in conjunction with SSH to create a tunnel. Users will then have to manually map ports to pass traffic through the resulting tunnel.

III. Testbed

To validate and characterize the performance of DNS tunneling, we built a DNS tunnel between two machines, one designated as the client and the other as the server. The server waits for incoming connections and supplies the client with the information required to set up the tunnel. The client is a workstation on the UoW network whereas the server is a laptop connected to Telstra's BigPond ADSL network. We chose DNSCat [10] for our experiments due to it being an open source program and because it is relatively easy to set up. Moreover, it has full tunneling capabilities using Linux PPPd, which provides benefits such as packet compression.

Figure 2 depicts the test-bed. It shows the client on the same 10Mbit network as the local UoW DNS server. The round trip time between the client and the tunneling DNS server is approximately 16ms. The client is a Linux workstation on the UoW network. It is behind a firewall and does not have direct access to the Internet by default. It does, however, have direct access to a local DNS server. This DNS server resolves hostnames within the UoW network as well as those on the Internet. The client can obtain direct access to the Internet using a web interface provided by the UoW's IT services that "opens" the firewall for a given machine. This allows us to conduct experiments that directly connect to the tunneling server.

On the server side, the DNSCat tunneling software listens on port 53. This software acts like a regular DNS server. This server is authoritative for the domain *tomvl.mooo.com*, which is supplied free of charge by FreeDNS[2]. The ADSL connection has a downstream speed of 1.5Mbit/s and an upstream speed of 256kbit/s.

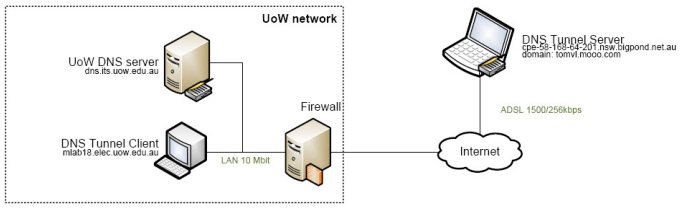

Fig. 2. Testbed.

A. DNSCat Settings

DNSCat provides various settings that influence tunnel behavior and performance. Tweaking these settings is necessary in order to find the optimal balance between overhead and low delay. Two settings are of interest:

- *Polling Interval* (P). A DNSCat client has to continuously poll the server for data. To get the fastest

[2]http://freedns.afraid.org

response times, this interval needs to be small. However, a small interval causes higher overhead, due to the high number of polling messages.
- *Minimum Send Delay* (L). This controls the interval which a DNSCat client waits before sending a request. This setting is used to throttle the number of DNS requests the client sends per minute to prevent excessive amounts of requests from being sent to the server. This delay has to be smaller than the polling interval, since that interval can be seen as the maximum send delay.

To study the impact of P and L values, we conducted preliminary measurements to determine the delays between the client, UoW's DNS server, and the DNS tunneling server; see [12] for more details. We then chose the following three sets of settings:

- **Default**. This setting is designed to minimize redundant DNS queries, hence preventing excessive DNS requests from being sent. This is achieved by setting P to 1000ms, and L to 100ms.
- **Medium**. A medium setting is chosen to be the middle ground between the slowest and fastest setting. The polling interval is much smaller than in the default setting, thereby allowing the server to send back data more frequently. The P value in this case is 200ms, and the L value is 100ms.
- **High**. To get the highest performance, both P and L are set to a low value; 80ms and 40ms respectively. Although this minimizes RTTs, there is a lot of redundant traffic, which in turn means high overheads. Note, since this setting generates many DNS requests, DNS servers might throttle or even drop DNS requests, thereby decreasing tunnel performance and reliability.

B. Test Modes

All experiments are executed in three modes: (i) control, (ii) direct tunnel, and (iii) tunneling via UoW's DNS server. The control mode involves a direct, non-tunneled connection between the client and the server, hence providing base results which we can compare DNS tunneling to. Ideally, we want DNS tunneling to have similar performance to results obtained in this mode. The direct tunnel mode entails a direct DNS tunnel between C^T and S^T_{DNS}. Theoretically, this gives the best possible DNS tunnel performance, since packets are not routed via S^L_{DNS}. The last mode tunnels packets via UoW's DNS server. This setup represents a scenario that would be used in practice to bypass a firewall.

C. Tools/Metrics

1) Ping: We use ping to measure delay and packet loss. We experimented with seven packet sizes, ranging from 16 to 1024 bytes, and sent 10 ICMP echo requests per experiment. We then record the following metrics:

1) Minimum RTT (ms).
2) Maximum RTT (ms).
3) Average RTT (ms).
4) Packet loss (%).

2) Wireshark: To investigate the overheads due to DNS tunneling, we use Wireshark[3] to capture traffic and count the number of bytes and packets generated. Here, we define overhead as any extra bytes or packets generated by DNS requests and responses.

3) Iperf: We use Iperf (v2.0.2) [1] to measure TCP throughput (KB/s), UDP jitter (ms) and UDP packet loss (%). The TCP throughput tests run for increasing lengths of time; ranging from five to 120 seconds. This gives a clear picture of throughput consistency. Additionally, it shows the time taken by a TCP connection to saturate the DNS tunnel. For UDP experiments, we use the following rates: 410, 820, 1230 and 1640 kb/s.

IV. Results

A. Ping

Figure 3 shows that the default setting introduce enormous delays; 1300ms higher than non-tunneled delays. The medium and high settings fare much better, introducing respectively 12 and 6 times more latency than a direct ping. Without tunneling, a direct ping from the client to the tunneling DNS server shows a linear increase in delay as packet size increases. None of the tunneled tests follow this trend. This is because the polling messages of DNSCat fluctuate over a wide range, up to hundreds of milliseconds using the default setting. Hence, the relatively small increase in network delay due to packet sizes becomes negligible.

One thing that is immediately apparent from Figures 3 and 4 is that tunneling via S^L_{DNS} performs less consistently than tunneling directly to S^T_{DNS}. Additionally, the average delay is approximately 150 milliseconds higher across all settings. This was to be expected, as the UoW DNS server serves many clients, both on and off the university campus. Moreover, the highly fluctuating results, observed in the UoW tunnel test with high setting, are due to packet losses. We will discuss this further in Section IV-A.1. Apart from that, tunneling via UoW's DNS server yielded high RTTs; as packets have to be routed through an extra server that is under constant load.

[3]http://www.wireshark.org

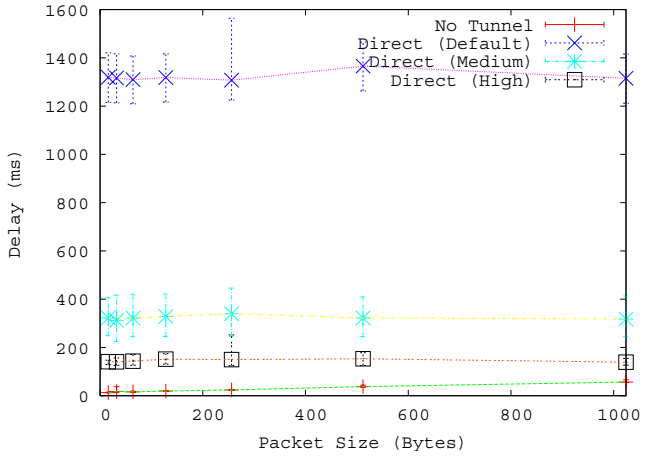

Fig. 3. Direct Tunnel.

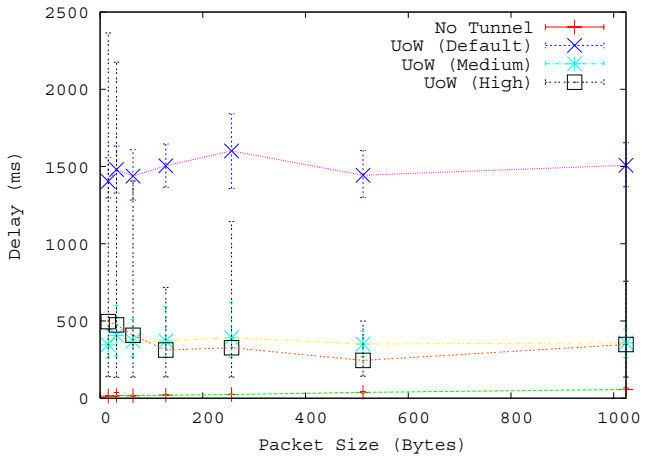

Fig. 4. Tunneling via UoW's DNS Server.

1) Packet Loss: All tests experienced a packet loss of zero percent, except when tunneling through the UoW DNS server with high setting, see Figure 5. The high packet loss may be due to UoW's DNS server implementing a denial-of-service protection mechanism that drops packets from clients that send DNS requests at a high rate. Another reason could be that the server is congested.

2) Overheads: From Figure 6, it is clear that DNS tunneling generates significant amount of overheads, even at the least aggressive setting. For example, the 16 byte ping results generate only 1140 bytes without tunneling. With the default setting, 4675 bytes of traffic are transmitted. This is an increase of 310%. The medium and high settings generate even more overheads, increasing the amount of bytes transmitted respectively by 1024% and 1967%.

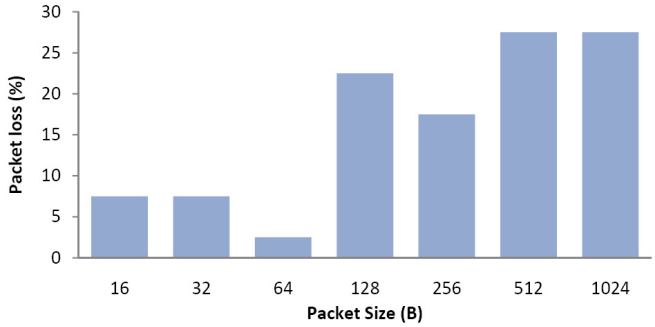

Fig. 5. Packet loss when tunneling via UoW's DNS server using high seting.

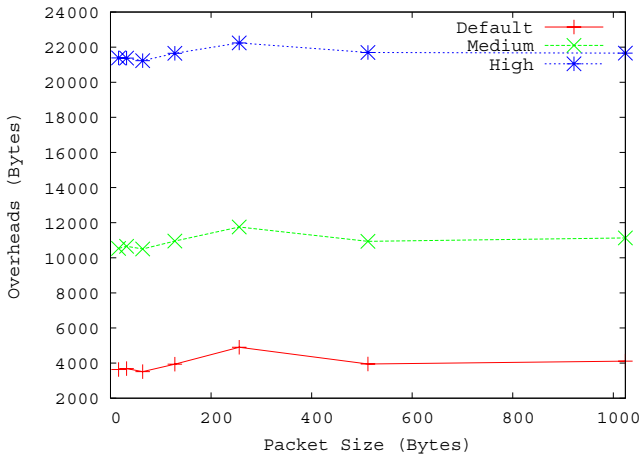

Fig. 6. Total additional bytes transmitted compared to no tunneling.

B. *Iperf*

Figures 7 and 8 show Iperf's TCP throughput. All tunneled tests show low throughput in the short duration tests. This is most noticeable when tests are done using the default setting. However, in all cases, throughput grows as the test duration increases. This is due to TCP's ack clock. Recall that TCP's congestion window (*cwnd*) grows in proportion to the incoming rate of acknowledgments. Hence, when delays are high, such as when using the default setting, TCP's *cwnd* grows slowly and the client is unable to saturate the DNS tunnel. In fact, from the figures, the client requires 40 seconds before it is able to saturate the tunnel. We see that the direct tunneled test reaches about 110 kilobytes per second as compared to the 145 kilobytes per second in the non-tunneled test. This means a tunnel can reach up to 75% of the speed of a non-tunneled connection, albeit with high overheads.

Figure 9 shows a large amount of jitter when running UDP at 50 KB/s. When we increased the transmission rate, jitter decreases to around 10-15 milliseconds. At higher bandwidths, jitter values are very similar. Even

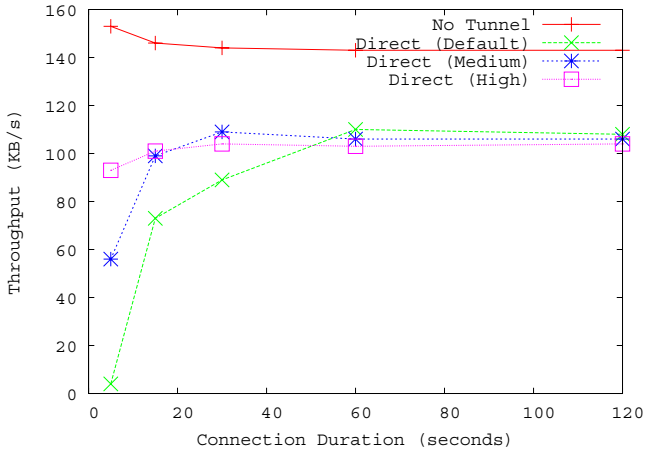

Fig. 7. Throughput (Direct Tunnel).

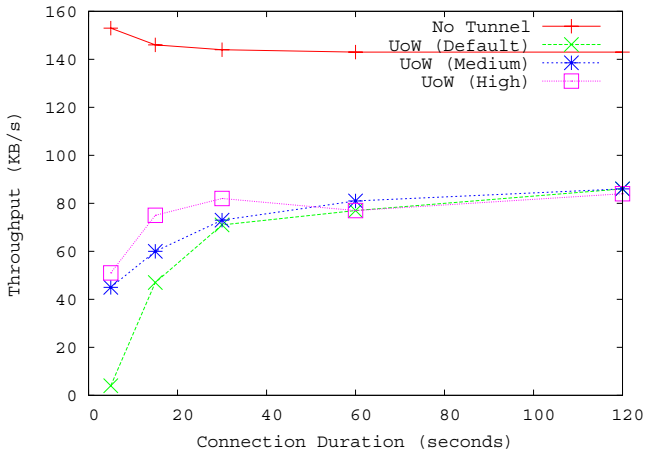

Fig. 8. Throughput from tunneling via UoW's DNS Server.

when using the high setting, which caused performance to fluctuate heavily in earlier tests, jitter is only 10 to 15 ms higher than when using a direct connection.

Table II shows the percentage of UDP packets lost at a given transmission rate. In some cases, there was no packet loss. In other cases, a higher percentage of packets were dropped. The only pattern that we could discern was that packet loss was more prevalent in tests that tunneled packets through the UoW DNS server. Additionally, tests that run at 200KB/s showed consistently higher packet losses.

V. Conclusion

In this paper we have investigated the viability and performance of DNS based covert channels. The results of our research show that DNS tunnel performance can be quite good. We have shown that tunneled TCP connections can reach up to 110KB/s in throughput, and that delays can be as low as 150ms. However, the overheads generated by DNS tunneling are significant;

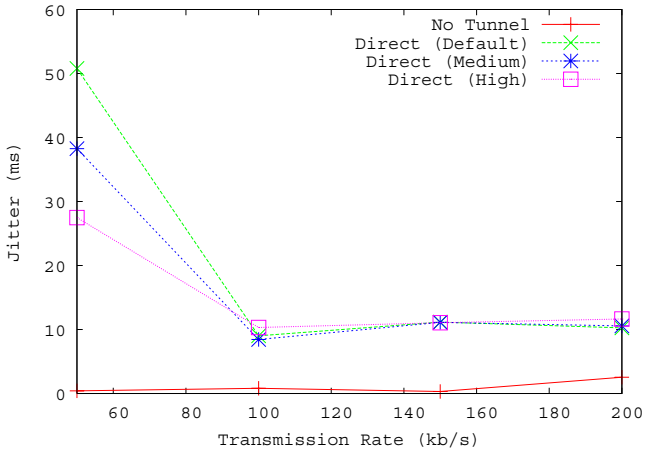

Fig. 9. Jitter: Direct Tunnel.

we measured up to 2000% increase in traffic due to overheads.

Future works include the detection and prevention of DNS tunnels. As it stands now, there are a number of detection methods [11]. The first method involves finding anomalies in the number or size of DNS requests. The second method involves packet inspection to find domain names generated using base32 encoding. This, however, requires the development of an algorithm to detect base32 encoded data.

Another area for future research is the prevention of DNS tunnels. Three approaches exist [11]. Firstly, DNS access for regular users can be limited on networks where users only use local services. If users only use a local mail and proxy server, they do not need the ability to resolve remote hostnames. Secondly, for wireless LANs with DNS-enabled demilitarized zones, DNS rules can be altered to resolve all hostnames to a local IP address until the user has logged in. Lastly, excessive or suspicious DNS requests can be blocked based on certain rules, such as the number, type or size of DNS requests.

References

[1] NLANR/DAST : Iperf - the TCP/UDP bandwidth measurement tool. http://dast.nlanr.net/Projects/Iperf/, Accessed 2007.

[2] Dept. of Defense. Trusted computer system evaluation criteria. DoD 5200.28-STD, Dec. 1985.

[3] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil. Eliminating steganography in internet traffic with active wardens. In *Information Hiding*, pages 18–35, 2002.

[4] T. Gil. NSTX (IP-over-DNS), 2007. http://thomer.com/howtos/nstx.html.

[5] D. Kaminsky. Attacking distributed systems - the DNS case study, 2004. http://www.doxpara.com/slides/BH_EU_05-Kaminsky.pdf.

[6] D. Kaminsky. DoxPara Research, 2007. http://doxpara.com.

[7] A. Karasaridis, K. Meier-Hellstern, and D. Hoeflin. Detection of DNS anomalies using flow data analysis. In *Global Telecommunications Conference (GLOBECOM)*, Nov. 2006.

TABLE II
UDP PACKET LOSS (%).

	Control	**Direct**			**UoW**		
Bandwidth		*Default*	*Medium*	*High*	*Default*	*Medium*	*High*
50	0	0	0	0	0	0	0
100	0	0	7	10.5	2.25	3.75	1.05
150	0	0	0	11.75	12.75	3	4.48
200	14	4	8.25	2.75	29	29.25	26.25

[8] S. Murdoch and S. Lewis. Embedding covert channels into TCP/IP. *Lecture Notes in Computer Science: Information Hiding*, pages 247–261, 2005.

[9] National Computer Security Center. A guide to understanding covert channel and analysis of trusted systems. NCSC-TG-030, Nov. 1993.

[10] T. Pietraszek. DNSCat, 2007. http://tadek.pietraszek.org/projects/DNScat/.

[11] M. van Horenbeeck. DNS tunneling. http://www.daemon.be/maarten/dnstunnel.html.

[12] T. van Leijenhorst. DNS Tunneling. Master's thesis, School of Electrical, Computer, and Telecommunications Engineering, University of Wollongong, Northfields Ave, NSW, Australia, 2522, Oct 2007.

[13] B. Venkatraman and R. Newman-Wolfe. Capacity estimation and auditability of network covert channels. In *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 1995.