ASAP: A LOW-LATENCY TRANSPORT LAYER

BY

WENXUAN ZHOU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Assistant Professor Matthew Caesar

Abstract

[1] Achieving low latency is crucial for interactive networked applications such as web browsing. According to some recent research, every round-trip time (RTT) matters a lot to those applications. Because most requested web objects are small, the transmitting latency is fundamentally dominated by the connection establishment delay in the underlying network protocols, such as Domain Name System (DNS) and Transmission Control Protocol (TCP). In this work, we introduce ASAP, a new naming and transport protocol that significantly reduces latency by shortcutting DNS requests and eliminating TCP's three-way handshake (3WH). Meanwhile, ASAP ensures the key security property of verifiable provenance of client requests which is originally provided by 3WH, yet avoids the need for a handshaking step on every connection. This approach eliminates between one and two RTTs for each connection, cutting the delay of small requests by up to two-thirds.

[1]Based on previous publication: Wenxuan Zhou, Qingxi Li, Matthew Caesar, Brighten Godfrey, "ASAP: A Low-Latency Transport Layer," ACM CoNEXT 2011. The copyright owner has provided permission to reprint. Thanks to the co-authors.

Acknowledgments

This project would not have been possible without the support of many people. Many thanks to my thesis advisor, Professor Matthew Caesar, and the other advisor on this project, Professor Brighten Godfrey. Thanks to the University of Illinois Computer Science Department for providing me with the financial means to complete this project. Thanks to Verisign, Inc. for supporting this work with an Internet Infrastructure Grant, and Barath Raghavan and Vern Paxson for helpful comments. And finally, thanks to my family and numerous friends who endured this long process with me, always offering support and love.

Table of Contents

Chapter 1

Introduction

In today's Internet, most web sites have evolved into widely distributed interactive applications. Achieving low latency is a key challenge for such applications and other online services, as even relatively small delays can affect user experience seriously, causing user frustration, and thus loss of customers and revenue [1–4]. A recent study by Google [5] quantitively proved this argument. By artificially adding delays into searches, they found that even as small as a few hundred milliseconds added delays reduced users' frequency of conducting searches measurably. The effect increased over time, after 4 weeks of 400 milliseconds added delay to a 0.74% drop, and moreover, it persisted for weeks after removal of the artificially added delay. In case this number sounds small, 0.74% of Google's search advertising revenue was $188 million in 2010 [6]. As cloud services are more and more popular, web delay will become increasingly problematic.

There are numerous causes for end-to-end delay, but one important contributor is connection establishment delay, which is rooted in underlying network protocols, such as DNS and TCP. To see why, we investigate the steps of downloading a web page and the latency overhead of each step in Section 2. In order to download a web page, before the actual data transmission, the client may first need to perform a DNS lookup, adding a variable amount of delay; and must execute TCP's three-way handshake (3WH), since HTTP is built on top of TCP, adding one round trip time (RTT) between client and server. Since RTTs are commonly on the order of 50-100 ms [7] and the client may need to initiate multiple TCP sessions in serial to download the page, the overhead from DNS and TCP together account for a significant fraction of the total delay. Our measurements (Section 2) confirm that for small web requests, this overhead is in fact dominating the total delay.

This paper studies the following question: *how can transport protocols be designed to* *perform transactions with delay as close as possible to a single RTT between the endpoints?* Moreover, we are aiming at designing a protocol deployable in today's Internet, as opposite to a clean-slate design. To achieve this goal, we revisit the two core protocols involved in connection establishment, DNS and TCP.

First, although DNS latency can be reduced with caching on either name resolver or the client's local machine, a number of lookups [8] still need to visit an authoritative DNS server (ADNS) which is relatively far away from the client. On the other hand, in many cases, ADNS may located no farther to the web server than to the client, if not closer. Based on this observation, we piggyback the first transport request packet upon DNS query; when arriving at our protocol enabled ADNS the transport packet is stripped off and sent directly to the webserver, thus cutting the latency by as much as one RTT. Although related DNS shortcutting techniques have been proposed before [9], we show that it can be done with modifications to only the client and the authoritative DNS server, without requiring changes in the global DNS infrastructure. In addition, embedding connection establishment information into a DNS request requires care to interact well with DNS caching.

Second, our protocol eliminates the need for TCP's 3WH, cutting another RTT for each connection. However, doing this in a naive fashion suffers from potential problems. First, the original motivation for the 3WH was to ensure an "idempotence" property, so old packets would not cause duplicate connections. According the end-to-end principle, we are able to argue (Section 4.1.1) this is neither sufficient nor necessary for applications' needs. Second, eliminating TCP's 3WH creates the possibility of denial of service (DoS) attacks. Most significantly, the attacker can perform *reflection and amplification attacks* [10]: it sends requests with the source address set to a *victim*'s address. The attacker thus instructs servers to conduct DoS attacks on the attacker's behalf.[1] To solve this, we design a *Provenance Verifier* (PV) which verifies a client's location, providing a certificate that the client can give to a server when opening a transport connection.

Our Accelerated Secure Association Protocol (ASAP) combines these techniques to cut connection establishment by between one and two RTTs.

[1]Even with a 3WH, an attacker can reflect packets off a server; but only a single packet will be reflected for every packet the attacker sends, limiting the damage per unit work by the attacker.

In summary, our contributions are as follows:

- ASAP is the first transport protocol that can complete requests in a single RTT without exposing significant DoS vulnerabilities.
- We show how up to one more RTT can be cut from transport connection setup by shortcutting DNS in a deployable manner.
- We implement and evaluate ASAP with wide-area experiments and microbenchmarks, verifying that ASAP offers significant improvement in latency with limited computational overhead, and is able to effectively limit eavesdropping attacks.

The rest of this paper proceeds as follows. We motivate our work in Section 2 with measurements demonstrating the causes of latency in web requests. Our protocols appear in Section 3, Section 4 and Section 5. We describe our implementation in Section 6 and experimental evaluation in Section 7. We then discuss related work in Section 9 and conclude in Section 10.

Chapter 2

Motivation

Downloading a web page in today's Internet (Figure 2.1) involves several steps, each of which contributes to the total delay. First, the browser initiates a query to the Domain Name System (DNS) to resolve the server's hostname to an IP address. This involves a query to the user's *local nameserver*, which may (if the resolution is not already cached) initiate another repeated set of queries to locate and retrieve the information from the server's *authoritative nameserver*. Second, the browser establishes a TCP connection to the server. As part of session establishment, TCP performs a three-way handshake procedure to negotiate parameters for the connection. Third, upon a successful connection, the client's browser sends an HTTP GET request for the specified object, to which the server responds. Finally, the web server uses TCP to send data back to the client. TCP uses a set of design features to avoid congesting the network, such as a fixed initial window size, slow-start, and congestion control. However, these design features also increase latency.

Each of these steps can introduce one or more round trip time (RTT) delays into the download. In addition, traversing all these steps makes the connection more subject to application-layer processing delays (during each step) and packet loss, which can further increase delays: since TCP's RTO round-trip time estimation does not yet have information about the connection's round trip time, TCP waits for a timeout of 3 seconds before retransmitting [11].

To better understand how these sources of latency affect typical web traffic, we performed a small-scale measurement study. Here, we examined latency to the top 100 most popular websites in the United States as reported by Alexa Internet on September 19, 2010. Since our focus is on connection setup, we download only the main HTML page from each site; these had average size 25 KB. We collected results from five different sites

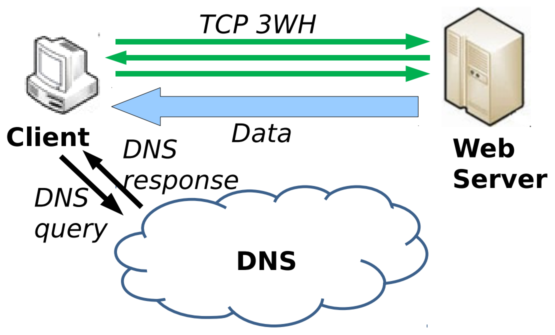


Figure 2.1: *Downloading a web page in today's Internet.*

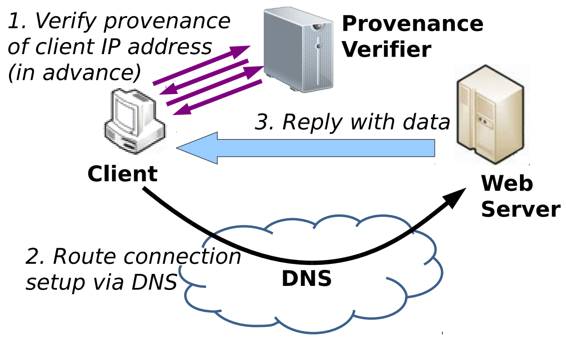

1. Verify provenance of client IP address (in advance)
Provenance Verifier
3. Reply with data
Client
Web Server
2. Route connection setup via DNS
DNS

Figure 2.2: *Overview of ASAP.*

(Wisconsin, Chicago, Urbana, Florida, and Beijing China), each running a dedicated machine that was otherwise idle, to avoid biasing timing results. For each of these five sites we performed experiments over a 24-hour period from midnight 10/24/2010 to midnight 10/25/2010, querying each site 10 times per hour using the *curl* web downloading tool. We found that on average, 41.5% of the time was spent on name resolution, 20.5% on TCP three-way handshake (3WH), 24.6% on requesting the web page, and 13.3% on data transmission. And the median latency caused by the 3WH was 72 ms, and by DNS was 101 ms. Thus, actual data transmission time is relatively small for these HTML pages (excluding their embedded objects), with the rest of delay being caused by multiple round trip delays between the client and other entities in the network such as DNS and the web server.

Chapter 3

Design Overview

Every networked application depends on, and inherits limitations of, the Internet's core protocols. While DNS and TCP have been highly flexible, they greatly inflate latency. To reduce this latency, ASAP comprises a new protocol that combines the functions of DNS and TCP. In ASAP, the client can piggyback the HTTP request within the DNS query, and the web server's authoritative DNS forwards this directly to the server. The server then responds directly to the client, eliminating two RTTs. While ASAP requires changes to existing infrastructure, these changes only need to be deployed in client software, server software, and authoritative DNS software (in a similar manner to "Akamaized" web sites), as opposed to requiring changes to other DNS servers (such as local or root DNS servers) that are not typically under administrative control of either the client or server and hence more difficult to change.

An example is shown in Figure 2.2. When the client first starts up, it performs a procedure to acquire a cryptographic proof that it resides at its IP address. To do this, it sends a request to a *Provenance Verifier* (PV). The PV is an entity that is trusted by the destination server. The PV may be located at the destination server or the server's domain, or at a third party such as a large CDN provider. The client performs a handshake, allowing the PV to verify that the client resides at its IP address. The PV then returns a *Provenance Certificate* (PC), signed by the PV, containing the client's IP address. Thereafter, when the client wishes to request an object from the server, it constructs a request containing the PC and a signed request server. Next, the client needs to determine the IP address of the server, given the server's hostname. Instead of performing a separate DNS lookup, the client piggybacks its request within the DNS lookup itself. To do this, the client encodes its request as a character string, appends that string to the hostname, and sends a DNS lookup for the resulting string. DNS routes the message to the ASAP-enabled authoritative DNS. The authoritative DNS then decodes the request, and forwards the request directly to the ASAP-enabled web server, which in turn responds directly back to the client.

ASAP's improvements could also be deployed separately, as an optimization to DNS using standard TCP, or as an optimization only to TCP.

In our transport connection protocol, the key challenge is security. Earlier approaches to eliminating the three-way handshake led to security problems. In particular, T/TCP [12,13] avoided the handshake by having the server store a "connection count" (CC) across client sessions. This led to several critical vulnerabilities. [14]. The CC could easily be guessed or eavesdropped, which (among other problems) allows attackers to mount denial-of-service attacks and reflection attacks in which the attacker spoofs its source address causing the server to flood a victim (Section 4.1.2). It also requires the server to store state for significant periods of time, which can be used as a more damaging version of a traditional SYN flood, and offers no latency improvement to clients that contact the server for the first time. ASAP avoids these problems by securely verifying source provenance without an extra RTT, and without storing extra state on servers.

In our name resolution protocol, the key challenge is deployability. Forcing all requests to arrive at a single authoritative nameserver prevents caching of requests, and makes it difficult to redirect clients to geographically-near replicas. The ASAP protocol includes mechanisms to simplify deployability and compatibility with these existing uses and features of DNS.

The following two sections describe ASAP in detail: the transport-layer design in Section 4, and name resolution in Section 5.

Chapter 4

Fast Transport Connection Establishment

In this section, we present ASAP's core transport connection establishment protocol, which eliminates the 3WH while guarding against DoS attacks. We begin by revisiting the motivation for the 3WH.

4.1 The role of the three-way handshake

In TCP's 3WH, the client sends the server a SYN packet containing an initial sequence number (ISN); the server acknowledges this with a SYN-ACK including its own ISN; and the client ACKs the server's ISN. The client can then begin sending data (such as an HTTP request). The 3WH thus adds one RTT of delay.[1]

There are two primary benefits of the 3WH. We discuss each, and how they fit into ASAP's design.

4.1.1 Idempotence

The 3WH was originally [15] designed to ensure a form of *idempotence*: if a packet is retransmitted or duplicated in the network, it should not cause a connection to be opened more than once. By challenging the client to echo back a pseudorandom number (the ISN), the 3WH verifies that the client's request is still current.

We argue that this transport-layer idempotence is neither sufficient nor necessary for applications' needs. First, it is not sufficient by an end-to-end argument: *transport-layer* idempotence does not ensure *end-to-end* idempotence. If a higher-layer entity retries the request, such as when a human clicks on a link twice after the server appears to respond

[1]The client *can* include data with the first SYN packet. But if the server acts upon this data before receiving the client's ACK, then the functionality of the 3WH is nullified.

slowly, the transaction may be executed twice. As a result, some web sites resort to imploring the human user, "Do not click Submit twice!"

Second, transport-layer idempotence is not necessary: an application that desires this property can simply perform a handshake at the higher layer. Moreover, many applications do not need it. If a server delivers a web page twice a very small fraction of the time, this is only a slight inefficiency, rather than a correctness problem.

Therefore, we argue that the benefit of ensuring idempotence in a general-purpose transport protocol does not justify its cost in added delay. ASAP will however make idempotence violations unlikely (Section 4.2.4).

4.1.2 Denial of service protection

The 3WH also lets the server s test the provenance of a client request from some source IP c. The fact that the client is able to echo the server's pseudorandom ISN, is a reliable indicator that the client is located at c.

If the server completes requests without waiting for the 3WH, two DoS vulnerabilities emerge. First, the client could fabricate a large number of requests from many spoofed IP addresses, making it difficult for the server to filter requests from a single attacking client. Second and more critically, the attacker could perform *reflection/amplification attacks* [10]: it sends relatively small requests with the source address set to a *victim*'s address. The server then sends a larger amount of data to the victim, thus amplifying the attacker's power and hiding the origin of the attack.[2]

One could hope that ISP networks perform egress filtering to block source spoofing. However, security is preserved only if *all* networks across the Internet choose to perform filtering and do so without bugs. This idealistic assumption is false in practice [16].

We conclude that the DoS protection afforded by the 3WH is highly valuable. The main goal of the rest of this section is to develop a protocol to verify source provenance without introducing an RTT delay.

[2]Even with a 3WH, an attacker can reflect a SYN-ACK packet off a server, but no significant amplification occurs.

4.2 Verifying provenance without a handshake

ASAP leverages cryptographic proof to verify the provenance of client requests without requiring an RTT delay on every connection. First, the client handshakes with a **provenance verifier (PV)** to obtain a **provenance certificate (PC)**. The PC corresponds to cryptographic proof that the PV recently verified that the client was reachable at a certain IP address. After obtaining this certificate once, the client can use it for multiple requests to place cryptographic proof of provenance in the request packet sent to servers, in a way that avoids replay attacks.

This subsection presents our basic provenance verification protocol. Subsequently, we will deal with two subtle problems: eavesdropping near the PV (Section 4.3) and mobility (Section 4.4). Fortunately, those two refinements only require changes to the process of obtaining a PC.

4.2.1 Choosing a Provenance Verifier

The PV may be any party trusted by the server. We envision two common use cases.

First, the PV may simply be the web server itself, or a PV run by its domain at a known location (`pv.xyz.com`). The first time a client contacts a domain, it obtains a PC from the PV prior to initiating the application-level request to the server; thereafter, it can contact the server directly. Thus, the first connection takes two RTTs (as in TCP), and subsequent connections require a single RTT. This technique will be highly effective for domains that attract the same client frequently (even if the specific server varies each time), such as popular web sites or content distribution networks.

Second, one or more trusted third parties could run PV services. The advantage is that a client can avoid an RTT delay for each new server or domain. The disadvantage is that servers need to trust a third party. But this is not unprecedented: certificate authorities and root DNS servers are examples in today's Internet.

The above two solutions can exist in parallel. If the client uses a PV the server does not trust, ASAP falls back to a 3WH and can use an appropriate PV for future requests.

4.2.2 Obtaining a Provenance Certificate

The protocol by which a client obtains a PC is shown in Fig. 4.1. Before beginning, the client and PV have each generated a public/private key pair (K^c_{pub}/ K^c_{priv} and $K^{pv}_{pub}/K^{pv}_{priv}$ respectively) using a cryptosystem such as RSA. The client then sends a request to the PV:

$$\{K^c_{pub}, d_c\}$$

where d_c is the duration for which the client requests that the PC be valid. The PV replies with the PC:

$$PC = \{K^c_{pub}, a_c, t, d\}_{K^{pv}_{priv}}.$$

Here a_c is the source address of the client, t is the time PC becomes valid, and d is the length of time PC will remain valid. The PV sets t to be the current time, and sets d to the minimum of d_c and the PV's internal maximum time, perhaps 1 day (Section 4.4).

To verify provenance (Section 4.1.2), it is sufficient to use a single UDP message: while it doesn't prove *to the PV* that the client can receive messages at a_c, the client can only use the PC if it is able to receive it at a_c. However, the PV itself is somewhat better protected from DoS by using TCP, especially with SYN cookies, since this ensures that the PV checks for address spoofing before it performs cryptographic functions (which are expensive relative to sending a SYN-ACK).

4.2.3 Sending a request

Once the client c has a current PC for its present location, it can contact a server and include the PC in its request in order to bypass the 3WH.

However, a naïve implementation including only the PC would allow anyone who obtains the PC (an eavesdropper or a malicious server that c contacts) to use it to induce any server to send data to c. To guard against this attack, c also constructs a **request certificate (RC)** encrypted with its private key:

$$RC = \{hash(meta, data), t_{req}\}_{K^c_{priv}}.$$

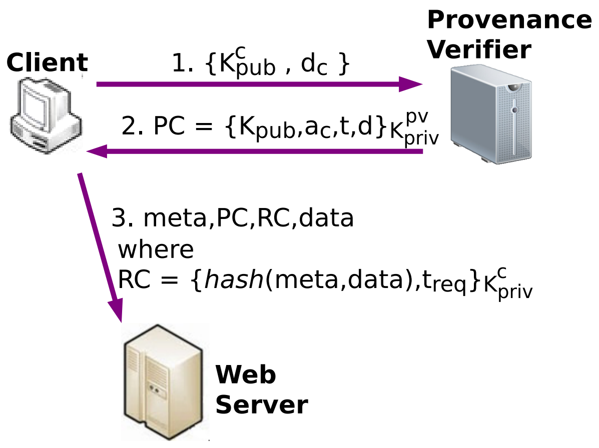

Provenance Verifier
Client
1. $\{K^c_{pub}, d_c\}$
2. PC = $\{K_{pub}, a_c, t, d\}_{K^{pv}_{priv}}$
3. meta,PC,RC,data where RC = $\{hash(\text{meta},\text{data}), t_{req}\}_{K^c_{priv}}$
Web Server

Figure 4.1: *Key messages in the basic ASAP transport protocol. Obtaining a Provenance Certificate when acquiring a certain IP address (messages 1 and 2); opening a transport connection (message 3).*

Here *hash* is a secure hash function, *meta* is the message metadata (source and destination IP address and port, protocol number, initial sequence number), *data* is the application-level data (such as an HTTP request), and t_{req} is the time the client sends the request.

The client can now open a transport connection to the server with a message of the form:

$$meta, PC, RC, data.$$

Upon receipt, the server verifies validity of the request. To do this, the server must already know the public key of each PV that it trusts. It determines whether the PC is valid for one of these[3] by checking that it decrypts correctly, the current time lies within $[t, t+d]$, and a_c matches the source address. If so, it uses the client's public key K^c_{pub} from PC to check that RC decrypts correctly, the hash value in RC matches $hash(meta, data)$, and the time t_{req} is recent, e.g., within the last 5 minutes. (This timeout only needs to be long enough to cover most clock inaccuracy, which is on the order of hundreds of milliseconds on NTP-enabled hosts, and packet transit time.)

If all these tests pass, then the request is accepted and the connection proceeds as in TCP after the 3WH: *data* is passed to the application, and data (e.g., a web page) may be sent immediately back to the client. Thus, the client can receive results within a single

[3]We expect the number of trusted PVs to be small, but to avoid iterating through each, the message could simply include a short identifier for the relevant K^{pv}_{pub}.

RTT.

4.2.4 Errors and idempotence

In the protocol above, a number of errors can occur. Clocks at the PV, client, or server could be out of sync so that certificates are rejected, or the PV chosen by the client may not be trusted by the server. In these cases, we can simply fall back to a regular 3WH.

Another error case is *sending of duplicate requests*, most commonly when the client retransmits after a timeout. There are several cases. If the server has not received the request yet (e.g., the first was dropped), then it proceeds as in the protocol above. If the server has already received the request and the connection is still active, then it can realize this and simply ignore the duplicate. If the server has already received the request but the connection is closed, it may believe the request is new, and pass the data to the application.

The last case violates idempotence (Section 4.1.1). It is unlikely to occur in benign cases: closing the connection requires receipt of a FIN packet from the client, which it would only send *after* a retransmission of the original request, and even after that the TCP stack enters the TIME_WAIT state before finally clearing the connection after a timeout. However, the server may receive the connection request after that due to a replay attack. The server may choose to guard against such cases by remembering hash values of recent requests. The server needs to maintain this state only for the RC timeout period (5 minutes as specified above).

4.3 Eavesdropping attacks and defense

Thus far, determining the validity of a client's address hinges on the client being able to receive messages at a given address a. But in fact, this depends not only on the client's location, but also on *from where the message is sent.* If an attacker can eavesdrop on any part of the path $PV \rightsquigarrow a$, then it can obtain a PC for a. In the same way, a client can induce a TCP server s to send data to a if it can eavesdrop on any part of the path $s \rightsquigarrow a$. Therefore, if the PV is colocated with the server, ASAP's security (in this sense) is equivalent to TCP's.

But if a single PV is used by servers in many locations, the attack could be more damaging. Consider, for example, a PV run by a globally-trusted third party. An attacker who can eavesdrop on the PV's network providers can obtain PCs for any address, and these PCs are valid at every server in the Internet!

To defend against this attack, we use the following technique: the organization running the PV service places PV servers in several diverse locations; the client must successfully handshake with all of them to obtain a PC. In this case, the attacker would need to eavesdrop on *all* paths $PV_1 \rightsquigarrow a, PV_2 \rightsquigarrow a, PV_3 \rightsquigarrow a$, traversing diverse geographical locations. Intuitively, the attacker has either compromised many networks or is in fact physically close to a. Note that this change only requires modification of the protocol between the client and the PV, with no changes to the PC format or interaction between client and server.

But how many PVs are necessary, where should they be placed, and how much (quantitatively) can the eavesdropping attack be limited? We give two answers to this question.

First, in the Appendix, we prove that $O(k \log n)$ PVs are sufficient (though perhaps not necessary) to defend against an attacker that can eavesdrop on k nodes in an n-node network, even if the attacker can choose those nodes after knowing the PV locations. Specifically, we show that for nearly all possible PV placements, for every client c, either: (1) the attacker cannot impersonate c in ASAP, or (2) the attacker might be able to impersonate c but even in TCP, the attacker could fool at least half of the Internet into believing that it is c.

4.4 Dealing with mobility

In our basic protocol, PCs remain valid for a fixed timeout, such as 1 day. The fixed timeout may be suitable for many applications. In some cases, however, a client may be authorized to use a source IP for only a short duration, perhaps because it is mobile, and we may wish to bound the duration of invalid use of a PC.

Suppose a client is authorized to use an IP address only for time period $[t, t+T]$. The difficulty is that the PV does not know T. A simple approach would be to pick a fixed PC timeout d. This results in a tradeoff: d is short and the client has to contact the PV frequently (T/d times); or d is long and invalid use of the PC could last arbitrarily longer than T.

However, we can do much better with an adaptive expiration time. The following protocol guarantees that the duration of invalid use is $\leq T + O(1)$ with only $\log_2 T - O(1)$ requests sent to the PV. The protocol extends our basic client-to-PV protocol (again without any changes to the PC format or client-to-server protocol). The first time the client contacts the PV, the PV issues a certificate for a short duration d_0, e.g., 30 minutes. Just before this PC expires, the client requests another, with a *refresh* option where it includes its old PC in the message to the PV. The PV verifies that the old PC is valid and current, and if so, issues a PC with duration *twice* that of the old PC. This is then repeated, and guarantees that the client will be certified to use the address only during $[t, t + \max(2T, d_0)]$ with $\lceil \log_2(T/d_0) \rceil$ requests sent to the PV. In practice, there would likely be a maximum duration as well (e.g., 1 month). Note that the PV remains stateless.

4.5 Additional security properties

Replay attacks. Assuming the private keys and the secure hash function are not compromised, an eavesdropper that has heard every message has few options for replay attacks. Specifically, even knowing a PC for a client, the attacker cannot create a novel valid RC. It can only replay existing RCs for a limited amount of time (e.g. 5 minutes, using the timeout above), which can be filtered by the server (Section 4.2.4).

DoS attacks on servers. ASAP introduces cryptographic overhead on servers, which could be exploited to perform DoS attacks. The worst case would be that a large number of hosts present valid PCs but invalid RCs to the server. In our implementation (Section 6), the server would be forced to perform one RSA 1024 verification, and one RSA 512 verification before declaring the request to be invalid.

Although ASAP increases the amount of work the attacker can force the server to do with a single packet, this attack has an easy defense: if a server detects that it is under attack and cannot handle the rate of requests, it can simply fall back to standard TCP handshaking. The attack thus only causes a single-RTT increase in latency rather than a service outage; thus, the attack may have limited value to attackers.

We also note ASAP's extra computation overhead is partially compensated for by slightly reduced resources: (1) the server sends and receives one fewer packet than in the 3WH, and (2) as in TCP with SYN cookies, it avoids storing state for half-open connections. Finally, ASAP's computational overhead could be decreased in future implementations with a faster cryptographic algorithm, such as elliptic curve cryptography (ECC) [17].

Key compromise. If the client's key K^c_{priv} is compromised (e.g., if the client is infected with a bot) this will expose *only the client* to DoS attacks. An attacker can then impersonate the client and mount reflection attacks, but only directed to the compromised client.

The more serious problem is if the PV's keys are compromised. If the PV is run on a per-domain basis, it can simply discard its old keys and create new ones. If the PV is a trusted third party, servers that trusted it will have to be made aware of the compromise and remove the PV from their list of trusted PVs. Similar problems are also encountered in web Certificate Authorities [18] and similar solutions apply here.

Privacy. ASAP clients might be easily tracked across requests and across locations, since each request includes the client's public key. However, the client can simply change its (arbitrary) public key when it changes its IP address and obtains a new PC, thus providing privacy that is essentially equivalent to today.

Chapter 5

Fast Name Resolution

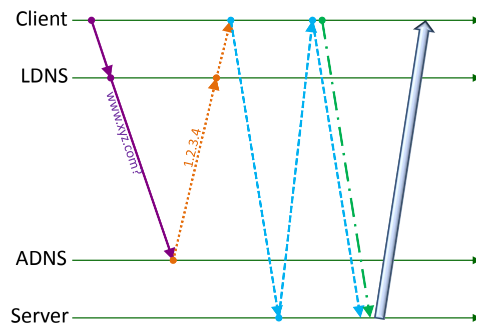
To reduce delay, ASAP piggybacks transport connection establishment atop the DNS lookup process. The intuition is that once the client's request reaches a DNS server that knows the web server's IP address, forwarding the message directly will be faster than going via the "triangle route" from the DNS to the client to the server, which occurs today. In general, this shortcutting will save up to 1 RTTs, depending on the location of the DNS server that knows the server's IP address.[1]

In realizing this idea, our key goal is deployability. The procedure described here requires changes only at resources under control of the client and the server interested in using the protocol: the client, the server, and the authoritative DNS server (ADNS).

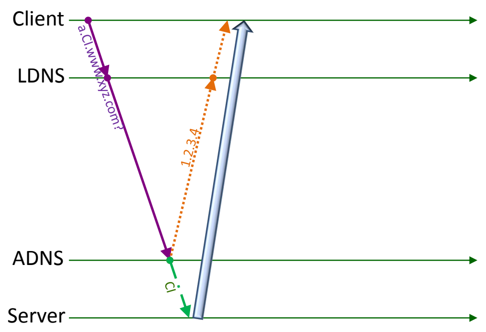
5.1 Basic protocol

In ASAP (Figure 5.1b), if the client does not have the server's IP address, it first constructs a DNS query. In the query, it inserts connection establishment information CI. Specifically, CI is a sequence of bytes encoding $meta, PC, RC, data$ as described in Section 4.2.3. ASAP does not modify the format of or add fields to the DNS query. Instead, we encode the connection establishment information into the hostname field of the DNS request, concatenated with the hostname being looked up. For example, if the client is looking up `www.xyz.com`, it would generate a DNS request for $a.CI$`.www.xyz.com`, where a is an arbitrary character which will not appear in the normal name (e.g., ASCII code 13), used by ASAP to determine if the request is from an ASAP-enabled client or a legacy client.

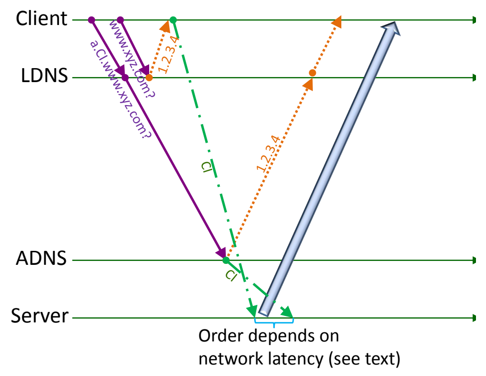
[1]The Internet occasionally violates the triangle inequality [19]. This could either lessen or heighten ASAP's benefit. Our evaluation will show shortcutting offers significant improvement in practice.



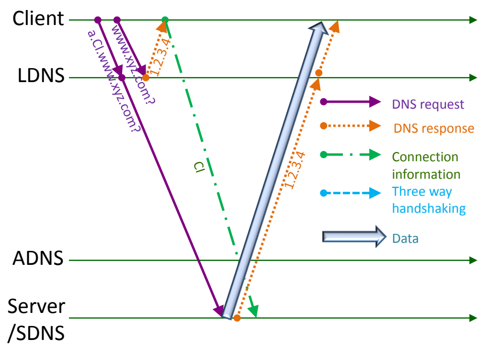
(a) TCP



(b) Basic ASAP



(c) With DNS caching support



(d) With server-side DNS

Figure 5.1: *Timeline for TCP and three variants of ASAP.*

Since CI is unique, the local DNS (LDNS) will not have the name cached, and will route the client's query towards the ADNS for `xyz.com`. The ADNS (which supports ASAP) strips off the CI field and sends CI, which carries the address of the client, spoofed by the DNS server, to the server. The server then responds directly back to the client with the requested object using the ASAP transport protocol (Section 4). Since the LDNS is still waiting for a response, the ADNS also returns an `A` record mapping $a.CI$`.www.xyz.com` to the web server's IP address. This response is given a small TTL since caching it will be useless for future requests (as CI is unique for each request).

While Extension Mechanisms for DNS (EDNS0) [20] enable the use of long names, to be compatible with older DNS servers, the total size of the hostname field in the query should remain below 253 bytes. In our implementation $meta, PC, RC$ occupies a total of 212 bytes, leaving 41 bytes for the actual hostname and application data. This would be enough space for retrieving HTTP objects that do not require long parameters from the client. Specifically, the server can choose short hostnames and compressed pathnames for web objects (i.e., a URL like `www.xyz.com/bX4r` could be mapped by the web server to a longer pathname). However, some HTTP requests may require long client-specific parameters. At a minimum, the server will receive the first line of the HTTP request, which includes the URL [21]; conceivably, servers can decide individually whether they have enough information to act on the request (e.g. setting parameters to default values), or must wait for more data. Moreover, cryptographic algorithms using smaller key size for an equivalent amount of security, such as ECC [17], could be leveraged to provide more room for the application data.

Making this scheme practical requires solving two more problems below.

5.2 Handling DNS caching via multiple queries

Unfortunately, embedding CI into the requested hostname presents problems for DNS caching, which DNS uses to improve latency and scalability. In the basic protocol described above, ASAP prevents caching because CI, and thus the hostname, varies across every connection. If the client's LDNS (where we are particularly interested in caching) supports ASAP, then it can simply strip off CI from the name. However, in practice, ASAP may not be supported at all LDNS servers.

To address this, ASAP clients also generate a second DNS request (Figure 5.1(c)), for the *original* hostname (`www.xyz.com`), to cause that hostname to be cached at the LDNS. The cached entry can then be used by ASAP clients (to avoid intermediate DNS lookups), as well as non-ASAP clients (to directly lookup the remote server). As an optimization, upon receiving a response to the second DNS request, if the ASAP client has not yet received a response from the server, the ASAP client immediately sends its CI request directly to the server's IP address. This can speed ASAP connections for cases where the remote server's IP address is locally cached. Note that this optimization can cause multiple copies of the request to reach the server; but this case is functionally equivalent to the client retransmitting after a timeout, which the ASAP server (like a TCP server) must handle anyway, by ignoring the duplicate.

5.3 Reducing latency with server-side DNS

One remaining shortcoming of ASAP is that all queries must traverse a single ADNS. This increases load on the ADNS, and increases latency for sites that replicate content across the wide area to place it near users.

To address this, ASAP embeds some DNS functionality into the web servers. In particular, each ASAP-enabled server can run a Server DNS (SDNS), which performs similar functionality to the ADNS but is co-located with the server. When the ADNS is first queried, it responds to the LDNS with an NS record mapping `www.xyz.com` to the SDNS, and an A record mapping the SDNS to the server's IP address. Thereafter, the LDNS will map requests of the form $a.CI$`.www.xyz .com` directly to the SDNS, and thus to the server, avoiding the ADNS.

The SDNS can be implemented as an extension to the web server that decapsulates the ASAP query, to avoid running an additional process. Also note that the service provider can perform standard load balancing and redirection by choosing which SDNS IP address to return (e.g., one physically near the requesting client).

Chapter 6

Implementation

Fast transport connection establishment: We implemented two versions of ASAP's transport layer. First, we built an application-layer implementation using UDT (version 4.8) [22], a reliable and congestion aware UDP-based transport protocol. We modified UDT to implement our fast transport connection establishment protocol (Section 4) by (a) adding a new *connect* interface on the client side that, in addition to taking the socket descriptor as input, also takes the domain name, certificates, and application data; (b) adding a new *pre-fetch* interface on the server side, which lets the server begin immediately transmitting data upon establishment of the connection. For ASAP's cryptographic operations, we used SHA-256 for *hash* and 1024-bit RSA, as implemented in OpenSSL [23] version 1.0.0c.

To compare more precisely to TCP, we built a second implementation of ASAP within Linux kernel 2.6.38.4. The client sends a special SYN message carrying connection information and data; the server's kernel validates provenance and passes the data to the application immediately. Note that this special SYN is compliant with TCP. By adding an ASAP option in the kernel and setting it to true, the special SYN message is allowed to have a data section, instead of only a header as in typical SYN messages. Since we could not directly make use of OpenSSL in kernel space, and since no asymmetric cryptographic algorithm is included in the standard release of the Linux kernel, we ported and modified an RSA patch [24] for Linux kernel 2.6.21, and incorporated the cryptographic part of the design as a kernel module.

The fast transport connection implementation may be run alone, or to further reduce delay, may be run jointly with our fast name resolution implementation.

Fast name resolution: We implemented ASAP's fast name resolution as a set of extensions to the Unbound DNS server version 1.4.8. This code forms the foundation for the ADNS and the SDNS operations. We run unmodified Unbound as the LDNS and intermediate DNS servers in our experiments.

Chapter 7

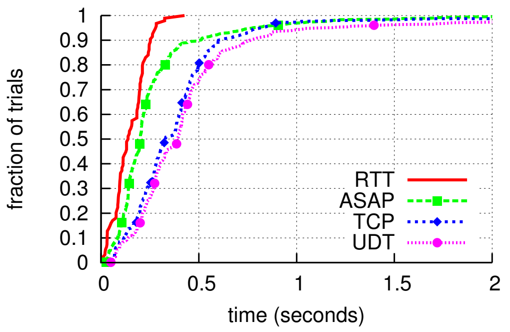
Evaluation

We evaluated our implementation of ASAP in a PlanetLab deployment, a local deployment with emulated latency, and microbenchmarks (Section 7.1). Overall, we find that ASAP can reduce transmission time by up to two round trips, significantly reducing latency of short web traffic (Section 7.2). However, ASAP also has computational overhead for cryptographic processing; we show this is manageable (Section 7.3).

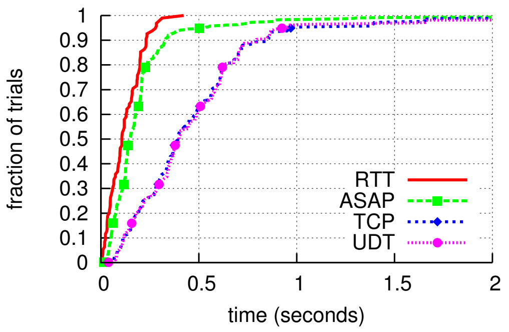
7.1 Methodology

We chose 24 representative PlanetLab nodes to act as clients and servers: 18 domestic nodes (UIUC, UCLA, UPenn and so on) and 6 international nodes (Brazil, New Zealand, Japan, Singapore, Taiwan, Zurich). The RTTs among these 24 nodes range from 3 ms to 440 ms. Each client was assigned a LDNS, in the same site as the client. Similarly, the server's ADNS was placed at another PlanetLab node located in the same site as the server. We also tried placing the ADNS server on more distant nodes to see how much influence this variation causes to the performance of our design. Unless otherwise mentioned, the client downloads 11.18KB of data, the median size of data downloaded from per single host per connection according to [25], during the connection.

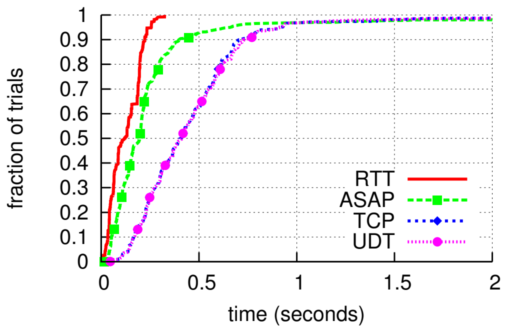
We implemented the transport component of ASAP as extensions to UDT. We compare against an unmodified implementation of UDT, and unmodified TCP. We implemented the name resolution component of ASAP as extensions to the Unbound DNS server, and use an unmodified copy of Unbound as a baseline. We targeted a design with low complexity, resulting in an implementation with relatively few lines of code (less than 3000 for name resolution, transport, client/server, and provenance verification functions).



(a)



(b)



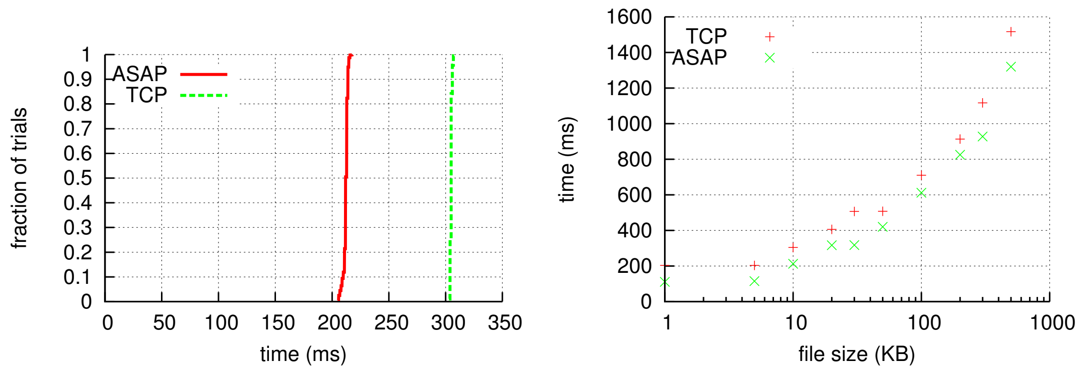
(c)

Figure 7.1: *Overall download time, when DNS caching is (a) enabled, bypassing the ADNS; (b) disabled, with the ADNS colocated with the server; and (c) disabled, with the ADNS in a random location.*

7.2 Latency reduction

In this section, we study the latency of ASAP. We define the *latency savings ratio* (LSR) as the time it takes ASAP to download an object divided by the time it takes today's Internet to download the object (using UDT or TCP).

Overall latency: We first evaluate the complete ASAP protocol. We consider two separate cases: (a) the server's IP is already cached on the LDNS, and (b) the server's IP is not cached, requiring a lookup to traverse to the server's ADNS. We achieve the first case by sending an initial request to "warm up" the cache before collecting results. For the first case (Figure 7.1a), the latency savings ratio is similar to the transport-only experiments. For the second case (Figure 7.1b), ASAP can save up to two RTTs, resulting in more



(a) CDF of total delay for 11.18 KB download (b) median total delay for sizes 1KB to 500KB

Figure 7.2: *Latency improvement of ASAP's transport layer, kernel implementation, excluding name resolution.*

significant latency reduction. Finally, since the ADNS may not always be near the server, we perform an experiment where we place the ADNS server at a random site (Figure 7.1c). We find that ASAP can still reduce latency in this case: even if the ADNS and server are not colocated, it is generally faster to go directly from ADNS to server, rather than from ADNS to client to server.

Transport latency: To understand the benefits of ASAP's connection setup procedure, we microbenchmark only its transport operations (including connection setup, requesting the web page, and data transmission delays). Figure 7.2 shows transport latency as compared to TCP in the kernel.

Here, we use two virtual machines on one physical machine acting as the client and the server respectively. We use [26] to add latency between the two virtual machines, making the RTT between them roughly 100ms. The client downloads 11.18KB data, as in the previous evaluation. Figure 7.2a shows the CDFs of ASAP's and TCP's transmission time over 1000 trials. The results confirm a significant reduction in latency. ASAP nearly always achieves one RTT reduction in latency (100ms) as compared to standard TCP. We then vary the file size from 1KB to 500KB, and run each case over 200 trials. Figure 7.2b shows the median values of total delay. ASAP achieves lower latency in all cases.

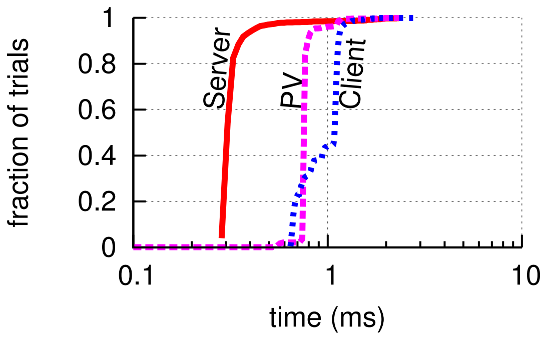


Figure 7.3: *Cryptographic overhead.*

7.3 Computational overhead

ASAP adds some additional computational overhead to several parts of today's Internet, including clients, web servers, DNS servers; and the new infrastructure we deploy, the Provenance Verifier. To characterize overhead of each of these components, we instrumented our implementation with code to measure the (a) pass-through time, i.e., the time from when a packet was received to when it was processed and forwarded, and (b) microbenchmarks, to characterize what fraction of overheads were due to cryptographic processing. We collected our experiments on a single core of a 2.83GHz Intel Core 2 Quad Q9550 processor with 4GB RAM.

Provenance Verifier: Each request to the PV requires a private encryption to generate a PC. We would like the overhead at the PV to be low, to reduce the number of PVs, by enabling each PV to service a larger number of clients. Figure 7.3 shows a CDF of the PV's per-request processing time with a 1024-bit RSA key, over 1000 trials. We find that more than 95% of certification requests can be processed in less than 0.8ms. Assuming each client needs to renew a PV once per day, a single PV server with a single core could handle several tens of millions of clients.

Client: There are two key sources of overhead at clients: the time spent communicating with the PV, and the time spent communicating with the server. The former happens rarely — it is done when the client obtains a new IP address (e.g. via DHCP), and therefore does not affect the latency associated with connection setup (except for connections initiated immediately after obtaining a new address). To evaluate the amount of overhead ASAP introduces when communicating with the server, we perform microbenchmarks (Figure 7.3). ASAP's overhead at the client was dominated by cryptographic operations, namely, the encryption operations involved in constructing the request certificate (RC). However, this overhead was typically on the order of 1-2ms, substantially less than typical round-trip times.

DNS servers: ASAP increases DNS overhead in two ways. First, the client sends two queries to DNS, which could double its workload in the worst case (if entries are cached, this overhead could be substantially reduced). Second, the ADNS does additional processing on packet contents: it removes the ASAP component of the request before processing and caching the appropriate information, then replaces the ASAP component before forwarding the response back to the LDNS. To evaluate this overhead, we performed an experiment where the client sends 1000 requests to the server. We found that overhead increased by on the order of 100 microseconds at the LDNS and the ADNS. However, this overhead is small compared to the total packet processing delay in Unbound (0.7 ms on average).

Web server: An ASAP-enabled web server performs an RSA verification of the message before processing it. We measure this overhead in Figure 7.3. Here, we made a client running in the UCLA PlanetLab site send 1000 requests to the ASAP-enabled web server. The median time required to perform the verification operations on a single request is 0.326ms, the mean is 0.359ms, and the 95th percentile is 0.462ms. This is significantly less than typical RTTs, so although it does add some computational overhead, ASAP would provide a large benefit in end-to-end delay.

Chapter 8

Deployment

Like most new transport and naming protocols, our design requires changes to certain clients and servers. However, our work differs from "clean slate" approaches which require large-scale changes to the Internet infrastructure to achieve their benefits. Instead, ASAP can be deployed end-to-end, only requiring cooperation of the end host client and server. In particular, ASAP can be deployed in the wide area through the following modifications to existing systems:

ADNS: Since the ADNS is typically owned and operated by the service provider, many deployed systems (e.g., Akamai) leverage the ADNS as an easy-to-modify location to place new functionality. Our implementation of ASAP consists of some simple extensions to software running at the ADNS. Moreover, if desired, ASAP's ADNS functionality could alternatively run directly at the ASAP-enabled web server, by running a DNS server at it, and registering that server with public DNS system.

Clients: ASAP requires modifications to the client's TCP implementation. A key question is whether this is deployable in a backwards-compatible manner, so clients interacting with legacy servers will simply fall back to TCP. One option is the strategy of our implementation: include the certificates in the data portion of the SYN packet. This is likely to work well in practice since most current TCP implementations discard this data; but technically it could be unsafe because legacy servers that *do* use SYN data would misinterpret our certificates as application data. Another option is to put the certificates in a TCP option, as in TCP Fast Open [27]. However, for this we would need more space for options, as proposed in [28].

To avoid changing end host network stacks, applications could use ASAP on top of UDP, as in our modification of UDT [22]. To avoid modifying end hosts entirely, ASAP could be deployed at web proxies and caches.

Server: Like the client host, the server should be modified with extensions to support the ASAP protocol. If desired, ASAP could also be deployed as a reverse proxy, to avoid the need to modify servers; however, we note that service providers already commonly customize their operating system and web server implementations.

PVs: Note that ASAP does *not* require deployment of a shared PV infrastructure. Such a deployment scenario would be useful and could arise, but in an initial deployment phase each organization, such as an ISP or an content distribution network provider, could host its own PVs, thus requiring little coordination. This deployment is likely to bring large benefits for content distribution networks and other large content providers.

DNSSEC instead of PVs: An alternative to verifying clients' source addresses with PVs is to use DNSSEC's [29–31] designated signer (DS), public key (DNSKEY), and signature (RRSIG) records to certify ownership of IP addresses via reverse DNS lookup, as in [32]. The server would verify the chain of certificates from the root of the DNS hierarchy to the client. Servers could cache some of these records near the top of the hierarchy, while the client would provide others in its request. This leads to a tradeoff: placing more records in the query increases its size, which is already constrained (particularly when piggybacked within DNS queries); but if we require the server to cache more levels of the hierarchy of certificates, it will lead to cache misses and the server will need to query the DNSKEY records from corresponding DNS servers, increasing delay. In addition, compared with our PV design, using DNSSEC may be more difficult to deploy, as it requires each client's local ISP to issue certificates of IP ownership to the client.

Leveraging accountable Internet architectures: Although ASAP does not require extensive changes to the Internet, it can benefit from deployment of previously proposed clean-slate designs. For example, systems that cryptographically certify location or ownership of IP addresses [32, 33] may obviate the need to run PVs.

Chapter 9

Related Work

There has been much work on *network-layer* changes to reduce delays. Most closely related and concurrently with our work, in TCP Fast Open (TFO) [27] a server gives a client a cookie, which it can use to skip the 3WH on future connections with the server (or others that have a shared secret key distribution arrangement with the server). TFO is similar to a special case of ASAP where the PV is the server. TFO is likely easier to deploy because it does not require public-key cryptography in the transport layer stack; servers can validate the cookie more quickly than ASAP's certificates and it can fit in a single standard-sized TCP option field. In ASAP, it would be easier for a client to use a single provenance certificate across multiple servers that may not trust each other. In addition, our work studied how to defend against PV eavesdropping which would be applicable to TFO as well if its cookies were used across distributed servers; and our DNS improvements are complementary to TCP Fast Open.

T/TCP [12, 13] bypasses the 3-way handshake and truncates TIME-WAIT State. However, T/TCP is quite vulnerable to attacks [14]. TCP Fast Start [34] enables clients to cache network parameters to speed up web transfers. Some protocols such as the Stream Control Transmission Protocol (SCTP) and the Host Identity Protocol use four-way handshakes (at the cost of additional delay) to gain better evidence that the initiator really exists at the given address before allocating state [35–37]; techniques we study in this paper may provide additional gains for such protocols. Dukkipati et al. [38] argue for increasing TCP's initial congestion window. To alleviate DNS resolution delays, DNS records may be proactively cached [39]. Proposals on HTTP aiming at reduce web latency include modifications to HTTP itself in the application layer [40], and the transport protocol HTTP carries traffic over, such as DHTTP [41]. There has been work on speeding up TCP for long-lived flows over high-bandwidth networks [42, 43]. Other works have observed that the congestion control algorithm of TCP makes flows last much longer than necessary, especially for short-lived flows like web traffic; to address this, they propose new congestion control and scheduling schemes [44–47]. To reduce latency resulting from DNS queries, TCP connection set up and HTTP session initiation, Cohen et al. propose a design based on pre-fetching, including pre-resolving domain names, pre-connecting, and pre-warming the connection [48, 49]. DEW [9] explores ways by which a variety of Web requests and responses could be piggybacked on DNS messages, but requires modifications on both LDNSs and ADNSs. Most of these works focus on individual protocols. Cooperative caching and lookup can speed up DNS in isolation [50, 51]. By merging functions of DNS and TCP, our design can realize additional gains. In addition, our design aims to speed connection setup while retaining the security and idempotency properties of existing TCP and DNS infrastructures. At the same time, our design is orthogonal to a number of these works, and can be used in conjunction with them to gain additional latency reductions.

In addition, there has been much work on making *application-layer* changes to web infrastructures to reduce delay. While this space is too broad to summarize here and is largely orthogonal to our work, we note that several techniques can reduce the effect of transport layer delays. Content distribution facilities enable migration and replication of services closer to end users. Application-layer protocol optimizations have been proposed and integrated into the HTTP specification such as persistent connections, and more recently Google's SPDY proposal [52]. Unfortunately, application-layer techniques cannot eliminate the additional delays that arise from transport and lookup operations. While these techniques have become widely deployed, interactivity of web requests remains a key concern that significantly affects usability of many Internet services [1, 3, 4].

Chapter 10

Conclusions

In this project, we proposed ASAP, a new low-latency transport protocol for interactive applications on the Internet, such as web browsing. ASAP revisits classic Internet design decisions by modifying and merging functionality of DNS and TCP to substantially reduce connection establishment delay, which is the dominant part of short request-oriented communication latency. ASAP attains its benefits while retaining desirable properties of existing TCP and DNS deployments.

We implemented ASAP both (a) in user-space application layer, as a modification to UDT library, and deployed a prototype on Planetlab, and (b) as a Linux kernel module. We evaluated transport-layer performance of ASAP, i.e., downloads of individual files, in terms of both downloading time and overhead at each component. However, downloading a multi-object web page may compound ASAP's benefit (because the browser often needs to open TCP connections to multiple servers in serial) and also may reduce ASAP's relative benefit (when connection establishment latency is dwarfed by other delays). We consider evaluating such application-level performance as an important direction of future work. From evaluation, we found ASAP able to reduce one to two RTTs per connection, cutting up to two-thirds latency of short web requests.

Appendix A

Bounding the number of PVs

We model a network as an arbitrary graph G with n vertices V, and assume that it employs fixed but arbitrary single-path routing. That is, when s sends a message to d, it follows a specific arbitrary path $P(s, d)$; these paths may or may not be related to shortest paths in the network. The attacker can eavesdrop on some set of locations $A \subseteq V$, and therefore can eavesdrop on $s \rightsquigarrow d$ traffic when $P(s, d) \cap A \neq \emptyset$.

Definition 1 *A source-destination pair (s, d) is* **attackable** *in a protocol (TCP or ASAP) for a given set E if the attacker can cause s to send a flow of data to d. A destination d is* **attackable** *if there exists a source s for which (s, d) is attackable. If there are $\geq n/2$ such sources, then d is* **highly attackable**.

We assume that ASAP uses a set P of PVs which are trusted by all servers. Therefore, if any d is attackable in ASAP, then (s, d) is attackable for all sources s.

Theorem 1 *Suppose $(k + 2)\log_2 n$ PVs are placed in uniform-random locations, and the attacker eavesdrops on an arbitrary set of k locations after knowing where the PVs are placed. With probability $\geq 1 - \frac{1}{n}$ (over the choice of PV locations), any destination that is attackable in ASAP is highly attackable in TCP.*

Proof:

Fix any destination d and attacker locations A. Let S_A be the set of sources s for which the attacker can eavesdrop on the path $s \rightsquigarrow d$, and let $f = \mathsf{S\_A}/n$. If $f \geq \frac{1}{2}$, then d is highly attackable in TCP and the theorem holds for this d.

Otherwise, if $f < \frac{1}{2}$, for ASAP, $\Pr[d \text{ is attackable}] = \Pr[P \subseteq S_A] = f^{\mathsf{P}} < 2^{-\mathsf{P}}$. Now, we want to bound the probability that any of the n possible destinations is attackable for any of the $\binom{n}{k}$ possible sets A. By a union bound over these $n\binom{n}{k}$ events, the probability that any bad event happens is $< n\binom{n}{k}2^{-\mathsf{P}} \leq n^{k+1}2^{-\mathsf{P}} \leq \frac{1}{n}$ since $\mathsf{P} \geq (k+2)\log_2 n$.

References

[1] J. Dabrowski and E. Munson, "Is 100 milliseconds too fast?" *ACM Conference on Human Factors in Computing Systems (CHI)*, April 2001.

[2] S. Okamoto, M. Konyo, S. Saga, and S. Tadokoro, "Detectability and perceptual consequences of delayed feedback in a vibrotactile texture display," *IEEE Transactions on Haptics*, April 2009.

[3] S. Savage, N. Cardwell, G. Voelker, A. Wolman, T. Anderson, and H. Levy, "Examining web latency: Performance analysis of a wide-area distributed system," *Technical Report, Department of Computer Science and Engineering, University of Washington, Seattle*, 1999.

[4] C. Metz, "Google seeks interwebs speed boost with TCP tweak," *The Register*, June 2010, http://www.theregister.co.uk/2010/06/23/google_engineering_vp_on_speed/.

[5] J. Brutlag, "Speed matters for Google web search," June 2009, http://code.google.com/speed/files/delayexp.pdf.

[6] TelecomPaper, "Google search advertising revenue grows 20.2 percent in 2010," January 2011, http://www.telecompaper.com/news/google-search-advertising-revenue-grows-202-in-2010.

[7] "Round-trip time internet measurements from caida's macroscopic internet topology monitor," http://www.caida.org/research/performance/rtt/walrus0202/.

[8] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *IEEE/ACM Trans. Networking*, 2002.

[9] B. Krishnamurthy, R. Liston, and M. Rabinovich, "Dew: Dns-enhanced web for faster content delivery," *WWW '03 Proceedings of the 12th international conference on World Wide Web*, 2003.

[10] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *ACM SIGCOMM Computer Communications Review*, July 2001.

[11] R. Braden, "Requirements for internet hosts – communication layers," RFC 1122, October 1989.

[12] ——, "T/TCP - TCP extensions for transactions, functional specification," 1994.

[13] B. Krishnamurthy and J. Rexford, *Web protocols and practice.* Addison-Wesley, 2001.

[14] C. Hannum, "Security problems associated with T/TCP," Unpublished work in progress (IETF Informational), September 1996.

[15] C. Sunshine and Y. Datal, "Connection management in transport protocols," *Computer Networks*, December 1978.

[16] Arbor Networks, "Worldwide Infrastructure Security Report," vol. V, February 2010, http://www.arbornetworks.com/report.

[17] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and rsa on 8-bit cpus," *Cryptographic Hardware and Embedded Systems - CHES 2004*, August 2004.

[18] E. Rescorla, *SSL and TLS: Designing and building secure systems.* Addison-Wesley, 2000.

[19] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The end-to-end effects of Internet path selection," *ACM SIGCOMM*, vol. 29, no. 4, pp. 289–299, 1999.

[20] P. Vixie, "Extension mechanisms for DNS (EDNS0)," RFC 2671, August 1999.

[21] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," June 1999.

[22] "UDT: UDP-based data transfer," http://udt.sourceforge.net/.

[23] "OpenSSL," http://www.openssl.org.

[24] "RSA algorithm patch (kernel version 2.6.21-rc5-git6)," http://lwn.net/Articles/228892/.

[25] "Let's make the web faster," http://code.google.com/speed/articles/web-metrics.html.

[26] "netem," http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[27] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP fast open," *ACM CoNEXT*, December 2011.

[28] W. Eddy, "Extending the space available for tcp options," http://tools.ietf.org/html/draft-eddy-tcp-loo-04.

[29] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033, 2005.

[30] ——, "Resource Records for the DNS Security Extensions," RFC 4034, 2005.

[31] ——, "Protocol Modifications for the DNS Security Extensions," RFC 4035, 2005.

[32] A. Li, X. Liu, and X. Yang, "Bootstrapping accountability in the internet we have," *Proc. NSDI*, 2011.

[33] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable Internet Protocol (AIP)," *Proc. ACM SIGCOMM*, August 2008.

[34] V. Padmanabhan and R. Katz, "TCP Fast Start: A technique for speeding up web transfers," *Proc. IEEE GLOBECOM*, 1998.

[35] W. Eddy, "TCP SYN flooding attacks and common mitigations," RFC 4987, August 2007.

[36] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream control transmission protocol," RFC 2960, October 2000.

[37] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "Host identity protocol," RFC 5201, April 2008.

[38] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin, "An argument for increasing TCP's initial congestion window," *ACM Computer Communication Review*, 2010.

[39] E. Cohen and H. Kaplan, "Proactive caching of DNS records: Addressing a performance bottleneck," *Computer Networks*, April 2003.

[40] E. Cohen, H. Kaplan, and J. Oldham, "Managing TCP connections under persistent HTTP," *WWW*, 1999.

[41] M. Rabinovich and H. Wang, "DHTTP: An efficient and cache-friendly transfer protocol for the web," *Proc. IEEE INFOCOM*, 2001.

[42] S. Floyd, M. Allman, A. Jain, and P. Sarolahti, "Quick-Start for TCP and IP," RFC 4782, January 2007.

[43] D. Wei, C. Jin, S. Low, and S. Hegde, "FAST TCP," *IEEE/ACM Trans. Networking*, December 2006.

[44] X. Chen and J. Heidemann, "Preferential treatment for short flows to reduce web latency," *Computer Networks*, April 2003.

[45] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control and why this means we need new algorithms," *ACM Computer Communication Review*, 2006.

[46] M. Mellia, M. Meo, and C. Casetti, "TCP smart framing: A segmentation algorithm to reduce TCP latency," *IEEE/ACM Trans. Networking*, April 2005.

[47] L. Guo and I. Matta, "The war between mice and elephants," *Proc. International Conference on Network Protocols (ICNP)*, November 2001.

[48] E. Cohen and H. Kaplan, "Prefetching the means for document transfer: A new approach for reducing web latency," *Computer Networks*, July 2002.

[49] ——, "Proactive caching of DNS records: Addressing a performance bottleneck," *IEEE Symposium on Applications and the Internet (SAINT)*, 2001.

[50] V. Ramasubramanian and E. Sirer, "The design and implementation of a next generation name service for the internet," *Proc. ACM SIGCOMM*, August 2004.

[51] K. Park, V. Pai, L. Peterson, and Z. Wang, "CoDNS: Improving DNS performance and reliability via cooperative lookups," *Proc. OSDI*, December 2004.

[52] "SPDY: An experimental protocol for a faster web," https://sites.google.com/a/chromium.org/dev/spdy/spdy-whitepaper.