Regular Paper

T/TCP for DNS: A Performance and Security Analysis

Kenji Rikitake,[†,††] Koji Nakao,[††] Hiroki Nogawa[†††]
and Shinji Shimojo[†††]

DNS (Domain Name System) is a mandatory subsystem of the Internet. DNS, however, has many vulnerabilities due to the complex structure. Major security incidents, such as a DDoS (Distributed Denial-of-Service) attack to the Root Servers, have been continuously and repeatedly hampering the Internet operation. While many research proposals have been made to secure the DNS by using cryptographic methods to protect the protocol data exchange, the attacks to the DNS transport layer remain effective, and the lack of transport reliability of DNS still hampers the overall security of DNS. In this paper, we first discuss the DNS overall security issues, focusing on the communication reliability such as the usage of Internet transport layer protocols. We then propose introducing T/TCP (Transactional TCP), a TCP enhancement, to the DNS transport layer. We evaluate the T/TCP by implementing the protocol to existing DNS program codes, and conclude that T/TCP is an effective alternative to enhance the overall system security by increasing the reliability of the query processing and giving another choice of configuring firewalls.

1. Introduction

One of the most important and critical subsystems of the Internet Protocol Suite is DNS (Domain Name System)[1),2)]. Many mission-critical applications depend on DNS for the domain name resolution. For example, Electronic mail messages use domain names to choose the source and destination addresses. The Web is fully dependent on the integrity of domain names to specify the appropriate servers.

The integrity of Internet depends on how DNS is managed. Improperly managed DNS database often turns out to be a persistent problem for the network administrators. For example, the address-to-domain-name resolution of IP (Internet Protocol) fully depends on DNS. The IPv4 (IP version 4) address resolution is performed by looking up the `in-addr.arpa` domain database. If entries of the database are incorrect, the whole integrity of this resolution method, is lost.

Internet systems nowadays are always under continuous and persistent attacks, as the social and business activities become dependent on Internet. All Internet services are the targets of the intruders to exploit. DNS is of no exception. DNS security is critical for the stability of Internet.

The research of DNS security, however, is not necessarily conducted on practical basis. For example, the mainstream of current research activities conducted by the `dnsext` (DNS extension) working group of IETF (Internet Engineering Task Force) are directed towards introducing the public-key cryptographic authentication called DNSSEC[3)] into the data exchange between the DNS servers and resolvers. While DNSSEC may help DNS programs to authenticate the exchanged data, the trust delegation mechanism is claimed not to be robust enough for the real-world deployment[4)], and the design is still subject to major changes[5)].

Moreover, DNSSEC does not solve the major problems which DNS administrators currently face, since they are mostly based on non-cryptographic issues, such as the transport layer security, the database management, and the program code integrity of the servers and resolvers.

In this paper, we rather focus on the transport security issues from an administrative point of view, and propose an alternative DNS transport with T/TCP (Transactional TCP)[6),7)], for improving the *overall* security of the DNS. We describe how T/TCP helps ensuring the transport security of DNS, by showing the practicality of replacing the current DNS queries of UDP with T/TCP, through the performance analysis and evaluation. As T/TCP is an extension of TCP, it preserves many advantages of TCP to UDP, such as reliable error-free data exchange, sophisticated retransmission algorithm, and the more detailed traffic control-

[†] Graduate School of Information Science and Technology, Osaka University
[††] KDDI R&D Laboratories, Inc.
[†††] Cybermedia Center, Osaka University

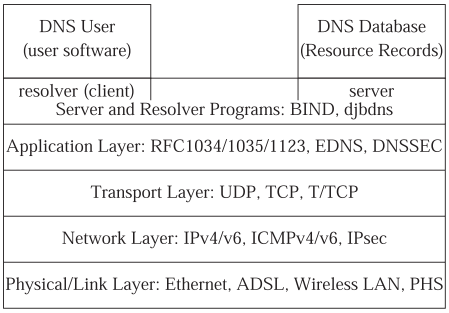

DNS User
(user software)
DNS Database
(Resource Records)
resolver (client)
server
Server and Resolver Programs: BIND, djbdns
Application Layer: RFC1034/1035/1123, EDNS, DNSSEC
Transport Layer: UDP, TCP, T/TCP
Network Layer: IPv4/v6, ICMPv4/v6, IPsec
Physical/Link Layer: Ethernet, ADSL, Wireless LAN, PHS

Fig. 1 DNS protocol structure.

lability on the firewalls.

In the later sections, we describe the DNS protocol structure and the security issues in Section 2, and analyze the transport layer behavior and requirements of DNS in Section 3. We also describe the T/TCP fundamentals and the advantages to traditional TCP in Section 4, and the evaluation results of T/TCP used as a DNS transport in Section 5. We conclude this paper on Section 6 with a discussion of the possible application fields of T/TCP to improve DNS security.

2. DNS Structure and Security

In this section, we describe the structure of DNS components and the security issues.

Figure 1 shows a simplified model of DNS component programs and protocols, described as a stack of multiple layers. In this model, DNS provides a mechanism for the users including other application programs to retrieve and update the database entries, or RR (Resource Records), located on the servers. The following is a list of description for each layer, from the bottom to the top.

2.1 Physical/link Layer

This layer consists of the physical media and the link-layer protocol of data exchange. The upper layers should be able to reliably handle and to equally treat the physical media and the link layer, which may have different latencies, bandwidths, and packet loss rates. These layers should guarantee adequate reliability of data transfer required from the upper layers.

2.2 Network Layer

This layer consists of the IP (Internet Protocol) and the related protocol components, such as ICMP (Internet Control Message Protocol). IPsec, the cryptographic authentication and data encryption capability of IP, is an important optional component. IPsec is widely used for protecting IP packets from spoofing and monitoring, though it does not guarantee the upper-layer data integrity.

2.3 Transport Layer

This layer consists of two protocols: TCP (Transmission Control Protocol)[8] and UDP (User Datagram Protocol). T/TCP[7] is a TCP enhancement for transactional data exchange. UDP provides the functionality of selecting data flow between different application services by assigning *port numbers* to each service, as well as the per-packet checksum to ensure the data integrity of each packet. TCP adds the retransmission functionality to provide a reliable communication between the application programs under data errors and packet losses. UDP has no notion of connection, while TCP has. UDP has less control on the packet filters and the proxy servers used at firewall devices. Reliability of this layer significantly affects the overall security of DNS.

DNS programs must make the UDP service port open to the external hosts to communicate with other programs. This makes the host which the DNS programs are running prone and vulnerable to external UDP-based DoS attacks, since all UDP packets must be directly handled by the DNS programs. DoS attacks to UDP ports are much easier than those to TCP ports because the attackers do not have to maintain the connection states.

2.4 Application Layer

DNS server-resolver communication protocol is collectively defined by many Internet RFCs (Request For Comments). The two important RFCs are RFC1034[1], which specifies the architecture of DNS, and RFC1035[2], which specifies the implementation details of DNS. RFC1123[9] also specifies the DNS usage as a part of the Internet host requirements.

Some of the recent research proposals include the protocol extension framework called EDNS0[10], DNSSEC, and the secure dynamic data update extension[11]. Most of the DNS programs, however, are *not capable* of performing the cryptographic extension of DNS, so in the production-level systems the overall security of DNS should be considered *without the cryptographic extensions.* We discuss the issues on DNSSEC later in Section 2.7.

Answers to the DNS queries for the Root Servers must be fit into 512 bytes, a limit imposed by RFC1035 Section 4.2.1. This limitation hampers the necessary change for the growth of DNS, such as increasing the processing performance of the Root Server network, and introducing IPv6 addresses to the Root Domain. The UDP size limitation means a restriction which only 13 IPv4 servers can be specified in the SOA answer for the Root Domain, of 1 SOA, 13 NS and 13 A RRs, 493 bytes in total. If the number of Root Servers were increased or some of the servers also announced the IPv6 addresses by the AAAA RR, the answer could easily exceed the 512-byte size limit[12], so the query reply for the Root Servers would not be able to be carried over UDP.

2.5 Server and Resolver Programs

DNS program packages belong here. The most popular package is BIND[13], which is bound to many operating system distributions. Another package called `djbdns`[14] is popular for production-level systems whose operational security is critical. A DNS program package has its own resolver library which provides programming interfaces to lookup DNS database, the database lookup programs for administrative use, the cache programs for optimizing outbound DNS traffics, and the server programs for providing the DNS database information.

Various stack-smashing and DoS attacks have been successfully made to the components of this layer. For example, a set of bugs on the BIND resolver library[15] forced major OS distributions such as FreeBSD to upgrade[16]. Another set of bugs expose vulnerabilities of BIND DNS server which allows to execute an arbitrary code or to crash the server program and/or the host operating system[17]. On the other hand, vulnerabilities of `djbdns` has not been reported yet, so we may suspect that BIND has a serious problem on the software development, if not on the quality of the code itself.

2.6 Users and Databases

An entity connected to Internet with its own domain name must maintain the set of RRs of the domain under the DNS servers of its control. DNS has a distributed network of databases, as the servers form their network of delegation. Maintaining DNS database consistency among the servers is critical for minimizing the lookup overhead and preventing illegitimate RRs to be distributed. While most of the DNS databases are read-only and manually maintained by the administrator, allowing dynamic updates on the database is being utilized by the network sites which have dynamically-configured client hosts. This update should be allowed with extreme care, since it may allow intruders to alter the DNS database contents.

2.7 DNSSEC and The Limitation

DNSSEC is considered as a primary means to secure the DNS. DNSSEC, however, does not provide the protection against DoS attacks, as described in RFC2535 Section 2.1. For preventing DoS attacks, we need to use other means than DNSSEC.

Three major authentication schemes proposed for DNSSEC: SIG in RFC2535 Section 4, TSIG in RFC2845[18], and SIG(0) in RFC2931[19]. SIG authenticates an entire DNS Zone with a public-key cryptographic system. TSIG and SIG(0) authenticates each transaction. TSIG uses a shared-key cryptographic system, while SIG(0) uses a public-key cryptographic system.

We consider that the feasibility of wide-range DNSSEC deployment is low because of the following reasons:

- TSIG is a shared-key system, and for the implementation, the key-distribution security of the secret key has to be maintained. This will not work for multilateral inter-organizational system such as the global Internet. Even using a public-key system such as SIG(0) or SIG, millions of public keys have to be maintained for each second-level domain name.
- SIG and SIG(0) uses a public-key system, which is computationally resource-intensive, and may impact the overall performance of DNS. For distributing the public keys whose digits are long enough for giving enough protection, the length of RRs will increase and may exceed the limit of 512 bytes for DNS UDP exchange. This may also hamper the DNS performance as a whole.
- The authentication model of DNSSEC assumes that the communication is performed directly between the resolvers and the servers. In a practical DNS configuration, however, the resolvers use caches and indirectly exchange information between the servers. In this cached model, TSIG and SIG(0) cannot provide the end-to-end authentication between the resolvers and the servers. A similar problem may occur when handling a replicated Zone data by DNS zone transfer.

3. DNS Transport Specification

In this section, we describe the behavior and requirements of DNS transport layer.

3.1 DNS Transport Usage

DNS has two major forms of data exchange between servers and resolvers, described in the Section 4.2 of RFC1035 [2)], as follows:

RR Queries: this occurs between the servers and resolvers. Most of the real-world traffic of the queries is over UDP, though TCP is also allowed and supported by the majority of servers.

Zone Transfer: this occurs between two servers for replication of a set of RRs to obtain redundancy against a possible server failure. This is performed solely over TCP.

We mainly discuss the RR query issues in the later sections, since the our goal is to improve the efficiency and security by experimenting an alternative transport protocol.

3.2 Choosing UDP or TCP

Section 4.2.1 of RFC1035 explicitly restricts the size of UDP queries and answers to 512 bytes. Section 6.1.3.2 of RFC1123 shows that a DNS server *must* service UDP queries and it *should* service TCP queries, and allows private agreement of servers and resolvers to solely use TCP for the queries.

Section 4.1.1 of RFC1035 specified the DNS header format. In the format, the `TC` bit is set when a server sends a truncated reply, due to length greater than that permitted on the transport. The `djbdns` behavior of the resolver which receives a UDP answer with the `TC` bit set is to reissue the request to the server using TCP [12)] all over again. This means the query reply longer than 512 bytes is always sent back by TCP, after waiting a UDP exchange solely for the notification purpose.

3.3 UDP Retransmission

DNS programs has its own retransmission and timeout algorithms for the UDP transport, since UDP does not have them. For example, `djbdns` uses the timeout algorithm [20)] of waiting 3, 11, and 45 seconds respectively for each UDP recursive queries, and terminates the operation if nothing received after retransmitting three times. This retransmission strategy works well when the packet loss rate of the network is small. When the packet loss rate is very high, however, it may cause delay of the completion of query processes, either succeeded or failed, since only four or less packets are sent for each query.

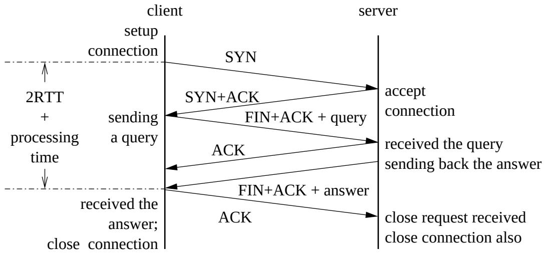


Fig. 2 Traditional TCP time line.

4. T/TCP and Traditional TCP

In this section, we describe the fundamentals of T/TCP and how it differs from the traditional transport protocols, UDP and TCP.

T/TCP is an extension of TCP. The concept model of T/TCP [6)] was proposed in 1992 and later updated by the functional specification [7)] in 1994. As of March 2003, FreeBSD and Linux operating systems have T/TCP-compatible kernels.

4.1 T/TCP Transaction Model

T/TCP is designed for a transactional use between a connection-based client-server communication, which proceeds as the following sequence:

- the client sends a request to the server;
- then the server sends back the reply;
- the exchange completes and the link is disconnected.

Some of the suggested applications of T/TCP include HTTP (Hypertext Transfer Protocol), RPC (Remote Procedure Call), and DNS queries [21)].

4.2 T/TCP and TCP Time Lines

Using traditional (non-transactional) TCP for the transactional model of Section 4.1 sequence requires two round-trip exchanges. **Figure 2** shows the time line of traditional TCP. It shows that the first of the two exchanges is solely for setting up a TCP connection, while the second one is actually used for the data exchange.

On the other hand, using T/TCP requires only one round-trip exchange, which is the same as in the UDP case. **Figure 3** shows the time line of T/TCP. It shows that the first packet sent from the client to the server carries the query data as well as the connection request. Putting the query data on the same packet for the connection request is performed by using

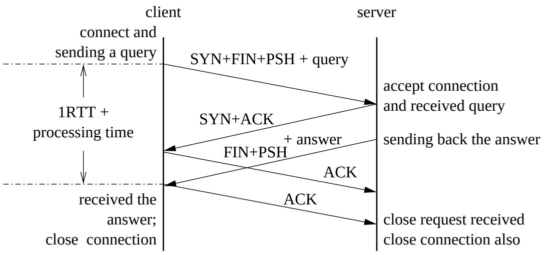

client
server
connect and sending a query
SYN+FIN+PSH + query
accept connection and received query
1RTT + processing time
SYN+ACK
+ answer
FIN+PSH
sending back the answer
ACK
ACK
received the answer; close connection
close request received close connection also

Fig. 3 T/TCP time line.

the CC (Connection Count) options of TCP, introduced by T/TCP, to indicate the support and to avoid duplicate old connections, as described in Section 4.3.

4.3 TAO Test

A TCP server needs to find out whether a received packet with the `SYN` flag set really means a new connection. Traditionally this is performed by performing the three-way handshake shown in Fig. 2, as the client and server acknowledges `SYN` request with each other.

On T/TCP, a mechanism called TAO (TCP Accelerated Open) is introduced to allow a T/TCP server to know that a `SYN` request from a T/TCP client is new, without the three-way handshake. An identifier called *connection count* (CC), a 32-bit integer, is assigned to each connection that a host establishes. The CC cache is maintained per each peer host.

Three new TCP options, `CC`, `CCnew` and `CCecho`, are defined for T/TCP as follows:

- The `CC` option carries the CC value in an initial `SYN` segment of the T/TCP client, or in the other segments if the other end sent a `CC` or a `CCnew` option with `SYN`.
- The `CCnew` option only appears in an initial `SYN` segment, when the client needs to perform the traditional three-way handshake while indicating the support of T/TCP.
- The `CCecho` option only appears in the `SYN+ACK` segment of a three-way handshake (from a T/TCP server), and echoes the received connection count value of a `CC` or `CCnew` option to tell that the server understands T/TCP.

Each T/TCP host performs the following procedure, called the TAO test, to decide whether to use TAO or not when a `SYN` request is received:

- When no cached value of CC is found for a peer host or a `CCnew` option is received, a three-way handshake is performed with the CC options and the CC values are exchanged, and the CC cache for the peer host is initialized.
- If a CC value is cached for a peer host, verification of a CC option in the received packet is performed.
  - If no CC option is found, the CC cache is cleared and the connection falls back to the three-way handshake sequence.
  - If a CC option is found with the received packet, verification for the value of received connection count by comparing it to the cached value. If the received value is greater, the `SYN` is recognized as a new one and accepted without the three-way handshake. If not, the CC cache is cleared and the connection falls back to the three-way handshake sequence.

When the TAO test fails, the data payload carried with the initial `SYN` request of the T/TCP is not passed to the application software. By performing the TAO test, T/TCP can avoid duplicate old connections without performing the three-way handshake every time.

T/TCP has the overhead for each pair of connected hosts to initialize the per-host CC cache of both on the client and the server. This initialization is, however, only required to perform for the first transaction between the two. Once the CC cache is properly initialized, the client and server pair will use the accelerated handshake sequence for the second and the later transactions, as long as the CC update is properly continued without external interference such as an intrusion attack of a spoofed host.

T/TCP CC cache consumes some amount of memory, though it is predictable and does not impact the system performance unless the available memory space for the kernel is limited. For example, on FreeBSD 4.7-RELEASE, the T/TCP-specific memory resources are listed as follows:

- a kernel 4-byte counter `tcp_ccgen` is allocated for each kernel to give CC values per each connection;
- For each host, two 4-byte variables called `tao_cc`, `tao_ccsent`, and a 2-byte variable called `tao_mssopt`, total 10 bytes, are allocated, as a per-host cache;
- For each TCP connection, three 4-byte variables called `cc_send`, `cc_rcvd`, and `t_starttime`, total 12 bytes, are allocated, as

a part of the TCP control block.

For example, when 10,000 hosts and 100 simultaneous T/TCP connections per each host are connected (1,000,000 connections total), the total number of bytes consumed is $(4 + 10 \times 10{,}000 + 12 \times 100 \times 10{,}000) = 12{,}100{,}004$ bytes. The memory block of this size is practically affordable for the PC servers which has usually a few hundred megabytes of main memory installed.

4.4 DoS Immunity

T/TCP has some immunity against simple DoS attacks which UDP does not, by performing the TAO test for each transaction. Here are some scenarios:

- For example, in case of a simple DoS attack of multiply sending the same packet, UDP has no mechanism of rejection. On the other hand, when using T/TCP, the TAO test fails from the second and later received packets, as it mandates that the CC value must monotonically increase for each transaction. The failure of TAO test leads into the protocol fallback to the traditional three-way handshake procedure. Without the completion of the handshake, the data payload in a transaction request packet will not be transferred to the application software.
- In case of a distributed DoS attack, meeting the requirement of monotonic increase of the CC value for each transaction at the server for a successful attack is highly improbable unless the sequence of received packets from the attacking hosts is thoroughly controlled.
- When the attackers use an spoofed source address of IP packets to anonymize themselves, the first CC initialization sequence of the TAO test will not be completed, and the data payload will not be accepted. If the host specified by the spoofed source address exists, the host sends an `RST` packet as the reply for a non-existent connection.

These examples show that the T/TCP does not have a weakness of UDP which blindly accepts all incoming packets. While T/TCP does not authenticate the data payload itself and may exchange a larger number of packets than UDP does in case of a successful DoS attack, the protection of the TAO test gives an advantage to T/TCP from UDP against a DoS attack.

4.5 `TIME_WAIT` State

T/TCP has another feature to shorten the time spent in TCP `TIME_WAIT` state which is to complete full-duplex closing of a connection and to allow old duplicate TCP segments to expire.

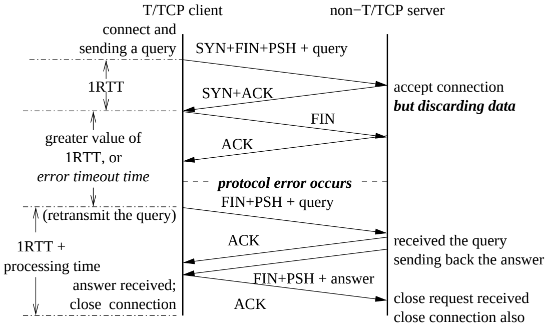


Fig. 4 Time Line of T/TCP client and non-T/TCP server on FreeBSD 4.6.2-RELEASE.

The amount of time spent in the `TIME_WAIT` is traditionally specified as twice the MSL (Maximum Segment Lifetime). On FreeBSD 4.6.2-RELEASE and the 4.7-RELEASE, the default MSL is 30 seconds, so the length of `TIME_WAIT` state for the traditional TCP is 60 seconds.

On the other hand, T/TCP specifies the length of `TIME_WAIT` as eight times the RTO (Retransmission Timeout) when the connection duration is less than the MSL. RTO is a dynamic value estimated using the measured round-trip time on the network link with a pre-defined minimum value. For example, the minimum RTO estimated by FreeBSD 4.6.2-RELEASE and the 4.7-RELEASE is 1 second. So the length of `TIME_WAIT` state of T/TCP is shortened approximately to 8 seconds, when the actual RTT of the link is much smaller than 1 second.

A smaller length of `TIME_WAIT` state means a smaller size requirement to the network control block, and an increase of number of TCP connections which a server host can simultaneously handle.

4.6 Backward Compatibility

As T/TCP is an extension of TCP, it is backward-compatible with the traditional TCP. When the server is T/TCP-aware, it can identify the client is T/TCP-aware or not, since a T/TCP-aware client will send a TCP connection request with a CC option, while a traditional TCP client does not. Figure 2 applies in the case of a traditional TCP client and a T/TCP-aware server.

A fallback procedure must be followed in case of a T/TCP-aware client and a traditional non-

T/TCP server. **Figure 4** shows the procedure and the time line of FreeBSD 4.6.2-RELEASE. In this case, the SYN cache[22)] of the server discards the data payload on the first packet, to avoid TCP SYN-flooding, a popular DoS attack which intends to consume the memory area allocated for the network control blocks. The server does not recognize the CC options either, so the first packet the client sends is treated only as a connection request. While this behavior is practically acceptable to protect the server from the possible SYN-flood attacks, it has an adverse effect of forcing the client to wait for an additional error timeout period for each transaction. Nevertheless, the backward compatibility of T/TCP to the traditional TCP is still retained.

Note that in either time line figures of Fig. 2 or Fig. 3, the meaning of the ACK bit in TCP header is left unchanged. The packet filtering rules of allowing only *established* connections of TCP are applicable to T/TCP with no need to change.

4.7 T/TCP Programming

Modifying existing network programs to be T/TCP-compatible is a straightforward task, since the protocol details are all implemented in the kernel of the operating system. For example, on BSD-derived operating systems, a flag in the include file `<sys/socket.h>` contains the flag `MSG_EOF` to show the T/TCP support. An example of the necessary changes in FreeBSD[23)] is as follows:

- On the server side, using `setsockopt()` system call for adding `TCP_NOPUSH` option to the listening socket is required to avoid unnecessary fragmentation of TCP segments.
- On the client side, the `connect()`-`write()`-and-`shutdown()` flow of system calls to initiate TCP connection and sending the query data must be replaced by a `sendto()` system call with `MSG_EOF` flag, since the T/TCP connection is implicitly established by the `sendto()` system call. The `TCP_NOPUSH` socket option is required as well.

To enable or disable the T/TCP functionality of a FreeBSD host, the administrator sets the kernel MIB (Management Information Base) variable of `net.inet.tcp.rfc1644` to 1 or 0, respectively. This value can be dynamically changed without rebooting the host.

4.8 Migration Issues

A few migration issues as follows should be considered on using T/TCP:

- The default state of T/TCP functionality is disabled in FreeBSD 4.6.2-RELEASE, as the document[7)] is still considered *experimental* in the IETF and the standardization process.
- The system administrators must be aware that all TCP-related security attacks are also applicable to T/TCP.
- Some systems with a high security concern is configured to simply ignore the TCP packets with the SYN+FIN flags to avoid revealing the protocol stack of the operating system. In this case, T/TCP packets do not get through.
- The migration should begin with the server-side first, to avoid the error-timeout issue described in Section 4.6.

The following is our perspective to these migration issues:

- The reason that IETF status of T/TCP is *experimental* is that the usage is limited to a single-query-and-single-answer transactional application. DNS database query will largely benefit from T/TCP especially when the query result no longer fits into a UDP packet because of increasing IPv6 address usage. We believe some actual deployment of T/TCP for DNS is essential, since T/TCP has already been implemented and ready to be used.
- As T/TCP is an extension of TCP, T/TCP is also prone to the security attacks to TCP. We consider, however, that the security risk imposed by the introduction of T/TCP is minimized by a proper security protection such as the TAO test.
- When a system rejects the SYN+FIN packet at all, no T/TCP connection request and the reply can be used to communicate with the system. Avoiding usage of T/TCP is a practical workaround for such a system.
- The reason we suggest to migrate first from the servers is that the programming needed for the migration is small, such as by enabling the TCP socket option of `TCP_NOPUSH` on FreeBSD.

4.9 What T/TCP Provides for DNS

We propose T/TCP as a replacement of the existing DNS UDP transport. We consider that the migration from UDP to T/TCP is feasible by the following reasons:

- T/TCP has the immunity against DoS attacks by the TAO test, as described in Section 4.3.

- T/TCP is backward-compatible with TCP as described in Section 4.6. This ensures the connectivity during the migration phase, when T/TCP and TCP DNS hosts coexist.
- T/TCP has already been implemented in the production-level server operating systems such as FreeBSD and Linux, so for these systems the migration cost is small. Using these systems as DNS caches is a practical workaround for non-T/TCP systems, which are mostly running resolvers only.
- The programming cost for migration of a TCP program is small, as described in Section 4.7. We needed less than 100 source code lines to modify `djbdns`[14] to make it T/TCP-compatible. The protocol stack implementations of T/TCP can be obtained as free software such as FreeBSD and Linux, and the detailed reference is available as a book[21].

5. Evaluation of T/TCP

In our research, we tested T/TCP as a DNS transport by modifying the program code of `djbdns` and measuring the performance and behavior. In this section, we describe the details and the results of the performed experiments.

5.1 Test Environment

The software packages chosen for the experiment are listed as follows:

- FreeBSD 4.6.2-RELEASE and the 4.7-RELEASE as the operating systems, for the stability of the T/TCP implementations:
- `djbdns` as the DNS software, for the highly-modularized structure;
- `dummynet`[24], for simulating random packet loss and high-latency links.

The modification details of `djbdns` for the T/TCP support are listed as follows:

- adding a function to set the `TCP_NOPUSH` socket flag, and an interface to `sendto()` system call for `djbdns` socket library;
- changing the DNS resolver interface functions called from the `djbdns` programs to use T/TCP instead of traditional TCP;
- changing `dnscache`, the DNS cache program, to use T/TCP for accepting the connections and external lookups;

The conditions of DNS query time measurements are as follows:

Table 1 Total elapsed time of 1,000 Sequential DNS queries to a `dnscache` server (in seconds).

	local	Ether	ADSL
RTT (ms)	≈ 0.04	≈ 0.4	60 ∼ 70
UDP	0.22	2.40	67.77
T/TCP	0.52	8.70	74.70
TCP	0.53	8.92	138.80

RTT: Round-Trip Time

- `dns_resolve()`, a DNS resolver function of `djbdns`, is called for each query. A modified version is used to perform TCP-only DNS queries. `DNSCACHEIP`, The environment variable is set to choose the appropriate `dnscache` to test.
- Each query contains a request to the NS RRs of the Root Domain (`"."`), which `dnscache` can answer solely by referring to a configuration file `root/IP/@`, with no external or internal lookup.
- Choosing the T/TCP or traditional TCP is done as explained in Section 4.7.

5.2 The Protocol Overhead

Table 1 shows the result of measuring the difference of query processing time between UDP, T/TCP and TCP for different types of links. We used a local interface, a 100BASE-TX Ethernet, and an ADSL (Asynchronous Digital Subscriber Link) of an Internet service provider.

For the local interface and Ethernet links, UDP is the fastest, since the number of packets exchanged for each query differs; 2, 5, and 6 for UDP, T/TCP, and TCP, respectively. On the other hand, the testing of the ADSL link shows that the overhead of T/TCP to UDP is only 10% of the total time, while TCP takes about twice as much as UDP does. This is consistent with the time line explanation on Section 4.2, as in the ADSL case, the RTT (Round-Trip Time) is much larger than the query processing time, and becomes a major portion of the total elapsed time.

5.3 On Allocated Connection Blocks

We performed a test on how the number of allocated sockets (connection blocks) changes between TCP and T/TCP. We performed 10,000 queries of each transport protocol by 10 concurrent processes of 1,000 sequential queries (total 10,000) each, and measured how the number of active connection blocks from the beginning of the queries. We evaluated how the `TIME_WAIT` value affects to the total processing time of the simultaneous query connections. The host used

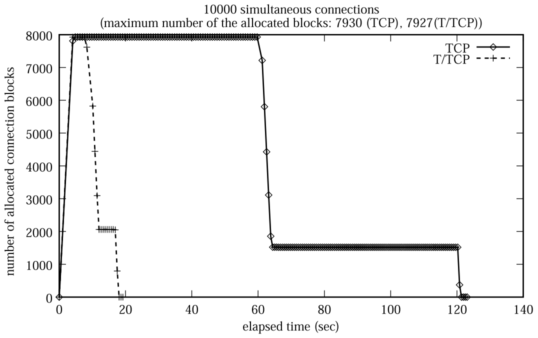


Fig. 5 How the `TIME_WAIT` value affects the number of allocated connection blocks.

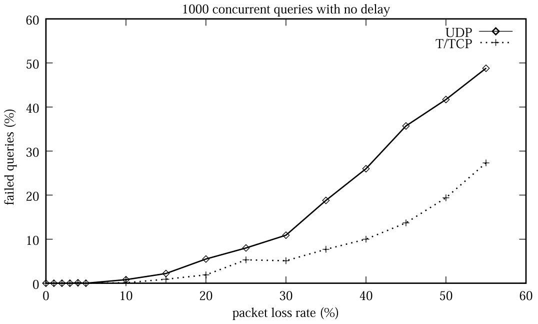


Fig. 6 Query failure rates of UDP and T/TCP for link with no delay.

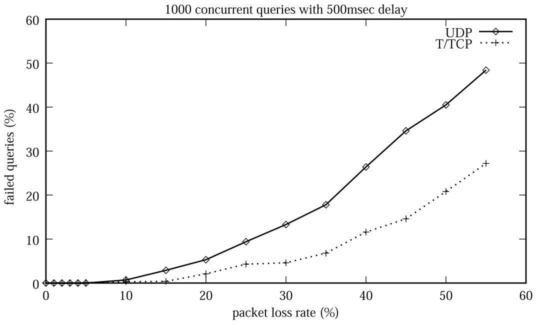


Fig. 7 Query failure rates of UDP and T/TCP for link with 500 ms delay.

for this test has only ≈ 8,000 connection blocks available to the cache program, acting as a DNS server. The server and the clients were connected through the local interface.

Figure 5 shows the result. In the beginning, the number of the allocated socket increased at the rate of ≈ 1,900 queries/sec, but after the connection blocks were used up by the query-generation processes, they waited until the first `TIME_WAIT` period expires; the suspended queries were processed later as the connection blocks became free after the `TIME_WAIT` state completion. For an application which accepts a large amount of queries, using T/TCP instead of TCP will reduce the total waiting time of queries to approximately two-fifteenth (8RTO / 2MSL $\simeq$ 8 / 60), which is shown in Fig. 5 as the length of time from the beginning of the test to when the number of allocated connection blocks starts falling from the largest value (≈ 8 seconds on T/TCP, 60 seconds on TCP). This behavior is consistent with the explanation on Section 4.5, which suggests the length of the `TIME_WAIT` value shortened from 60 seconds to ≈ 8 seconds by the protocol change from the traditional TCP to T/TCP.

5.4 On Packet Loss Rates

We performed a test to evaluate how the random packet loss rate affects the query success rates of UDP and T/TCP. Since UDP exchange takes 59 seconds as the maximum value by the retransmission algorithm in Section 3, the value of T/TCP timeout to determine the success of query is extended from the default value of 10 to 60 seconds on both the server and resolver sides. We used two hosts connected with a 100BASE-TX link and `dummynet` for simulation. 1,000 concurrent queries were conducted for each random packet loss rate value. Two delay cases, none and 500 milliseconds for simulating mobile access environment were conducted to evaluate how the delay affects the query failure rates.

Figures 6 and **7** show the results. In either delay-time case, UDP and T/TCP showed little difference for how the rate of failed queries increased as the packet loss rate did. This suggests that the 500-millisecond delay time has little effect for the measured failure rate values. We observed that on some packet loss rate values, the query failure rate values did not monotonically increased, such as those of T/TCP on the packet loss rate of 25%. We consider this behavior as a result of probabilistic bias and divergence, since in the result of a preliminary test using 100 concurrent queries, we observed much higher values of non-monotonic value changes.

UDP and T/TCP showed no failed queries when the packet loss rates ≤ 5%. As the packet loss rate increased, the difference between UDP and T/TCP results also increased, and the query failure-rate values of UDP were always larger than those of T/TCP. At the packet loss rate ≥ 30%, the values of query-failure rates for UDP is about twice as much as those of T/TCP.

The results indicate T/TCP is effective for

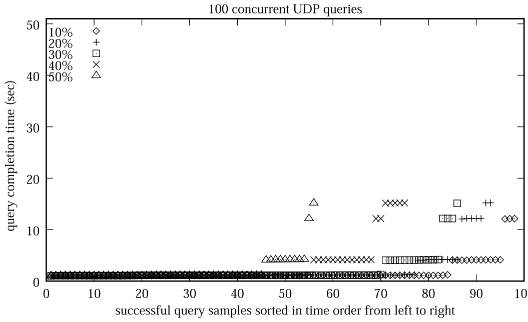


Fig. 8 Completion time of successful UDP queries on 500 ms delay link for different packet loss rates.

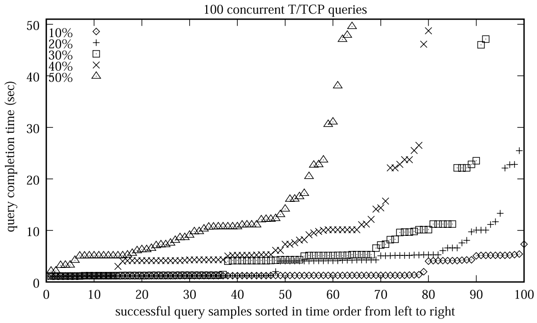


Fig. 9 Completion time of successful T/TCP queries on 500 ms delay link for different packet loss rates.

decreasing the worst-case failure rate for DNS queries in the networks of high packet loss rates.

We also evaluated how the difference of query completion time between the successful queries of T/TCP and UDP changes as the packet loss rate increases. 100 concurrent queries were conducted for each random packet loss rate value.

Figure 8 shows the numbers of successful packets for UDP categorized by the retries. It indicates that more than 90% of the successful UDP queries completes within single retry attempts. Note that the completion time for many of *failed* queries are shorter than the successful queries, due to the retransmission algorithm described in Section 3.3.

Figure 9 shows the distributions of the completion time in T/TCP queries. It indicates that the minimum completion time increases as the packet loss rate does, but T/TCP still retains the TCP characteristics of exponential distribution of the packet retransmission. Note that on T/TCP the completion time of failed query is always longer than that of the successful queries, since TCP will retry until a given timeout is reached.

6. Conclusion and Further Works

In this paper, we proposed to use T/TCP as a DNS transport, evaluated the protocol by an implementation, and showed that T/TCP is an effective alternative to enhance the overall system security by increasing the reliability of the query processing especially for mobile equipments, and giving another choice of configuring firewalls.

We list some possible application fields of T/TCP to improve DNS Security. The key issues are to avoid UDP queries whenever possible, for minimizing the risk of UDP-related attacks and to increase the rate of successful queries, while minimizing the overhead and the risk of attacks newly imposed by the T/TCP.

Mobile Equipments: DNS lookups from mobile equipments, such as from a notebook computer in a car using a cellular phone link, often fails because of the high packet loss rate. As shown in Section 5.4, T/TCP work better than UDP in such a case. Even in a lower packet loss rate, T/TCP has only 10% of query time overhead than UDP in a wide-area network environment which has the RTT of $\geq$ 60 milliseconds, as shown in Section 5.2 and described in Section 4.2. Changing the current UDP queries to T/TCP is a practical solution for mobile equipments, since it eliminates a requirement for UDP protocol stack and gives more control on the firewall between Internet and the networks of the equipments.

Inter-firewall DNS Exchange: DNS has become the only mandatorily-required UDP protocol which a firewall connected to the public Internet must support for non-private exchange. While simply prohibiting the UDP queries may work, it will increase the consumption of the server host resources, as TCP exchange requires the connection blocks inside the operating system kernel. As shown in Section 5.3, T/TCP shortens the timeout state length, which largely affects the Maximon processing capability of a server host, to 2/15 of the traditional TCP. This will reduce the average resource consumption of the server host. And as shown in Section 5.2, T/TCP is a practical solution to replace UDP DNS lookups on an ADSL or other kinds of network, which have larger latencies, and which many of the end-user Internet sites use. The zone transfer exchange of DNS will benefit from T/TCP for the fast startup and earlier closing of connections as well. If the workload increase due to the T/TCP resource consumption is of concern, the practices for Web servers are applicable to reduce the impact.

We consider two major issues should be discussed for the further works: the detailed security analysis on T/TCP, and how T/TCP affects other applications, especially those based on UDP.

Acknowledgments Our thanks go to Yutaka Miyake and Yasuyuki Watanabe of KDDI R&D Laboratories, Inc., for reviewing the draft. We also thank Mr. Tohru Asami, the president and CEO of KDDI R&D Laboratories, Inc., for supporting our research activities.

References

1) Mockapetris, P.V.: Domain names — concepts and facilities, RFC1034 (also STD13) (1987).

2) Mockapetris, P.V.: Domain names — implementation and specification, RFC1035 (also STD13) (1987).

3) Eastlake, D.: Domain Name System Security Extensions, RFC2535 (1999).

4) Gilmore, J.: AS IF: draft-ietf-dnsext-ad-is-secure-03.txt (2001). The Risks Digest Vol.21: Issue 56, `http://catless.ncl.ac.uk/Risks/21.56.html#subj7`.

5) Vixie, P.: A message of IETF `dnsext` Working Group Mailing List (2002). `Message-Id: <20021121053500.8C17737A037@as.vix.com>`, accessible from IETF Web site `http://www.ietf.org/`.

6) Braden, R.: Extending TCP for Transactions — Concepts, RFC1379 (1992).

7) Braden, R.: T/TCP — TCP Extensions for Transactions Functional Specification, RFC1644 (1994).

8) Postel, J.: Transmission Control Protocol RFC793 (also STD7) (1981).

9) Braden, R. (Ed.): Requirements for Internet Hosts — Application and Support, RFC1123 (1989).

10) Vixie, P.: Extension Mechanisms for DNS (EDNS0), RFC2671 (1999).

11) Wellington, B.: Secure Domain Name System (DNS) Dynamic Update, RFC3007 (2000).

12) Gudmundsson, O.: DNSSEC and IPv6 A6 aware server/resolver message size requirements, RFC3226 (2001).

13) Internet Software Consortium: BIND. `http://www.isc.org/bind/`.

14) Bernstein, D.J.: `djbdns`. `http://cr.yp.to/djbdns.html`.

15) CERT/CC: Buffer Overflows in Multiple DNS Resolver Libraries, CERT Advisory CA-2002-19.

16) The FreeBSD Project: Buffer Overflow in Resolver, FreeBSD Security Advisory FreeBSD-SA-02:28.resolv.

17) CERT/CC: Multiple Vulnerabilities in BIND, CERT Advisory CA-2002-31.

18) Vixie, P., Gudmundsson, O., Eastlake, D. and Wellington, B.: Secure Key Trancsation Authentication for DNS (TSIG), RFC2845 (2000).

19) Eastlake, D.: DNS Request and Transaction Signatures (SIG(0)s), RFC2931 (2000).

20) Bernstein, D.J.: User's guide to name resolution, `http://cr.yp.to/djbdns/resolve.html`.

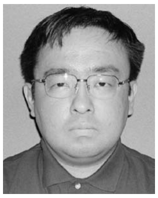
21) Stevens, W.R.: Part 1: TCP for Transactions, *TCP/IP Illustrated, Volume 3*, pp.3–158, Addison–Wesley (1996).

22) Lemon, J.: Resisting SYN flood DoS attacks with a SYN cache, `http://people.freebsd.org/~jlemon/papers/syncache.pdf`.

23) The FreeBSD Project: The `ttcp(4)` man page. Available as a part of FreeBSD distributions.

24) Rizzo, L.: Dummynet: a simple approach to the evaluation of network protocols, *Computer Communication Review*, Vol.27, No.1, pp.31–41 (1997).

(Received November 28, 2002)
(Accepted June 3, 2003)



Kenji Rikitake is a Senior Research Engineer at Computer Security Laboratory of KDDI R&D Laboratories, Inc. He is also a Ph.D. candidate at Osaka University. His research interests include DNS, computer security, and teleworking. He is a Gijyutsushi (Japanese government-licensed consulting/professional engineer) of Information Engineering since March 2001. He received the B.E. and M.E. degrees from University of Tokyo, Japan, in 1988 and 1990, respectively. He published a book *Professional Internet* from Ohmsha in 1998. He received the Best Paper Award of the IPSJ 63rd National Convention. He is a member of IPSJ, ACM, Japan Telework Society, and Internet Society.

Koji Nakao received the B.S. degree of Mathematics from Waseda University in 1979. Since 2003, he has been a General Manager in Information Security Department of KDDI Corporation and a Senior Project Manager at Computer Security Laboratory of KDDI R&D Laboratories, Inc. He is a member of IEICE and IPSJ.

Hiroki Nogawa received M.D. and Ph.D. degrees in Medicine from Osaka University in 1990 and 1997, respectively. From 1997 to 2000, he was a research assistant at Sapporo Medical University. His primary job was network design, implementation, and maintenance. He also involved in the regional network activities. Since 2000, he has been a Lecturer at Cybermedia Center, Osaka University, Japan. His research interest includes computer security, security management and social security issues. He is a member of IEICE, IPSJ, and JSSM.

Shinji Shimojo received the M.E. and D.E. degrees from Osaka University, in 1983 and 1986, respectively. Since 1998, he has been a Professor at the Cybermedia Center. His research interest includes distributed operating systems. He is a member of IEICE, IPSJ, IEEE and ACM.