Peer-to-Peer Netw. Appl.
DOI 10.1007/s12083-012-0128-8

iDispatcher: A unified platform for secure planet-scale information dissemination

Md Sazzadur Rahman · Guanhua Yan · Harsha V. Madhyastha · Michalis Faloutsos · Stephan Eidenbenz · Mike Fisk

Received: 24 July 2011 / Accepted: 1 March 2012

Abstract Traditional software and security patch update delivery mechanisms rely on a client/server approach where clients pull updates from servers regularly. This approach, however, suffers a high window of vulnerability (WOV) for clients and the risk of a single point of failure. Overlay-based information dissemination schemes overcome these problems, but often incur high infrastructure cost to set up and maintain individual information dissemination networks. Against this backdrop, we propose iDispatcher, a planet-scale, flexible and secure information dissemination platform. iDispatcher uses a hybrid approach with both push- and pull-based information dissemination to reduce the WOV period and achieve high distribution coverage. iDispatcher also uses a peer-to-peer based architecture to achieve higher scalability. We develop a self-contained key management mechanism for iDispatcher. Our prototype for iDispatcher is deployed on more than 500 PlanetLab nodes distributed around the world. Experimental results show that iDispatcher can have small dissemination latency for time-critical applications, is highly tunable to optimize the tradeoff between bandwidth and latency, and works resiliently against different attacks such as flooding attacks.

Keywords Information dissemination · Peer-to-peer · Software update

The work is partially supported by Los Alamos National Laboratory, NM. Los Alamos National Laboratory Publication No. LA-UR 11-02386

Md S. Rahman (✉) · H. V. Madhyastha · M. Faloutsos
Department of Computer Science, University of California, Riverside, CA, USA
e-mail: rahmanm@cs.ucr.edu

H. V. Madhyastha
e-mail: harsha@cs.ucr.edu

M. Faloutsos
e-mail: michalis@cs.ucr.edu

G. Yan · S. Eidenbenz
Los Alamos National Laboratory,
Los Alamos, NM 87545, USA

G. Yan
e-mail: ghyan@lanl.gov

S. Eidenbenz
e-mail: eidenben@lanl.gov

M. Fisk
Advanced Computing Solutions, Los Alamos National Laboratory, Los Alamos, NM 87545, USA
e-mail: mfisk@lanl.gov

1 Introduction

Information dissemination at a large scale has many applications, such as software update delivery [1], security patch delivery [2], fast code dissemination in a data center [3], and cyber C&C (Command and Control). Traditional information dissemination approaches mostly rely on client/server-based architectures, where clients pull information from the servers regularly. Due to the nature of such architectures, these information dissemination methods suffer from the risk of a single point of failure (e.g., under a Distributed Denial of Service (DDoS) attack) and poor scalability due to the performance bottleneck at the server side. For time-critical applications such as security patching, client/server architectures also expose clients to a large window of vulnerability (WOV) due to its pull-based mechanism for content delivery. A large WOV poses severe security risks to clients, as it allows attackers time to reverse engineer security updates and attack those clients who have not yet updated their systems. For instance, Gkantsidis et al. have shown that 20% out of 300 million Windows clients take more than 24 h to install a patch and thus the WOVs for these clients are at least as long as 24 h [1]. Such large WOVs may lead to epidemic malware spreading, as evidenced by the Code Red worm, which compromised 360 K machines within only 13 h and eventually caused $2.4 billion loss in 2001 [4].

To address the limitation of client/server-based architectures for information dissemination, push-based approaches relying on overlay networks have been investigated in the literature [2, 5, 6]. These methods commonly adopted peer-to-peer networks for information dissemination and were mostly focused on time-critical applications such as security update. Some other information dissemination schemes were designed, and therefore optimized, for a specific type of applications. For instance, Vigilante [7] uses overlay broadcast for disseminating security patches for stopping fast propagating worms that has a machine-verifiable proof of vulnerability. Twitter [3, 8] and Facebook [9] use Bit-Torrent based information dissemination systems to update their codebase in their data centers, and as these systems are used in a closed environment, security is of less concern than performance. Different system configuration tools such as CFengine [10] use mostly client-server based architecture for disseminating configurations to end hosts. iDispatcher can transperantly replace the dissemination of configuration mechanism of such tools to achieve scalable, fault tolerant and fast dissemination.

Compared with client/server-based architectures, these overlay-based information dissemination schemes enjoy higher level of scalability and resilience against failures. These systems, however, do not provide a unified platform that can be shared across different types of applications. Such a unified platform has the following advantages:

- Many applications have only a small user base, and developing a separate information dissemination platform for each application is costly. By contrast, resources within a generic information dissemination are shared by many applications and thus can be used in a more cost efficient manner.
- An important component in an information dissemination system is trust management. A computer receiving a security update has to verify its authenticity before installing it. A unified information dissemination system greatly simplifies the process of each computer managing its trusts on a number of information sources.
- A modern computer may need to interact with many information sources, and maintaining a separate information dissemination system for each of them complicates software and network management and may even introduce security loopholes.

Against this backdrop, in this work we aim to design a massively scalable and tunable security system for distributing information. It uses a distributed hash table (DHT) based architecture for achieving high scalability. iDispatcher supports any number of dissemination centers, which send information to different user groups on a shared platform. iDispatcher provides a novel self-contained PKI mechanism for nodes to verify the authenticity of information that flows in the network in a seamless manner. It can be configured to support not only time-critical applications, which demand real-time dissemination latencies and short WOV period, but also those that require high dissemination coverage. Obviously, such a flexible platform imposes high tunability of the system, which distinguishes iDispatcher from previous information dissemination systems. Table 1 shows how iDispatcher differs from other existing work for information dissemination.

Table 1 Comparison with different information dissemination approaches

	Push	Pull	Source verification	Scale to multiple groups
Microsoft Update [1]	✘	✔	✔	✘
Murder [3]	✔	✘	✘	✘
Bit-Torrent	✘	✔	✘	✔
Revere [2]	✔	✔	✔	✘
Bee [6]	✘	✔	✘	✔
Vigilante [7]	✔	✘	✔	✔
iDispatcher	✔	✔	✔	✔

In a nutshell, our contributions made in this work are summarized as follows:

- **Design of iDispatcher** iDispatcher is designed to be a tunable and secure planet-scale information dissemination framework. The system integrates different modes of communication such as multicast and broadcast. It supports both push- and pull-based information dissemination to strike a balance between timeliness and communication overhead. iDispatcher adopts a self-contained mechanism for distributing certificates in the entire network. iDispatcher is also robust against different attacks such as flooding and index poisoning attacks.
- **Implementation of iDispatcher** Its implementation is modular and tunable. Modularity ensures easy extensibility of the existing code when new features are added or some available functionalities need to be replaced. Tunability guarantees ability to get desired performance. It is written in C++ (11 K new line of code and 25 K line of code ported for DHT and multicast) and developed for the Linux environment as PlanetLab machines currently run on Linux OS. However, it is possible to compile the code for other platforms such as Windows or embedded OS. It is intended to be open source and therefore, is available for public download and use [11].
- **Experiments with iDispatcher** We deployed iDispatcher on hundreds of PlanetLab [12] nodes distributed around the world. Experimental results show that iDispatcher can be configured to produce small dissemination latency (20 s to disseminate information among 1,000 nodes). We performed intensive experiments to see how iDispatcher performs under different parameter settings. We also evaluated its robustness against different attacks.

The rest of the paper is organized as follows. We discuss related work in Section 2. Section 3 shows the system architecture of iDispatcher. We further discuss how to defend against different attacks in Section 4. We evaluate the performance of iDispatcher in Section 5 and draw concluding remarks in Section 7.

2 Related work

Although a number of approaches have been proposed in the literature for information dissemination to a large number of nodes, they were not developed with tunable parameters to meet the requirements of diverse applications. Moreover, unlike iDispatcher, these previous information dissemination systems do not provide a unified platform to support information dissemination from multiple sources in a seamless manner. In the following, we give a brief introduction to these systems.

Software update distribution The Windows Update system, which provides the largest software update service in the world, uses a client/server architecture for software update patch distribution. Apart from the Windows Update system, other products that use Automatic update include Red Hat Network [13] and Mac OS X software update [14]. Gkantsidis et al. [1], after analyzing data traces of Windows Update service collected over the period of more than a year, show that 20% of 300 million clients takes at least 24 h to receive software update, which thus leads to a long period of vulnerability for exploitation by attackers. To address this issue, they further explored P2P as an alternative update-delivery strategy. In their proposed system, a software patch is downloaded from any randomly chosen peer. Their results show that the P2P update delivery scheme leads to lower aggregate server load compared against the client/server architecture.

The Fedora project uses BitTorrent, a peer-to-peer file distribution technology for software binary distribution [15]. However, Serenyi et al. have identified several problems regarding BitTorrent as a patch distribution strategy, such as large overhead for small-sized patch files and thus large bandwidth consumption, poor quality of service due to the flash crowd, and BitTorrent's well know firewall traversal issues [16].

Alert dissemination against worm propagation P2P-based approaches have been studied for disseminating alerts to stop fast propagating worm in the Internet. Costa et al. have proposed a system called Vigilante, which uses broadcast of self-certifying alerts (SCA) on an overlay topology [7]. Vojnovic et al. improve the strategy further by using a super host per subnet and show that 100,000 alerts can be disseminated within 300 s in their simulation-based study [17]. In addition, Xie et al. propose different download- and search-based approaches in the P2P network that take 35.5 and 62.5 h, respectively, to achieve 90% of immunized population [18].

Security patch dissemination Li et al. propose a self-organizing overlay network architecture called Revere for security update delivery [2]. However, Revere assumes that the certificate of the dissemination center is known by other nodes in advance and nodes have to trust different certificates of different dissemination centers separately. In this work, we consider a unified framework for update dissemination such that multiple dissemination centers can co-exist but nodes share the same CA hierarchy. This raises the challenge to key management and distribution for different dissemination centers. Moreover, to make update dissemination more scalable and efficient, iDispatcher uses both multicast and overlay gossip unlike Revere which only uses overlay dissemination. In the article [19], Johanses et al. argue that large security updates should be encrypted before dissemination and the decryption key should be disseminated later to shorten the WOV period.

Fast code deployment in data centers Large service providers such as Twitter and Facebook need a fast way to deploy their updated code in their data centers. Twitter recently reported that BitTorrent is used for this purpose [3]. However, they deploy BitTorrent together with several techniques to optimize its performance, such as disabling encryption, disabling DHT, and upload from memory. These optimization methods are only possible in a closed environment, where there are no NAT/firewall issues and every node is trusted. Moreover, as BitTorrent works only in a pull-based fashion, they have to use different protocols such as Capistrano [20] to push the command to individual nodes for downloading the update using BitTorrent. However, they use a central tracker system which can be a bottleneck for such systems. Similar to Twitter, Facebook also uses BitTorrent to disseminate updated code in their data center [9].

3 System architecture

In this section, we first describe the system architecture of iDispatcher from both high and low level perspectives. Then we provide a more detailed discussion regarding key management by iDispatcher.

3.1 High level view: workflow of iDispatcher

The iDispatcher system contains a number of components, including dissemination centers, nodes, CA (Certificate Authority), and community servers as shown in Fig. 1. Figure 2 further shows the workflow of the system, which is described in more detail as follows:

1) Nodes can join and leave the system at any time.
2) A dissemination center (any node with a valid key) starts distributing information to a subset of its neighbors using broadcast or multicast. It appends its digital signature along with the information before dissemination.
3) Nodes receive the information from neighbors using push/pull based scheme.
4) Nodes verify the authenticity of received information using the digital signature to ensure it indeed came from the dissemination center. They may use Distributed Hash Table (DHT) for resolving certificate dependency.
5) After authenticating received information, nodes further participate in information dissemination by sending it to a subset of neighbors. Here, they may tune system parameters locally to strike a balance between latency and bandwidth usage.

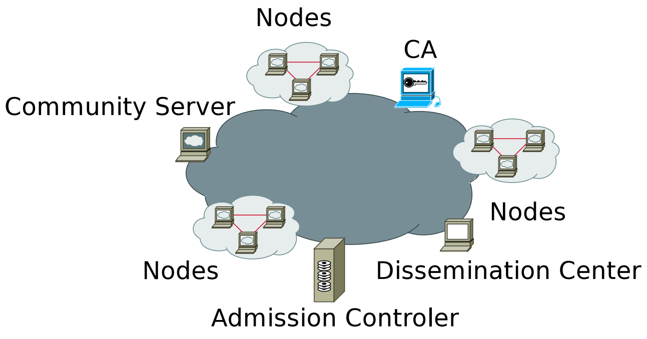


Fig. 1 Components of iDispatcher

From a high level standpoint, iDispatcher resembles existing information dissemination systems that apply gossip protocols in P2P-type of overlay networks. In the following, we present the detail of each module of the system and discuss how iDispatcher meets the requirements of a generic shared information dissemination system.

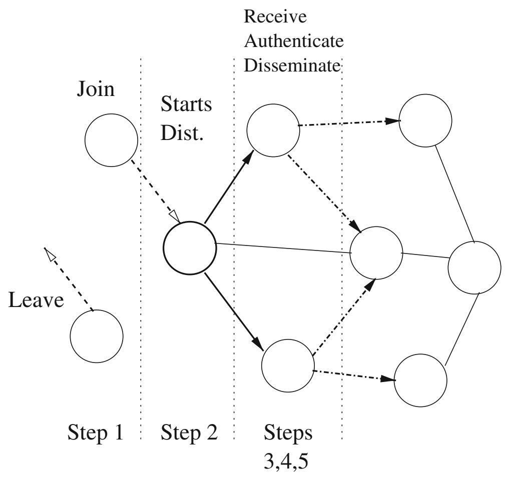


Fig. 2 Workflow of iDispatcher

3.2 Low level view: modules of the system

iDispatcher has a highly modular architecture by design. Figure 3 shows different modules of iDispatcher. We describe the functionality of each module as follows.

Topology creation When a node wants to join the network, it needs to know bootstrapping information such as the IP address and port number of another node already connected with the network. To get such information, the node may consult with some community servers who are responsible for maintaining a list of available nodes in the P2P network. The authenticity of a community server is verified before the new node initiates any further communications with this server. When a new node is connected to a community server, the server replies with a list of available nodes. After getting the list, the new node connects to a random node from the list and thus becomes a part of the network. The random node also provides a random subset of its neighbors to the new node so that the new node can expand its neighbor set by connecting to those nodes.

Distributed Hash Table (DHT) For maintaining a P2P overlay network and having an efficient lookup service, iDispatcher relies on a decentralized Distributed Hash Table protocol. For maintaining overlay network, each node maintains a set of links to other nodes (called neighbors) in its routing table according to its network topology. The network topology in DHT exhibits a property such that for any key k (e.g., file hash), a node's ID either matches with k or has a neighbor whose node ID is closer to k. To publish a key-value pair (e.g., filename with data), a node sends a *store* request with key-value to a neighbor, which is then forwarded to other nodes (based on the closeness between the node ID and the key) in the overlay network until it reaches the node responsible for storing the key with the value. Now, to retrieve the content of the file, a node generates the key k (hash of the file name) and sends a *retrieve* request with k to a neighbor, which is then forwarded to other nodes until it reaches the node having k. Typically, such *store* and *retrieve* operations take $O(logn)$ in an overlay network with n nodes, and thus very efficient [21].

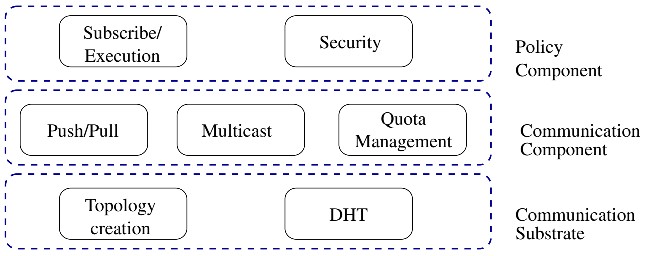


Fig. 3 Different modules of iDispatcher

Multicast To improve information dissemination efficiency, iDispatcher uses a reliable multicasting protocol to disseminate information among nodes in the same subnet. This helps reduce inter-subnet traffic when information is being disseminated in the network. For example, consider a subnet that has 100 nodes inside the network. If most of those 100 nodes receive the same information from outside of the subnet, the inter-subnet traffic will be larger compared to the case when only a few nodes of the subnet receive the information from outside of the subnet and then distribute the information among nodes inside the subnet. The administrator of a network can create a multicast group for iDispatcher and invite other nodes in the same network to join this multicast group. Thus, multicast not only enables nodes to receive information from the multicast group with low latency, but also reduces inter-subnet traffic in the Internet.

Push/Pull To minimize the window of vulnerability, iDispatcher uses a push-based method to disseminate information in the network. However, in a push-based system, it is possible that some nodes may not receive the information from a push campaign. For example, if a node is down during the push campaign and then comes alive after the push campaign is completed, it will not receive that information. In another example, since each node randomly selects a subset of its neighbors to push information, it is possible that some nodes may remain unselected during the dissemination. Therefore, iDispatcher uses pull as a secondary method for receiving missing information from the network. Each node sends the list of all information IDs it received in the last few hours to its neighbors in a certain interval. After receiving such a list, the neighbor can verify whether it has missed any information. If it finds any information missing, it requests the sender node to send that specific information to it.

Quota/Broadcast management When a node receives some information, it forwards it to a subset of its neighbors. The number of neighbors the node selects as the subset is called *quota*. Managing quota is important in such a system because a larger quota increases redundant traffic in the network and a lower quota will make some nodes left out from getting the information. In iDispatcher, quota is tunable. If the information is time critical then a higher quota can be used to decrease dissemination latency with the cost of redundant traffic in the network. On the contrary, if the information is not time critical, then a better-suited quota should be used to achieve more efficient bandwidth usage.

Security In iDispatcher, every dissemination center signs the information it wants to disseminate in the network with its private key and every node validates a dissemination center's identity using its public key upon receiving that information. If the validation fails, the node stops processing the information further. On the other hand, if the validation is successful, the node further sends the information to its neighbors. For such validation, iDispatcher assumes that the key a node trusts and the key the dissemination center uses for signature are part of a single CA (Certificate Authority) hierarchy.

Figure 4 shows an example of certificate chain hierarchy in the system. In such a hierarchical structure, each child certificate is signed by its parent node. If all nodes trust only Root CA (and thus have a local copy of Root CA certificate), that is sufficient for a node to validate any certificates in the system. This is because, if there is any missing certificate between the trusted CA certificate and the information issuer certificate, then the receiver will search those missing certificates in the network and cache them locally once they are found. In Section 3.3, we shall provide more details about key management by iDispatcher.

Subscribe/Execution iDispatcher assumes that multiple dissemination centers (originator of information) may co-exist in the same network and can disseminate information concurrently. But nodes in a network may not be interested in the information disseminated by some dissemination centers. Therefore, each node maintains a subscription list of the identities of dissemination centers from which it is interested in receiving information. The subscribe module ensures that a node will only use the information received only from the dissemination centers that it subscribes to. For information originated from dissemination centers it does not subscribe to, the node will still validate the source and forward to its neighbors after successful validation as a service provided to the community, but itself does not consume the information. It is noted that the current implementation of iDispatcher wastes network resources when nodes not subscribing to a certain information source may still need to forward messages from them. To address this drawback, we plan to explore in the future topology control methods that form overlay network structures to minimize such wastes.

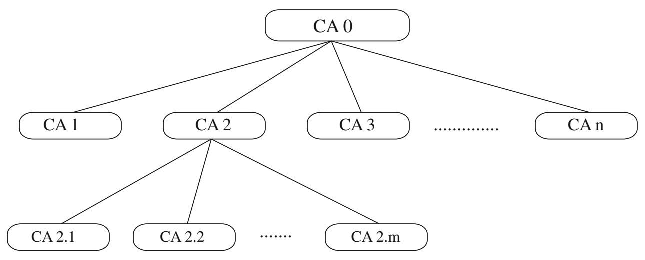


Fig. 4 Certificate chain example

3.3 Key management

Every node must authenticate the dissemination center before they use the information disseminated by that dissemination center. Traditional software update system relies on either transport layer security (i.e. SSL/TSL) or digital signature [22]. iDispatcher uses the digital signature approach for authentication: every disseminati on center appends its digital signature in the information it tries to disseminate so that every recipient node can validate the dissemination center by examining that signature. For a successful validation, the key a dissemination center uses for signature and the certificate a node trusts must belong to the same CA hierarchy. If different dissemination centers use keys that are derived from different CA hierarchies, a node has to trust at least one certificate from each of CA hierarchies and consequently, this will increase the complexity of managing certificates. To reduce such complexity, iDispatcher imposes the restriction that the dissemination center's key and the certificate a node trusts must be derived from a single CA hierarchy. In such settings, it is sufficient for a node to trust only a single certificate to verify any dissemination center using a chain of trust. For example, if a node trusts only certificate P and it receives information signed with a key Q by a dissemination center, it will retrieve intermediate certificates between P and Q and validate Q. Moreover, such a single authority simplifies admission control for dissemination centers in a unified information dissemination platform.

Suppose that there are thousands of dissemination centers and millions of nodes in the network. Then, what is the best way for iDispatcher to manage certificates that dissemination centers use to sign information and those different CA certificates that nodes trust? In the above example, how a node should retrieve all intermediate certificates between P and Q so that it can validate Q? We first explore the following two options here:

1) All certificates are stored in a central server and nodes download all intermediate certificates if nec-

essary from the server. But such a central server solution may pose a severe performance bottleneck due to its poor scalability specially when a large number of nodes try to download a certificate simultaneously. Moreover, if the central server goes down under some DDoS attacks, nodes will not be able to retrieve certificates from it, causing large delays and even unsucessful certification verification at each node.

2) In another approach, the dissemination center appends all the CA certificates to the message it tries to disseminate. This approach, however, imposes a constraint on the length of the CA chain hierarchy. In the traditional SMIME or XMPP signature encoded in emails or IMs (instant messages), respectively, the sender concatenates its own certificate with the payload and thus the receiver does not need to search for the certificate of the sender. This works well as an email or IM is meant to be received by one or a small number of receivers. On the contrary, in iDispatcher, information is disseminated to a large number of nodes and therefore, sending certificate along with the information will significantly increase the amount of traffic in the network. Moreover, the CA hierarchy for email or IM is not large as separate software vendors can use a separate CA hierarchy. But for iDispatcher, there is only one CA hierarchy which can be very large.

Due to these challenges of both methods, iDispatcher uses a different approach to obtain certificates for authentication. Instead of using a central server solution or embedding the full list of certificates in the information, iDispatcher allows each node to search certificates from the P2P network on demand, and cache them locally once they are found. This method reduces the amount of traffic in the Internet because, once a node receives a certificate, it caches the certificate to avoid a redundant search next time when it is needed. iDispatcher naturally uses the DHT module for publishing and retrieving certificates on demand. The flip side of this approach is that when a node needs to search and retrieve certificates from the P2P network to authenticate a received message, this postpones the process of it disseminating the message to its neighbors. However, as we use a gossip-like protocol for information dissemination, such delay should not slow down the whole information dissemination process prominently unless a significant fraction of nodes incur cache misses (e.g., at the early phase of an information dissemination campaign).

4 Security enhancement of iDispatcher

In this section, we discuss possible malicious attacks that can be launched against iDispatcher and as a response how iDispatcher mitigates those attacks.

4.1 Flooding attack

In a flooding attack, an attacker disseminates bogus information into the network. If participating nodes do not validate such bogus information and thus allow it to spread freely in the network, iDispatcher nodes have to waste bandwidth and CPU on processing it, which may lead to severe performance deterioration of normal information dissemination. In iDispatcher, on the arrival of a new message, every node must use the attached digital signature to verify the authenticity of the information source before it forwards the information to its neighbors. Hence, bogus information without a signature signed by a legitimate dissemination center can be stopped immediately one hop away from the attacking node.

An attacker can also replay messages sent from legitimate dissemination centers. These information, if allowed to flow in the network, also degrade the performance of iDispatcher because bandwidth and CPU resources are wasted on processing them. As these messages carry authentic signatures from the legitimate dissemination center, an iDispatcher node cannot distinguish it from normal ones. In iDispatcher, however, there are two mechanisms deployed to mitigate this type of attacks. On one hand, the quota mechanism limits the number of times that a node forwards a legitimate message to its neighbors. On the arrival of a replayed message, even if it is verified to come from a legitimate source, the node stops forwarding it when the quota has been used up. On the other hand, information from dissemination centers carries an expiration time. When a node receives a new message, it first checks whether this message has already expired, and if so it refuses to forward it further. This is important because the attacker may want to disseminate some old patches known to be buggy that he can exploit later if they are installed.

4.2 Pollution and poisoning attack

In a DHT-based P2P file-sharing system, nodes publish titles of files they intend to share. However, attackers can also publish the titles of files that either they do not have (index poisoning attack) or they have a corrupted copy of the file (pollution attack). When a benign node tries to download a file that has been advertised falsely by the attacker, it will fail to download the file or download a corrupted copy.

In iDispatcher, certificates of dissemination centers and CAs are stored in the P2P network. It is thus possible for attackers to mount pollution attacks by publishing corrupted certificates or mount index poisoning attacks by publishing false claims that they have a certificate. When a node needs to download a certificate of a dissemination center or a CA to verify information, it may receive these falsely published messages from the attacker. If the published message is from a poisoning attack, it may delay the verification process by the node because the node has to use other sources to download that certificate successfully. If the published message is from a pollution attack, it will fail the CA chain verification process and the node thus will not forward further the received information.

To counter both pollution and poisoning attacks, iDispatcher requires each node to enforce authentication strictly before it responds to a publishing request: it first downloads the original certificate from the publisher, validates it with the CA chain and then keeps the index of the publisher only if the validation is successful. With such a scheme in place, mounting pollution or poisoning attack becomes difficult in iDispatcher.

4.3 Sybil attack

In a Sybil attack, the attacker creates a large number of false entites and uses those entities to perform illegitimate activities. In iDispatcher, sybil nodes can be used to launch the following attacks:

- They can refuse to cooperate in information dissemination. In an information dissemination campaign, an upstream node uses quota to control the number of neighbors it forwards the message to. So if a downstream node is a sybil node and refuses to forward the message further, it may slow down the whole information dissemination process.
- Sybil nodes can hijack requests from normal nodes and send back bogus information. For example, a sybil node can respond to a request for a CA certificate with a bogus one. Although this type of attacks may slow down information dissemination, its impact is limited because iDispatcher has provided a strong authentication mechanism to stop bogus information from spreading.
- Multiple sybil nodes can even collude to disrupt normal P2P operations. For example, they can create a cut in the P2P topology by manipulating the routing tables, or launch eclipse attacks against a target node (e.g., a dissemination center) such that all communications to the target can be hijacked by sybil nodes. Although this type of attacks is disruptive to iDispatcher, they are difficult to implement in practice, especially when there are only a small fraction of sybil nodes.

Defense against sybil attacks in P2P systems is known to be difficult. We refer interested readers to the literature [23] for a survey of existing solutions. The current version of iDispatcher still does not have a systematic approach to it yet, and this remains as our future work.

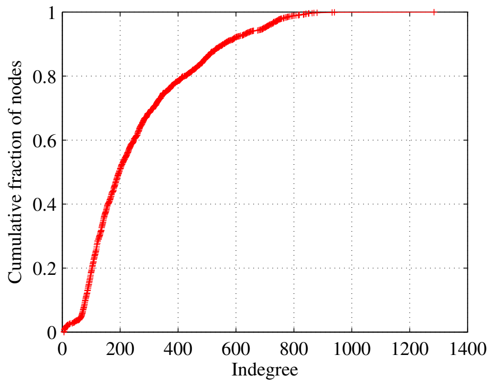
5 Implementation and evaluation

We implemented a prototype of iDispatcher for performance evaluation. The iDispatcher prototype has two separate components: one component is responsible for managing authentication (*Policy Component*) and the other component is responsible for establishing and maintaining connections with the existing P2P overlay network (*Communication component* including the communication substrate). The Policy component handles Security and Subscribe/Execution modules while the Communication component handles the rest of the modules. Both components reside in the same node and talk over TCP to each other. We use two separate components since the Policy Component code is trusted while the Communication Component code may or may not be trusted.

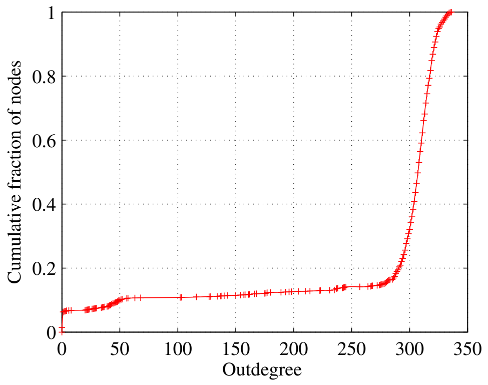
We use the Kademlia [24] protocol for the underlying DHT from aMule-2.1.3 implementation. For reliable multicast, we use the PGM (pragmatic general multicasting) protocol [25]. Information is signed by the private key of a dissemination center in the SMIME format. We used 1024 bit encryption for keys and SHA1 digest hash (using openssl library [26]) for generating digital signature of information and X509 certificates. For DHT signaling, we used the UDP transport protocol where all file transfers are accomplished with the TCP transport protocol. For all experiments, the SMIME-encoded file size was 2KB unless mentioned otherwise.

We study the performance of iDispatcher using about 500 PlanetLab machines distributed around the world running in total about 1,400 virtual nodes. In our experiments, we use one community server for serving bootstrapping information to each new node that wants to join the network. Figure 5 shows the degree distribution of nodes in the network. We see that most of the nodes has around 300 neighbors in the network. Although the outdegree may seem to be high,

Fig. 5 Degree distribution of nodes in the network



(a) Indegree.



(b) Outdegree.

it actually does not affect our experiments due to the quota mechanism in iDispatcher. We will explain this further later.

5.1 Tunable parameters in iDispatcher

iDispatcher uses the following parameters for tuning performance in terms of distribution latency, message flow overhead and reachability:

Selective push During an information dissemination campaign, every node pushes information to a subset of its neighbors. In this strategy, a node downloads the same information from multiple neighbors. By contrast, with *selective push*, a node first queries its neighbors about whether they have received the information or not. If a neighbor has already received that specific information, it rejects the request; otherwise it accepts the request and downloads the information. This selective push strategy ensures that a node does not download the same information from multiple neighbors and thus saves valuable bandwidth and CPU time.

Quota Another important factor is *quota*, the number of neighbors a node forwards information to. Clearly, if we increase the quota, the dissemination latency should decrease with an increased number of information download requests from its neighbors.

Improved selective push The quota mechanism prevents a node from forwarding information to its neighbors who are not included in its quota set. Due to this, a node may only forward the information to its neighbors which have already received it but fails to forward the information to those neighbors that have not received it so far. Therefore, some nodes in the network may not receive the information at all. To tackle this problem, we use improved selective push such that, if a node finds that its neighbor in the quota set already received the information, it will keep trying to find a new node who have not received that information so far. This improves the distribution coverage as shown in Section 5.2.3.

Thread pool size We define a *thread pool* as a collection of threads iDispatcher is allowed to use during information dissemination. There are two sets of thread pools, *sender thread pool* and *receiver thread pool*. We will show that *thread pool size* affects iDispatcher performance in Section 5.2.4.

Pull In the push-based method, it is possible that some nodes may fail to receive disseminated information if the quota is too low or the node was down during the push campaign. To address this issue, iDispatcher can also use a pull-based method as discussed in Section 3.2.

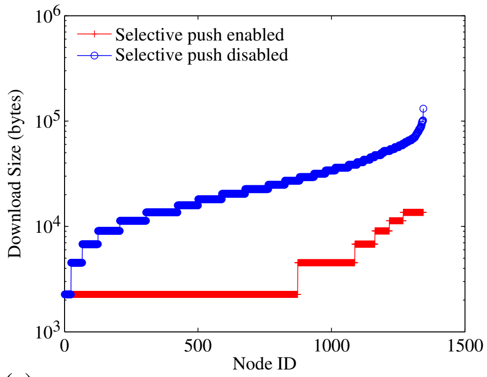
5.2 The effect of tunable parameters

In this part, we study the effect of different system parameters and design choices, including *selective push* (enabled by default), *quota* (15 by default), *improved selective push* (disabled by default), *thread pool size* (5 by default), and *pull-based scheme* (disabled by default). We are particularly interested in the dissemination latency, which is the amount of time by which all nodes receive information after the dissemination center sends out the information.

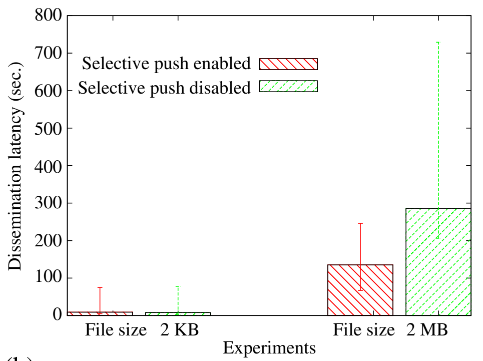
5.2.1 Effect of selective push

Figure 6 shows the effect of selective push on the download size and latency per node. As shown in Fig. 6a, we see that the download size in terms of bytes per node increases when selective push is disabled. In contrast, the download size is almost constant per node when selective push is used. This is because, with this strategy enabled, every node downloads the information exactly

Fig. 6 Effect of selective push on (**a**) download size and (**b**) dissemination latency



(a) Effect of selective push on the number of download requests received per node; Selective push significantly decreases download data size per node.



(b) Effect of selective push on dissemination latency. It is more prominent when file size is bigger. Errorbar shows 5th and 95th percentile latency.

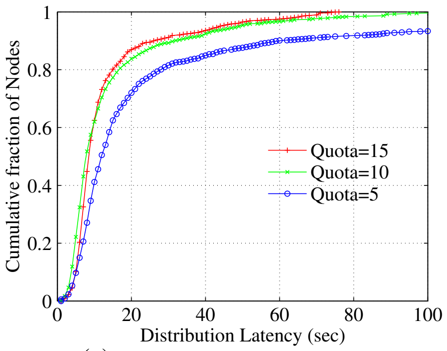
once although it may receive the push request multiple times. As shown in Fig. 6a, a few nodes download the information more than once due to concurrency issues. That means, those nodes received multiple download requests from their neighbors before any download was completed and therefore accepted all requests. Moreover, we find in Fig. 6b that, dissemination latency decreases when selective push is enabled because it makes available more resources, such as free threads in the thread pool. If the amount of information to be disseminated is larger, transmitting it will occupy resources for a longer time, which makes the effect of selective push on dissemination latency more prominent.

5.2.2 Effect of quota

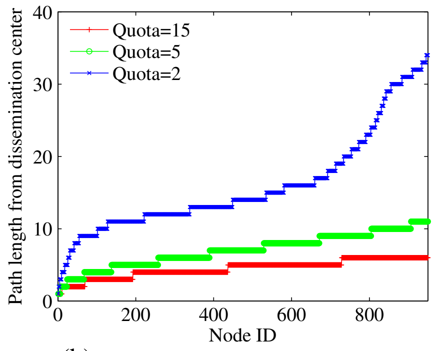
In Fig. 7a, we show the relationship between quota and dissemination latency: increasing quota helps reduce dissemination latency. This is because increasing quota effectively decreases the path length (as shown in Fig. 7b) between the dissemination center and each node so nodes receive information more likely in a shorter period of time. On the other hand, increasing quota also increases the number of push requests received per node as shown in Fig. 7c. However, using a lower quota poses the risk that some nodes may not receive information at all. For time critical dissemination, it is better to use a high quota while for non-time critical dissemination, a moderate quota is more appropriate.

5.2.3 Effect of improved selective push

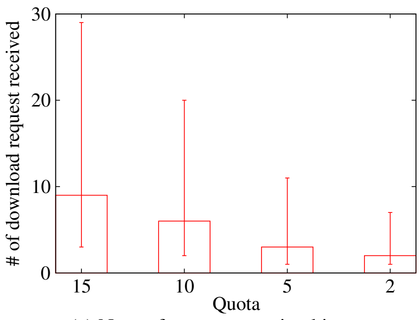
Improved selective push improves the distribution coverage as shown in Fig. 8a. However, this solution incurs an overhead of sending a large number of request messages to its neighbors for finding new nodes. This overhead is shown in Fig. 8b where nodes end up with sending the request message to almost all its neighbors (similar to the indegree distribution as shown in Fig. 5a). Therefore, improved selective push policy should only be used for time-critical information dissemination.



(a) Quota is inversely proportional to latency.



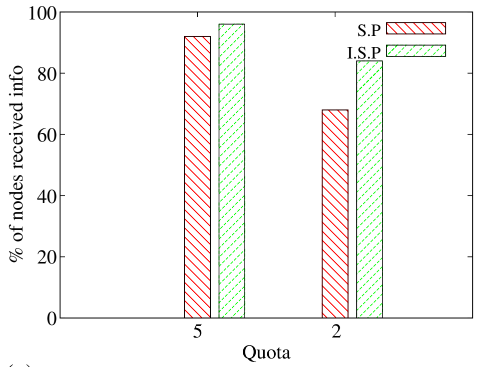
(b) Path length decreases with increase of quota.



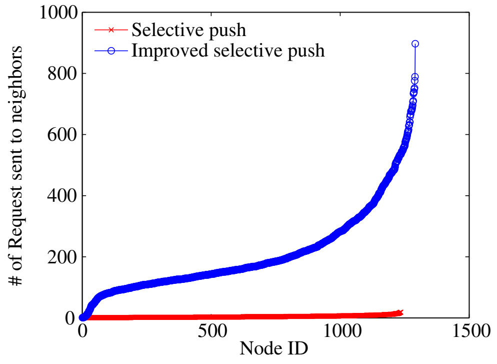
(c) Num of requests received increases with quota.

Fig. 7 Effect of varying quota on (**a**) dissemination latency, (**b**) path length and (**c**) the number of requests received per node (with 5th and 95th percentile errorbars)

Fig. 8 Effect of improved selective query on (**a**)distribution coverage and (**b**)number of requests received



(a) Distribution coverage with and without improved selective push (S.P: selective push; I.S.P: improved selective push). Improved selective push increases distribution coverage.



(b) Effect on the number of requests with and without improved selective push. Improved selective push increases the number of requests sent to neighbors significantly.

5.2.4 Effect of thread pool size

In this experiment, we quantify the effect of the thread pool size on dissemination latency. Once a node receives information, the node forwards it to a subset of its neighbors in two ways: it can send the information to them serially one after another (the sender thread pool size is one) or in parallel (the sender thread pool size is x which is less or equal to the size of quota). Also, using a higher number of receiver threads allows a node to receive information from its neighbors concurrently. Thus, increasing the thread pool size effectively increases the parallelism, which helps decrease the dissemination latency. Figure 9 depicts such relationship between the thread pool size and the dissemination latency. Here, thread pool size x means that both the sender thread pool and the receiver thread pool have x threads available.

5.2.5 Effect of pull

In this experiment, we show the effect of the pull-based method as shown in Fig. 10. Here, after the push dissemination campaign is finished, some new nodes joined the network. As we see in Fig. 10, those new nodes also received the information using the pull-based method.

5.3 Effect of dynamic tunability

In this experiment, we show that how iDispatcher configures its parameters and decision choices based on requirements of different applications. iDispatcher associates a "critical level" with the information to be disseminated. Based on the critical level value, nodes adjust their tunable parameters according to Table 2. Here, we assume that such a table is lo-

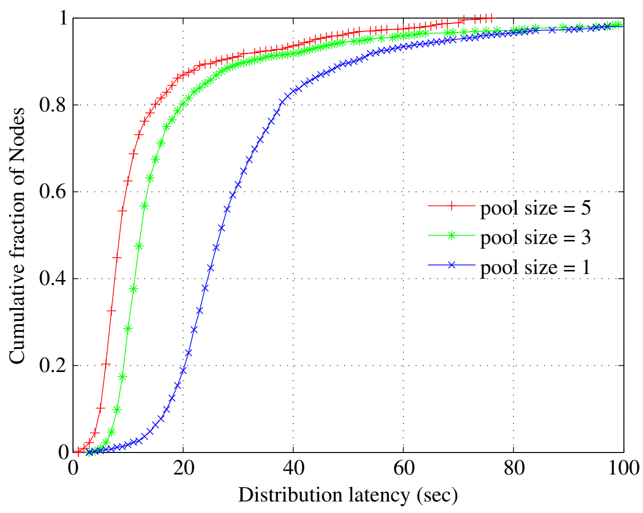


Fig. 9 Effect of thread pool size on latency. Increasing thread pool decreases latency

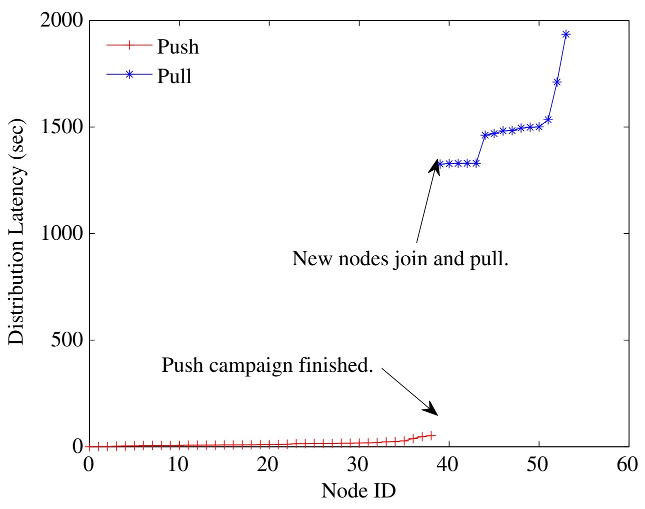


Fig. 10 Effect of pull on information received for newly joined nodes

Table 2 System parameters used for different critical levels

Critical level	Selective push	Quota	Improved selective push	Thread pool size
High	Yes	20	Yes	5
Medium	Yes	15	No	3
Low	Yes	10	No	2

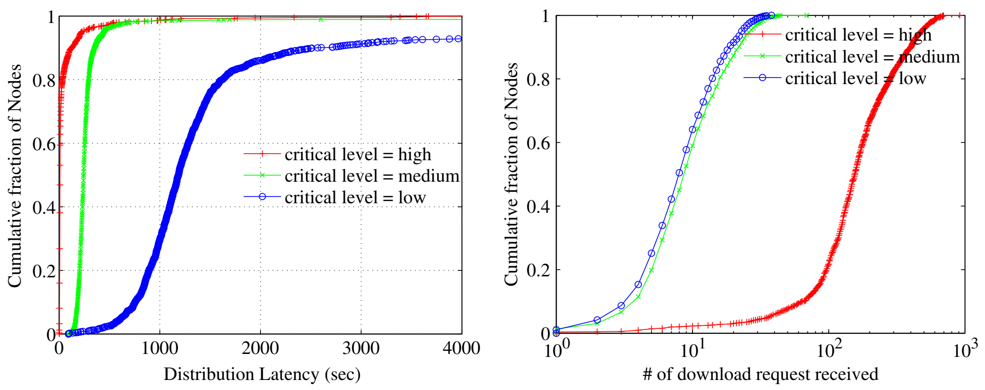
cally stored at each node. We show the dissemination latency for high, medium and low critical information in Fig. 11a. We used 2KB, 2MB and 15MB for high, medium and low critical information size, respectively. We assume that time critical information such as firewall rule update or worm signatures should have small information sizes while regular software update may have large information sizes. We see that iDispatcher has lower latency for time-critical information dissemination and lower bandwidth usage for non time-critical information with lower quota. Hence, lower distribution latency is achieved at the cost of a higher number of redundant messages in the network as shown in Fig. 11b and therefore, not all information dissemination should use high time-critical dissemination.

5.4 Effect of flooding attack

iDispatcher supports application level scalability. That means, any node can act as a dissemination center and can disseminate information to the network. In this experiment, we analyze the effect of flooding bogus information by some malicious nodes in a network of 85 nodes. In the first scenario, one benign node disseminates information and security is disabled on each node (**No Sec**). In the second scenario, eight percent of nodes start attacking the network by disseminating bogus information along with a benign node sending benign information and the security is disabled on each node (**No Sec** + **Attack**). In the third scenario, security is enabled on each node and one single benign node disseminates information in the network (**With Sec**). In the last scenario, again eight percent of nodes starts attacking the network by disseminating bogus information along with a benign node sending benign information and security is enabled on each node (**With Sec** + **Attack**).

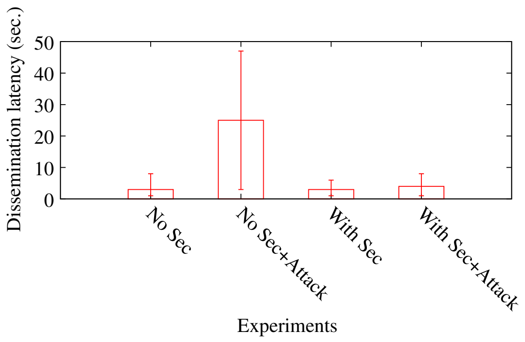
As shown in Fig. 12a, we see that the median dissemination latency for benign information dissemination increases when an attack is launched with security disabled in the second (No Sec+Attack) scenario. However, with security being enabled (Sec+Attack), we find that the dissemination latency is almost the same with that in the third scenario (With Sec). This is because, with security enabled, the malicious nodes can affect neighbors at most one hop away as discussed in Section 4.1. Once a neighbor receives information from a malicious node and fails to validate it, that node just rejects it and does not forward it further to other neighbors. This shows the robustness of iDispatcher under flooding attacks. From Fig. 12a, we can see that security validation does not increase the dissemination latency significantly as the dissemination latency is similar under the first (No Sec) and the third (With Sec) scenarios.

In Fig. 12b, we also see that the number of messages disseminated in the network in the second (No Sec+Attack) scenario is higher than that in the fourth (With Sec+Attack) scenario. This is because, after one

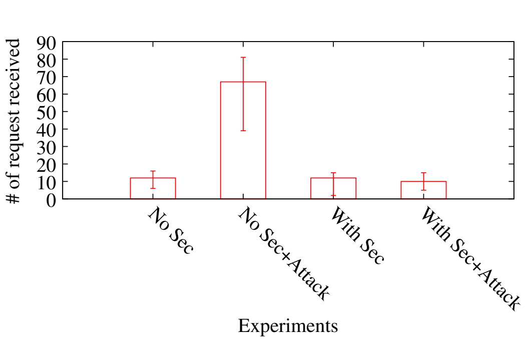


(a) Higher critical level decreases latency significantly. (b) Higher critical level increases the number of requests.

Fig. 11 Effect of dynamic tunability at different critical levels on (**a**) dissemination latency and (**b**) the number of requests received per node



(a) Dissemination latency does not increase during flooding attack when security is enabled.



(b) Number of requests received per node does not increase during flooding attack when security is enabled.

Fig. 12 Effect of flooding attack in iDispatcher. Plots show median valudes with 5th and 95th percentile errorbars

hop, all malicious information stops propagating further as it fails to pass the security check.

6 Discussion

In contrast to previously proposed information dissemination platforms as discussed in Section 2, iDispatcher is designed to be a tunable, general-purpose information dissemination framework. Therefore, iDispatcher can be used for different information dissemination applications by adjusting its tunability features such as quota, thread pool size and improved selective push grouped in critical level. For software update distribution or fast code dissemination in the data center, traditional mechanisms use pull-based approaches, which is slow. To make the dissemination faster, iDispatcher uses push-based approach. Typically such services are not time critical, therefore, a medium or low critical level can be used for dissemination in iDispatcher. However, a higher critical level needs to be used in iDispatcher for alert dissemination against worm propagation or security patch dissemination to minimize the window of vulnerability. Thus, tunability has made iDispatcher a general-purpose framework that supports a wide range of information dissemination applications.

7 Conclusion

In this paper, we propose iDispatcher, a tunable and flexible information dissemination tool which provides a single unified platform for a large number of dissemination centers and a massive number of participant nodes. We implement and evaluate the performance of iDispatcher on the PlanetLab. We show the robustness of iDispatcher under flooding attacks. With its high modularity, tunability and flexibility, iDispatcher can be used for information dissemination by different software vendors for software updates, or by system administrators to broadcast security rule update in enterprise networks.

References

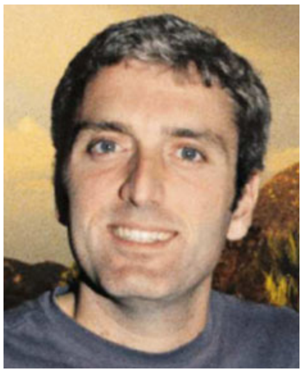
1. Gkantsidis C, Karagiannis T, VojnoviC M (2006) Planet scale software updates. In: SIGCOMM '06: proceedings of the 2006 conference on applications, technologies, architectures, and protocols for computer communications. ACM, New York, NY, USA, pp 423–434
2. Li J, Reiher P, Popek G (2004) Resilient self-organizing overlay networks for security update delivery. IEEE J Sel Area Comm 1:189–202
3. Fast code dissemination in Twitter data center. http://engineering.twitter.com/2010/07/murder-fast-datacenter-code-deploys.html. Accessed Feb 2011
4. Code-Red worm propagation. http://www.caida.org/research/security/code-red/coderedv2_analysis.xml. Accessed Feb 2011
5. Deshpande M, Xing B, Lazardis I, Hore B, Venkatasubramanian N, Mehrotra S (2006) Crew: a gossip-based flash-dissemination system. In: Proceedings of the 26th IEEE international conference on distributed computing systems, ICDCS '06. IEEE Computer Society, Washington, DC, USA, pp 45
6. Wu C-J, Li C-Y, Yang K-H, Ho J-M, Chen M-S (2009) Time-critical data dissemination in cooperative peer-to-peer systems. In: Proceedings of the 28th IEEE conference on Global telecommunications, GLOBECOM'09. IEEE Press, Piscataway, NJ, USA, pp 2942–2947
7. Costa M, Crowcroft J, Castro M, Rowstron A, Zhou L, Zhang L, Barham P (2005) Vigilante: end-to-end containment of internet worms. In: Proceedings of the symposium on Systems and Operating Systems Principles (SOSP), pp 133–147

8. Adams J (2010) Operations at Twitter: scaling beyond 100 million users. LISA. http://www.usenix.org/event/lisa10/tech/slides/adams.pdf
9. Fast code dissemination in Facebook data center. http://torrentfreak.com/facebook-uses-bittorrent-and-they-love-it-100625/. Accessed Feb 2011
10. Delaet T, Joosen W, Vanbrabant B (2010) A survey of system configuration tools. In: Proceedings of the 24th international conference on large installation system administration, LISA'10. USENIX Association, Berkeley, CA, USA, pp 1–8
11. iDispatcher: implementation and source codes. http://www.cs.ucr.edu/~rahmanm/iDispatcher/. Accessed June 2011
12. PlanetLab. An open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org/. Accessed Feb 2011
13. Red Hat network. http://www.redhat.com/red_hat_network/. Accessed Feb 2011
14. Mac OS X. Updating your software. http://support.apple.com/kb/HT1338?viewlocale=en_US. Accessed Feb 2011
15. Torrent Fedora project. http://torrent.fedoraproject.org/. Accessed Feb 2011
16. Serenyi D, Witten B (2008) Rapidupdate: peer-assisted distribution of security content. In: IPTPS 2008, the 7th international workshop on peer-to-peer systems, pp 423–434
17. Vojnovic M, Ganesh AJ (2008) On the race of worms, alerts, and patches. IEEE/ACM Trans Netw 16:1066–1079
18. Xie L, Song H, Zhu S (2008) On the effectiveness of internal patching against file-sharing worms. In: Proceedings of the 6th international conference on applied cryptography and network security, ACNS'08. Springer-Verlag, Berlin, Heidelberg, pp 1–20
19. Johansen HD, Johansen D, van Renesse R (2007) Firepatch: secure and time-critical dissemination of software patches. In: SEC, pp 373–384
20. Capistrano. http://en.wikipedia.org/wiki/Capistrano. Accessed Feb 2011
21. Distributed hash table (dht). http://en.wikipedia.org/wiki/Distributed_hash_table. Accessed Dec 2011
22. Samuel J, Mathewson N, Cappos J, Dingledine R (2010) Survivable key compromise in software update systems. In: Proceedings of the 17th ACM conference on computer and communications security, CCS '10. ACM, New York, NY, USA, pp 61–72
23. Levine BN, Shields C, Margolin NB (2006) A survey of solutions to the Sybil attack. Tech Rep 2006-052, University of Massachusetts Amherst
24. Maymounkov P, Mazières D (2002) Kademlia: a peer-to-peer information system based on the xor metric. In: IPTPS '01: revised papers from the first international workshop on peer-to-peer systems. Springer-Verlag, London, UK, pp 53–65
25. PGM Reliable Transport Protocol Specification (2001) RFC 3208 (Experimental)2001
26. Openssl library. http://www.openssl.org/. Accessed Feb 2011

Md Sazzadur Rahman is a PhD student in department of Computer Science, UC Riverside. His research interets lies in system, networks and security. His email is rahmanm@cs.ucr.edu and mailing address: Networking lab, Department of computer science, 900 University ave, University of Califonia, Riveside.

Guanhua Yan is a research scientist in the Information Sciences Group at Los Alamos National Laboratory. His mailing address: P. O. Box 1663, MS B256, Los Alamos National Laboratory, NM 87545.

Harsha V. Madhyastha is an Assistant Professor in the Department of Computer Science and Engineering at UC Riverside. He received his M.S. and Ph.D. degrees in Computer Science and Engineering from the University of Washington, and his B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Madras. He is the recipient of the NSF CAREER award, a NetApp Faculty Fellowship, and best paper awards from USENIX NSDI and ACM SIGCOMM IMC.



Michalis Faloutsos is a faculty member at the Computer Science Dpt in University of California, Riverside. He got his bachelor's degree at the National Technical University of Athens and his M.Sc and Ph.D. at the University of Toronto. His interests include, Internet protocols and measurements, peer-to-peer networks, network security, BGP routing, and ad-hoc networks. He is actively involved in the community as a reviewer and a TPC member in many conferences and journals. With his two brothers, he co-authored the paper on powerlaws of the Internet topology (SIGCOMM'99, which is one of the top ten most cited papers of 1999. His most recent work on peer-to-peer measurements have been widely cited in popular printed and electronic press such as slashdot, ACM Electronic News, USA Today, and Wired. Most recently he has focused on the classification of traffic and identification of abnormal network behavior. He also works in the area of Internet routing (BGP), and ad hoc networks routing, and network security, with emphasis on routing.

Stephan Eidenbenz is a scientist in the Information Sciences group at Los Alamos National Laboratory. His research interests are in network security and modeling and simulation. Email: eidenben@lanl.gov mailing, Mailing address: 998 Information Sciences Group (CCS-3) Los Alamos National Laboratory , Los Alamos, NM.

Mike Fisk is Director of Cybersecurity R&D Programs and the Advanced Computing Solutions Office at Los Alamos National Laboratory. Mailing address: MS M311 LANL, Los Alamos NM 87545.