Push vs. Pull: Implications of Protocol Design on Controlling Unwanted Traffic

Zhenhai Duan
Florida State University

Kartik Gopalan
Florida State University

Yingfei Dong
University of Hawaii

Abstract

In this paper we argue that the difficulties in controlling unwanted Internet traffic, such as email SPAM, stem from the fact that many Internet applications are fundamentally *sender-driven* and distinctly lack *receiver control* over traffic delivery. However, since only receivers know what they want to receive, receiver-driven approaches may often have clear advantages in restraining unwanted traffic. In this paper, we re-examine the implications of the two common traffic delivery models: *sender-push* and *receiver-pull*. In the sender-push model, a sender can deliver traffic at will to a receiver, who can only passively accept the traffic, such as in the SMTP-based email delivery system. In contrast, in the receiver-pull model, receivers can regulate *if and when* they wish to retrieve data, such as the HTTP-based web access system. We argue that the problem of unwanted Internet traffic can be mitigated to a great extent if the receiver-pull model is employed by Internet applications, whenever appropriate. Using three popular applications – email, mobile text messages, and asynchronous voice messages – as examples, we demonstrate that asynchronous communication protocols can be easily designed using the receiver-pull communication model to suppress unwanted Internet traffic.

1 Introduction

In recent years the Internet has been increasingly plagued by the seemingly-never-ending unwanted traffic, manifesting itself in large volumes of unsolicited bulk emails (spam), frequent outbreaks of virus/worm attacks, and large scale Distributed Denial of Services (DDoS) attacks. For example, it was estimated that 32 billion spam messages were sent daily on the Internet as of November 2003 [11]. Worse, spammers and virus/worm attackers are increasingly joining force to automate spamming by hijacking (home) user machines through virus/worm attacks. A recent study reported that as high as 80% of spam messages were sent from compromised user machines (zombies) [15]. In this paper, we focus our attention on *spam-like* unwanted Internet traffic, which plagues critical Internet applications and services such as emails, mobile text messages, and asynchronous voice messages (where a recorded voice message is sent to a list of receivers). We refer to such applications collectively as *message services*. In this paper, we are especially interested in the implications of the protocol design on controlling unwanted traffic on the Internet.

Given the importance of controlling spam for preserving the value of the messaging systems, this issue has attracted a great amount of attention in both networking research and industrial communities. Many different spam control schemes (in the context of Internet emails) have been proposed, and some of them have been deployed on the Internet [3, 8, 9, 12, 13, 14]. On the other hand, despite these anti-spam research and development efforts, the proportion of spam seen on the Internet has been continuously on the rise. It is estimated that nowadays spam messages constitute 79% of all business emails, up from 68% since the US federal Can-Spam Act of 2003 took effect in January 2004 [2]. It was also reported that 80% of mobile phone text messages were unsolicited in Japan [16], where SMS (Short Message Services) is popular, and is therefore attractive to spammers.

In this paper we argue that the difficulties in restraining spam can be attributed to the lack of *receiver control* over how messages should be delivered on the Internet. For example, in the current SMTP-based email delivery architecture [10], any user can send an email to another at will, regardless of whether or not the receiver is willing to accept the message. In the early days of the Internet development, this was not a big problem as people on the network largely trusted each other. However, since the commercialization of the Internet in mid-1990, the nature of the Internet community has changed. It has become less trustworthy, and email spam is possibly one of the most notable examples of the untrustworthy nature of the Internet.

In order to effectively address the issue of spam in the untrustworthy Internet, we argue that *receivers must gain greater control over if and when a message should be delivered to them*. Asynchronous messages on the Internet are delivered primarily using two different models: sender-push and receiver-pull (or a combination of the two). They differ in who initiates the message delivery process. In the sender-push model, senders control the delivery of traffic, and receivers passively accept whatever the senders push to them. The current SMTP-based email delivery system is a typical example of this model. In contrast, the receiver-pull model grants receivers the control over if and when they want to retrieve data from the senders. In this model, senders can only prepare the data but they cannot push the data to receivers. Examples of the receiver-pull model include the HTTP-based web access services and the FTP-based file transfers.

As we will discuss in the next section, the receiver-pull model comes with several appealing advantages because it grants receivers greater control over the message delivery mechanism. It takes advantage of the fact that receivers have more reliable knowledge of what traffic they want to receive. Moreover, the receiver-pull model may also simplify the challenging issues related to the resource usage accountability and sender authentication. For example, because spammers need to store and manage email messages on their own mail servers (waiting for receivers to pull), it becomes relatively easier to hold spammers responsible for the resources they consume. As a proof of concept, in this paper we present examples of three asynchronous messaging applications – emails, mobile text messages, and asynchronous voice messages.

The objective of the paper is two-fold. First, through the example designs of the message applications, we would like to demonstrate the feasibility and advantages of using receiver-pull model to design protocols for asynchronous messaging applications. Second, and more importantly, we want to raise the explicit awareness of the difference between the sender-push and receiver-pull models, and argue that, the receiver-pull model should be the strongly favored design choice, whenever appropriate.

The rest of the paper is structured as follows. In Section 2 we elaborate on the two different traffic models on the Internet. We outline the example design to support emails, mobile text messages, and asynchronous voice messages using the receiver-pull model in Section 3. We summarize the paper in Section 4.

2 Push vs. Pull: Implications of Protocol Design Choice

The choices made during protocol design phase have fundamental implications on security, usability, and robustness of any distributed message delivery system. One such important design decision is whether to adopt a sender-push or a receiver-pull model or a combination of the two models (see Figure 1). In this section we discuss the implication of these design choices and make the case that the receiver-pull model can prove to be highly effective in discouraging unwanted traffic.

2.1 The Sender-Push Model

In the sender-push model, the sender knows the identity of a receiver in advance and pushes the message in an asynchronous manner to the receiver. The receiver accepts the entire message, may choose to optionally examine the message, and then accept or discard it. An important aspect of sender-push model is that the entire message is received before any receiver-side processing is performed. A number of communication services in the Internet rely on the sender-push model. A prime example is email in which the sender relies on the Simple Mail Transfer Protocol (SMTP) to push an entire email message to a passive receiver. Asynchronous voice messages over the telephone network (both traditional and IP based) represent another important application of the sender-push model.

A common variant of the sender-push concept is the *receiver-intent-based sender-push* (RISP) model. The most common examples of the RISP model are the subscription-based services such as mailing lists, where user subscribes to a service which subsequently pushes the data to the receiver. Other popular subscription-based applications of the RISP model include stock and news ticker applications and automatic software updates. Similarly, Instant Messaging is another application where the message itself is pushed by the sender, but the receiver can allow or disallow messages from specific users.

A common feature among all the above examples is that the content itself is pushed to the receiver, whereas the receiver may optionally provide minimal control feedback to the sender. The primary advantage of the sender-push model is that its asynchronous message delivery framework is conceptually simple and fits naturally for many useful applications such as email, text, and voice messaging. Sender initiates message transfer when the message is ready, the receiver simply waits passively for any message to arrive and accepts one when it does arrive. Furthermore, there is no significant storage requirement on the sender side.

The biggest disadvantage of the sender-push model is

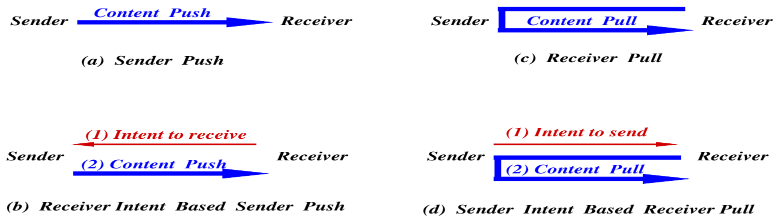

Figure 1: Common message delivery models.

that it is the sender who completely controls *what* message is delivered and *when* it is delivered. The receiver has neither the knowledge of what message he/she will receive, nor when the message will be received. The receiver is ideally expected to receive the entire message before processing or discarding it. Apart from generating and transmitting the message, the sender does not commit any resources for the transmitted message. On the other hand, the receiver has to wait, receive, process and store (or discard) the message even if the message is not of interest to the receiver.

The RISP model alleviates this concern to some extent by allowing receivers to provide control feedback. However it is not easy to implement in many popular applications. For example, adopting the RISP model for email, mobile text and voice messages requires the receiver to maintain an exhaustive white-list or black-list of email addresses and phone numbers of potential senders. Indeed, approaches such as Reverse Black Lists (RBL) [13] adopt this philosophy in trying to blacklist email spammers. However most potential correspondents, such as first time senders, fall in neither of the two categories. To handle such unclassified cases, receivers end up relying on content-based-filters, i.e. they receive the entire message, scan it to determine if it is wanted and then either accept or discard it. *The fundamental problem here lies in having to accept and examine the entire message before culling it.*

An additional disadvantage of the sender-push model is that the sender can vanish (go offline) immediately after pushing unwanted content to the receiver. This makes it quick and easy for a malicious sender to hide its identity. Once the receiver accepts the content, it is difficult at best to trace back a malicious sender.

In summary, while the sender-push model is both simple and convenient, it comes with a serious baggage, namely, that senders control what to send and when to send, and cannot be easily held accountable for sending unwanted content to receivers.

2.2 The Receiver-Pull Model

In the receiver-pull model, it is the receiver who initiates the message transfer by explicitly contacting the sender. The sender passively waits for the receiver and delivers the entire content upon receiving a request. Since it is the receiver who initiates the message transfer, the receiver would have explicit greater control over the message transfer and implicit greater trust in the received content, than in the sender-push model.

A number of successful communication services rely on the receiver-pull model. The most important examples using the receiver-pull model are the FTP and HTTP protocols. In both cases, the receiver initiates the data transfer by opening an FTP connection or by typing/clicking on a URL, respectively. (Interestingly, HTTP supports both receiver-pull and as well as RISP variant of sender-push, though the former is more commonly used. Examples of RISP model techniques in HTTP include automatic page refreshes and the hugely unpopular popup windows).

An interesting and useful variation of receiver-pull model, which is of special interest to us, is the *sender-intent-based-receiver-pull* (SIRP). In this model, the sender first expresses an intent to send content to the receiver via a small intention message. If the receiver happens to be interested, it contacts the sender and retrieves the content. A common example of the SIRP model is the *pager* service. Here the caller expresses an intent to talk to a callee by paging the latter and leaving a callback number. If the callee is interested, he/she contacts the caller back on the callback number. The main feature of the SIRP model is that the content itself is pulled by the receiver whereas only a short intent is pushed by the sender.

The primary advantage of the receiver-pull model is that a receiver exercises control over when and what it receives. The receiver has the freedom to first determine its own level of interest in the content (as well as the reputation of the sender) *before* it actually requests the content. Secondly, it becomes the responsibility of the sender to store and manage the content till the receiver is ready to retrieve it. For instance, an FTP or web server needs to store and manage its own files whereas receivers access it only when they are interested. Thirdly, there is a large window of time over which a malicious sender is forced to stay online and reveal its identity. For the pure receiver-pull model, this window is from the moment content is generated and named till the content is retrieved by the receiver. For the SIRP model, this window is from the moment sender expresses its intent to send till the time receiver retrieves the content. Thus, unlike the sender push model, there is a large window of time in which the receiver is free to verify a sender's identity.

One obvious disadvantage of receiver-pull model is that the sender is burdened with greater content management complexity. The sender needs to store outgoing messages and keep them available at least till the intended receivers are willing to retrieve them, and needs to have a deletion policy if a message is never retrieved by the receiver. Another issue that the sender needs to grapple with is to ensure that the party retrieving a message is indeed the originally intended receiver. However, another angle to look at these disadvantages is that, in the sender-push model, it is the receiver who needs to deal with the very same issues.

2.3 Implications on Unwanted Traffic

Given that the receiver-pull model grants more control to receivers in terms of traffic delivery, and only receivers know what they want to receive, the receiver-pull model has clear advantages in restraining unwanted traffic compared to the sender-push model. Moreover, the above discussion also makes it clear that the sender is accountable to a greater degree in the receiver-pull model than in the sender-push model. This brings us to the following key idea which underlies the theme of this paper: *When designing any communication protocol, it is advantageous to first consider using a receiver-pull model which inherently provides greater protection against unwanted traffic.*

The receiver-pull based model is a relatively low-cost design choice that can be considered early during any communication system design. Even if the receiver-pull model results in slightly greater protocol complexity, it can greatly help to simplify accountability and authentication issues by placing the overheads where they truly belong – at the sender of the unwanted traffic.

A legitimate concern with a receiver-pull model is that it may end up increasing the cost of sending messages for malicious as well as legitimate senders. We will show in the next section through an example of a receiver-pull based email architecture that, using simple design optimizations, one can easily lower the sending cost for legitimate senders while still holding senders of unwanted content accountable.

We do not claim that a receiver-pull based model may be universally suitable for all forms of communications. For example, soldiers in the middle of a desert war may not want to rely on remote senders being reachable when trying to retrieve their messages. However, in many important applications, such as civilian use of email, mobile text messages, and asynchronous voice messages, the receiver-pull architecture appears to offer strong advantages in fight against unwanted traffic.

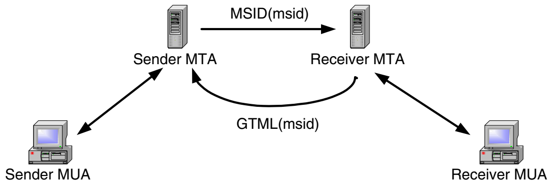

Figure 2: An email delivery architecture with receiver-pull model.

3 Applications of the Receiver-Pull Model

In order to illustrate the feasibility and advantages of the sender-intent-based-receiver-pull (SIRP) model in supporting asynchronous applications, in this section we outline the design of three important applications using the model: emails, mobile text messages, and asynchronous voice messages. We present the design of the SIRP based email system in greater detail and briefly sketch the design for the other two applications using a framework similar to the email design. (IM2000 [1] is another email architecture using the receiver-pull model, however it is not backward compatible.) We emphasize that these designs only illustrate the feasibility and effectiveness of supporting message services using the SIRP model, in reducing unwanted traffic. Many design details are omitted (see [4, 5] for supporting the Internet email application using the SIRP model).

3.1 SIRP based email System

In the SIRP based email delivery system, senders cannot directly push messages to arbitrary receivers. Instead, receivers decide if and when they want to retrieve (or pull) messages from senders. Figure 2 illustrates the basic architecture of the new email delivery system. In the following we will present the new system from both the senders' and receivers' perspectives. Before we delve into details, it is worth noting that the new system extends the current Simple Mail Transfer Protocol (SMTP) [10] by adding two new commands: MSID and GTML. In other words, all the commands and reply codes in SMTP are also supported in the new system. We will explain the two new commands when we use them.

3.1.1 Sender: Message Composition and Receiver Notification

Like in the current email architecture, a sender uses a Mail User Agent (MUA) to compose outgoing messages [10]. After a message is composed by the sender, the sender delivers the message to the sender Mail Transfer Agent (MTA). For simplicity, we refer to a sender MTA server as an SMTA, and a receiver MTA server as an RMTA.

All the outgoing messages are stored at the SMTA. For this purpose, the SMTA maintains an outgoing message folder for each *sender*. Instead of the complete message being directly pushed from the SMTA to the RMTA, only the envelopes (headers) of the messages are delivered. In particular, the SMTA notifies the RMTA about a new message by the new *message identifier* command MSID, which contains the unique identifier *msid* of the message. The identifier of a message is generated based on the sender, the message, the receiver, and a secret key of the sender.

We note that there is a fundamental difference between message pull in the new email delivery system and URL embedded in many current spam messages. The address in the URL is normally not related to the sending machine of the message, which makes it hard to identify the actual sender who is responsible for the spam message. On the other hand, outgoing messages in the new email system have to be stored on the sender mail servers instead of third-party machines before they are retrieved. In this way, we obtain several advantages in restricting spam. For example, senders need to keep their mail servers up until the messages are retrieved by receivers. This presents less flexibility for senders to move around by frequently changing their IP addresses and/or domains. In contrast, in the current (sender-push) SMTP-based architecture, spammers can send a large number of spam messages and shut down their mail servers, which makes it hard to hold spammers responsible for spamming. Moreover, in the new system, senders have greater responsibility to store and manage their outgoing email messages in comparison to the current email architecture, which imposes negligible responsibility on the senders. In summary, while the current SMTP-based email delivery architecture provides a *call-by-copy* interface to senders, the new system provides a *call-by-reference* interface to senders [6].

3.1.2 Receiver: Pulling Messages from Senders

The new email delivery system grants more control to receivers regarding if and when receivers want to read a message, senders cannot arbitrarily push a message to them. Receivers can be discriminate about which messages need to be retrieved, and which ones need not. If the receiver indeed wants to read a message, he will inform his own RMTA, and the RMTA will retrieve the message from the SMTA on behalf of the receiver. An RMTA retrieves an email message using a *get mail* command **GTML**, which includes the identifier *msid* of the message to be retrieved. After the message has been pulled to the RMTA, conventional virus/worm scanning tools and content-based spam filters can be applied to further alert the receiver about potential virus or spam. Therefore, *the new email delivery system does not exclude the use of existing email protection schemes*. For security reasons, when an SMTA receives the **GTML** command, it needs to verify that the corresponding message is for the intended receiver, and more importantly, the requesting MTA is the mail server responsible for the receiver (i.e. the one which was originally contacted for message delivery).

By only delivering the envelope (including *msid*) of a message from a sender to the receiver, less bandwidth, storage, and processing time is used at the receiver side, which is especially important for resource constrained users, e.g., wireless, PDA, or dial-up users. On the other hand, if the receiver indeed wants to read the message, negligible extra time and bandwidth is required. Since the receiver is less likely interested in messages from unknown sources, the majority of such messages will not be retrieved. As a result, considering the huge volume of spam on the Internet, much less bandwidth will be wasted by spam. For simple back of envelope calculation, assuming there are 30 billion spam messages sent daily on the Internet [11] and the average size of these messages is 5 KBytes [7]. We further assume the envelope of these messages occupies 1KBytes on average. Then it is easy to see that we will have daily 120 Tera Bytes worth of bandwidth saving on the Internet. Note that if content-based filter is used alone, these spam messages are still delivered on the Internet.

3.1.3 Differentiating Message Deliveries

The simple SIRP model not only puts more burden on spammers but also regular contacts of a receiver. To address this issue a hybrid email delivery system can be designed to support both the sender-push and receiver-pull models. In such a system, each receiver maintains a list of regular contacts, whose complete messages can be directly pushed from the senders to the receiver using the current SMTP protocol. In addition, a list of blacklisted contacts can be summarily declined. Messages from non-regular contacts should be stored and managed by the sender mail servers, and only the envelopes of such messages are directly delivered to the receiver to notify the pending messages.

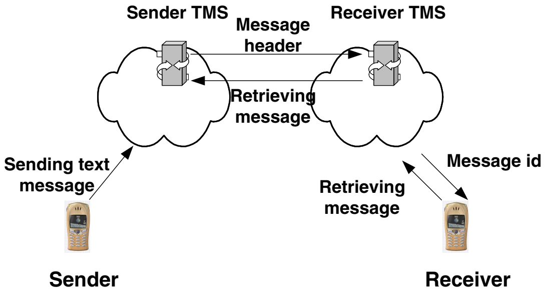

Figure 3: Supporting mobile text messages with SIRP model.

3.1.4 Practical Deployment Considerations

It can be shown that the new email delivery system can be deployed incrementally, and popular message applications such as mailing lists can also be supported [4, 5].

3.2 Mobile Text Messages and Asynchronous Voice Messages

Figure 3 illustrates the architecture in supporting mobile text messages using the SIRP model. Each mobile phone service provider will deploy one or multiple text message servers (TMS). When a user sends a text message to another user (who may be with another provider), the text message is stored in the sender provider's TMS, and only the message header (including the corresponding phone number and a message id) is sent to the receiver provider's TMS. The receiver provider's TMS will notify the receiver about the message header. If the receiver wants to read the message, the receiver provider's TMS will retrieve the message from the sender provider's TMS on behalf of the receiver.

Asynchronous voice messages are currently supported by cell phone service providers, where a recorded voice message is sent to a receiver, or a group of receivers. This service can be potentially exploited by spammers given its capability to send a voice message to a large number of receivers with relatively little effort. Moreover, as the service is being integrated into VoIP based applications, it becomes even more attractive to spammers. This service can be supported using the SIRP model instead of the sender-push model essentially in the same manner as mobile text messages. We skip the detailed discussion due to space considerations.

4 Summary

In this paper we examined the fundamental implications of the two different traffic delivery models, sender-push vs. receiver-pull, on controlling unwanted traffic on the Internet. Using examples of three popular applications – email, mobile text messaging, and asynchronous voice messaging – we illustrated that the receiver-pull model can be effectively used for asynchronous messaging in place of the current sender-push model to reduce unwanted Internet traffic. Another important contribution of this paper is that, by examining the implications of two traffic delivery models, we attempt to raise explicit awareness of the impact of the two models on unwanted Internet traffic, and argue that, a receiver-pull model should be strongly favored, whenever appropriate.

References

[1] BERNSTEIN, D. Internet Mail 2000 (IM2000). *http://cr.yp.to/im2000.html.*

[2] CLABURN, T. Big guns aim at spam. *Information Week* (Mar. 2004).

[3] DELANY, M. Domain-based email authentication using public-keys avertised in the DNS (domainkeys). Internet Draft (Aug. 2004). [draft–delany-domainkeys–base–01.txt].

[4] DUAN, Z., DONG, Y., AND GOPALAN, K. DiffMail: A differentiated message delivery architecture to control spam. Tech. Rep. TR-041025, Department of Computer Science, Florida State University (Oct. 2004).

[5] DUAN, Z., GOPALAN, K., AND DONG, Y. Receiver-driven extensions to SMTP. Internet Draft (May 2005). [draft–duan–smtp–receiver–driven–00.txt].

[6] FU, K. Personal communication. MIT, (Mar. 2005).

[7] GOMES, L., CAZITA, C., ALMEIDA, J., ALMEIDA, V., AND MEIRA, W. Charactering a spam traffic. In *Proceedings of IMC'04* (Oct. 2004).

[8] GRAHAM, P. A plan for spam
. *http://www.paulgraham.com/spam.html* (2003).

[9] JUELS, A., AND BRAINARD, J. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of NDSS-1999 (Networks and Distributed Security Systems)* (Feb. 1999).

[10] KLENSIN, J. Simple mail transfer protocol. RFC 2821 (Apr. 2001).

[11] LAURIE, B., AND CLAYTON, R. "Proof-of-Work" proves not to work. http://www.apache-ssl.org/proofwork.pdf (May 2004).

[12] LYON, J., AND WONG, M. Sender ID: Authenticating e-mail. Internet Draft (Aug. 2004). [draft–ietf–marid–core–03.txt].

[13] RBL. Real-time spam black lists (RBL). http://www.email-policy.com/Spam-black-lists.htm.

[14] RISHI, V. Free lunch ends: e-mail to go paid. *The Economic Times* (Feb. 2004).

[15] SANDVINE INCORPORATED. Trend analysis: Spam trojans and their impact on broadband service providers (June 2004).

[16] THE WASHINGTON POST. FCC sets sights on mobile phone spam (Mar. 2004).