Device Pairing Based on Adaptive Channel Fluctuation Control for Large-scale Organizations

Takashi Oshiba and Hideaki Nebayashi
Service Platforms Research Laboratories, NEC Corporation, Japan
{oshiba@cp, nebayashi@ap}.jp.nec.com

Abstract

We are developing device pairing methods that enable a user to establish two devices, such as a mobile phone and a PC as a device pair. In our previous developed method, a user can create a device pair by simultaneously clicking the pairing buttons shown on their screens. However, when the system handles many users, clicks from different users could overlap, resulting in request collisions.

In this paper, we propose a device pairing method that can keep the collision probability under a permissible limit by using multiple pairing buttons. In this method, a single pairing button with a numeric identifier is shown on a mobile phone, and multiple buttons with different numeric identifiers are shown on a PC. First, a user chooses the button on the PC with the same identifier as the one on the phone. Then, the user simultaneously clicks the single button on the phone and the corresponding button on the PC. A device pair is created based on not only the simultaneity of the two clicks but also the correspondence between the two buttons' identifiers.

If too many pairing buttons are shown on a PC, it will be bothersome for the user to choose one button from many candidates: the smaller the number of pairing buttons shown on a PC, the simpler the user operation. To ensure ease of user operation, our method adaptively fluctuates the number of pairing buttons shown on a PC, i.e., the number of device pairing channels, according to the latest amount of pairing requests from users.

1. Introduction

These days, many business users use multiple devices, such as mobile phones and PCs during a working day [1]. If the multiple devices could work cooperatively, the productivity of the business users would be improved. Let's consider potential scenarios.

(a) **Showing e-mails only from a phone caller on a phone callee's PC**: In this scenario, when user A makes a phone call to user B and they begin to talk with their mobile phones, recent e-mails sent only from user A to user B are automatically shown on a user B's PC. With this cooperative behavior between the user B's mobile phone and the user B's PC, user B can easily find the recent e-mails from user A without searching them manually from a lot of past e-mails sent from various other users. Hence, they can immediately begin discussing about a user A's recent e-mail.

(b) **PC-to-PC Web conferencing during a phone-to-phone audio conversation**: In this scenario, when user A and user B begin to talk with their mobile phones, a user A's PC and a user B's PC launch a Web conferencing application automatically and connect each other. With these cooperative behaviors among these four devices, user A and user B can immediately begin discussing about business documents stored in their PCs, e.g., sales reports and/or product specifications, with the Web conferencing application.

In both these scenarios, a mobile phone and a PC must be established as a pair by a device pairing process. This is an essential aspect of cooperative multi-device scenarios.

We are developing device pairing methods [2]. The new method presented in this paper is intended to be used in large-scale organizations such as large enterprises. It aims for good scalability to handle numerous pairing requests from the many users in a large-scale organization and for low-cost deployment of the device paring system in a large-scale organization. Assuming that every user performs ten device pairings per day on average in an organization with over several ten thousand users, the system must be able to process several hundred thousand pairing requests per day. Moreover, a large-scale organization usually owns various devices such as mobile phones, PCs, and personal digital assistants (PDAs) and various networks such as wireless and wired ones. Thus, the

need to modify existing devices or networks when a device pairing system is deployed should be minimized.

In this paper, we propose a novel device pairing method that can achieve both sufficient scalability and low-cost deployment in large-scale organizations.

2. Problems and Requirements of Device Pairing

2.1. Problems of Conventional Methods

There is a lot of prior work on device pairing that mainly focused on mobile phones and/or small sensors in wireless networks.

The movement-based method [3] automatically creates a pair of small sensors based on acceleration data from the sensors' accelerometer. Media access control (MAC) frames are broadcast in a wireless network to exchange acceleration data, and a device pair is then created. However, if there are too many sensors in the wireless network, the number of broadcasts is large, so MAC frames frequently collide. Furthermore, the broadcast range is limited to just a single local area network (LAN). Thus, the maximum number of sensors that could be handled was reported to be only 100 [3].

Devices with built-in accelerometers can be paired by simultaneously shaking both devices [4]. However, the devices must be equipped with special hardware (in this case an accelerometer). The same problem exists in [3]. In BEDA [5], a device pair is created by sequentially pushing the devices' buttons seven times. However, special software must be installed in the devices in advance. Moreover, in BEDA, the two devices must use the same wireless protocol; therefore, it is not possible to pair with Bluetooth device with an IEEE 802.11b/g device, for example. In SyncTap [6], a device pair is created by simultaneously pushing the devices' buttons. However, SyncTap has the same two problems as BEDA. Seeing-Is-Believing [7], which uses a two-dimensional barcode and a camera, also has the same two problems as BEDA.

2.2. Requirements for Device Pairing

Given the problems with the conventional methods, we identified two requirements for device pairing in a large-scale organization.

(1) **Limit on collision probability:** The collision probability of pairing requests from users must be kept below a certain limit even if the frequency of pairing requests increases as the number of users increases.

(2) **Low-cost deployment:** The cost of deploying a device pairing system must be low even if there are various types of devices and networks. No additional special hardware or software must be needed for the large number of devices in a large-scale organization. The system must be independent of the type of network, such as wireless or wired.

3. Proposal of a Scalable and Adaptive Device Pairing Method

Here, we propose a novel device pairing method that satisfies the two requirements described above. It is an extension of our previous method [2], which we hereinafter refer to as the basic method. Our method comprises a multi-channel-based device pairing control and an adaptive channel fluctuation control, which are described in Sections 3.2–3.4 and Section 3.5, respectively. The former ensures scalability and the latter ensures ease of user operation.

3.1. Outline of Basic Method

Here, we outline the basic method as an introduction to the multi-channel-based device pairing control.

In the basic method, when a user simultaneously clicks pairing buttons (i.e., buttons of a graphical user interface) on the screens of a mobile phone and a PC, two pairing requests are sent—one from each device—to a device pairing server (DPS). When the first request arrives, the DPS starts an acceptance timer. After a predefined time, e.g., 1.0 seconds, the acceptance period ends. If the DPS has received another pairing request within the acceptance period, it creates a device pair. Namely, in the basic method, a pair is created on the basis of only timestamp information. Consequently, if requests from two different users overlap, the DPS will receive four pairing requests. As a result, a device pairing collision will occur and the DPS will be unable to understand the relationship between the two users and four devices.

In the basic method, even if a collision occurs, the users can establish the correct pairs in an intuitive manner. After the end of the acceptance period, the system draws a still image on each screen. If the images on a user's two devices are the same, he/she can establish the pair by clicking the OK button. On the other hand, if the image on the mobile phone does not match the one on the PC, it means that the system has guessed wrong about the pairing. The user can correct the mismatched pair to the desired one by changing the image on the PC so that it matches the one on his/her mobile phone. However, this requires the user to make a bothersome manual change and could lead to security issues if users maliciously or accidently make the wrong choice. Thus, the first objective is to prevent collisions in order to achieve easy user operation.

3.2. Multi-channel-based Device Pairing Control

In our method, a pair is created on the basis of not only timestamp information, but also identifier information added to the pairing request to improve scalability. When a user begins a device pairing

operation, a single pairing button with an identifier chosen by the DPS is shown on the phone screen, and multiple pairing buttons each with an identifier are shown on the PC screen. On the PC, the user chooses the button with the same identifier as the one on the phone from the multiple candidates.

The protocol used in our method is HTTP; therefore, arbitrary devices with only a Web browser can be used.

3.3. User Operation

A user can begin a device pairing operation by accessing the DPS and opening a pairing Web page.

In the basic method, a single pairing button is shown on the phone screen. In our method, on the other hand, a button with a numeric identifier is shown on the pairing Web page on phone's screen. The value of the identifier is chosen by the DPS from 1 to *S*, where *S* is a positive integer. In the basic method, the PC also shows one pairing button, while in our method the PC shows a pairing Web page having *S* pairing buttons with different numeric identifiers (Figure 1).

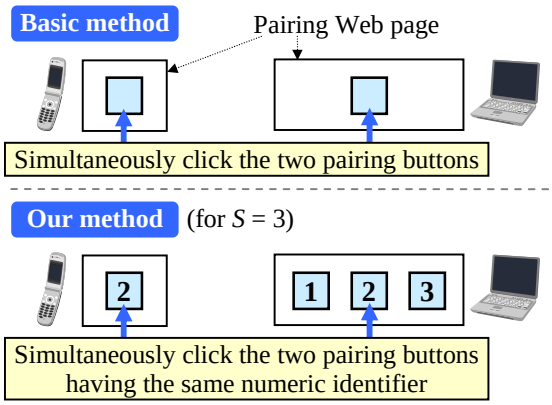

Figure 1: User operation.

In our method, the user simultaneously clicks the single button on the phone and the corresponding button on the PC. Each device then sends a pairing request to the DPS, which starts the device pairing process.

3.4. System Architecture

The architecture of a device pairing system using our method is shown in Figure 2. The parallel degree optimizer determines the number of buttons that will be shown on a PC, i.e., *S*. The identifier selector chooses a numeric identifier that will be shown on a phone. For every request from a pairing Web page on a phone, the identifier selector chooses a numeric identifier from 1 to *S* by round-robin scheduling. Each time-based pairing channel corresponds to a device pairing channel and executes the basic method, i.e., device pairing based on timestamp information, as described in Section 3.1. The number of channels that can be active concurrently, i.e., the number of device pairing channels, is set to *S* by the parallel degree optimizer.

When a user simultaneously clicks pairing buttons on a mobile phone and a PC, the identifier-based load balancer receives two pairing requests that include identifiers. It assigns them to a time-based pairing channel on the basis of the identifier in the requests. After the time-based pairing channel has established the phone and PC as a pair, the pairing data is stored in the pairing storage.

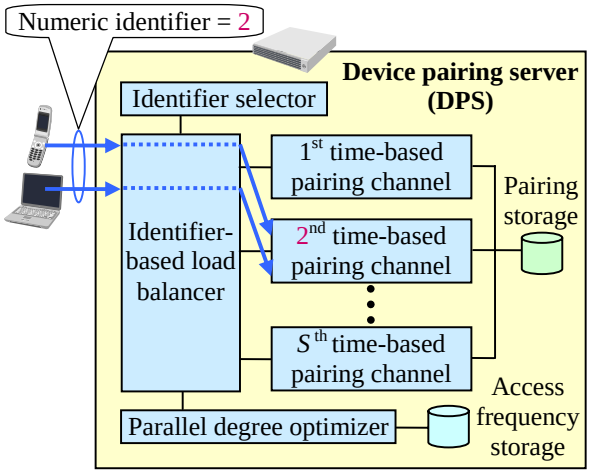

Figure 2: Architecture of system using our method.

Since the identifier selector chooses the identifier by round-robin scheduling, when user A's mobile phone displays a pairing Web page after user B's mobile phone had displayed one, the identifiers shown on these two phones certainly differ. Hence, even if the two users' clicks overlap in time, the identifier-based load balancer can distinguish the two users by checking the identifiers. Since the pairing requests from two users are assigned to different time-based pairing channels, collisions can be avoided.

3.5. Adaptive Channel Fluctuation Control

If too many pairing buttons are shown a pairing Web page on a PC, choosing a button from a lot of candidates is bothersome for the user; therefore, fewer pairing buttons on a PC leads to simpler user operation. Consequently, the relationship between scalability and ease of user operation is a trade-off.

To balance this trade-off, we present the adaptive channel fluctuation control which can improve ease of user operation in the multi-channel-based device pairing control.

The adaptive channel fluctuation control dynamically adapts to the temporal variation in the amount of pairing requests from users and adaptively minimizes the number of time-based pairing channels *S*, which is also the number of pairing buttons shown on a PC, to keep the collision probability below a

permissible limit. Consequently, both scalability and ease of user operation can be ensured.

3.5.1. Model of Multiple Device Pairing Channels

Here, we model our multiple device pairing channels by using queuing theory. We assume that the arrival process of pairing requests obeys a Poisson distribution. The service time is always fixed to the acceptance period, so the service time distribution is a degenerate distribution. The number of servers is *S*. Hence, our method can be expressed by the *M*/*D*/*S*(0) loss model. If *S* = 1, our method can be expressed by the *M*/*D*/1(0) loss model, which is the same as the basic method.

When all the servers in the *M*/*D*/*S*(0) loss model are busy, i.e., providing service, if another request arrives while all the servers are busy, blocking occurs. Thus, the blocking probability corresponds to collision probability B_S. In our method, a collision occurs only if the DPS receives pairing requests from *S* + 1 or more users during an acceptance period. For example, assuming that *S* = 9 and the acceptance period is 1.0 s, a collision occurs only if ten or more users send pairing requests during this 1.0-s period; therefore, our method can make collisions quite rare.

3.5.2. Principle of Adaptive Channel Fluctuation Control

The parallel degree optimizer periodically observes the number of users who have sent pairing requests as the access frequency λ and stores λ into the access frequency storage shown in Figure 2. Namely, λ is half the number of pairing requests. Let $B_{\max}$ be the maximum permissible value of collision probability B_S. The parallel degree optimizer calculates B_S on the basis of λ and finds the minimum *S* that satisfies $B_S < B_{\max}$ as the optimal number of time-based pairing channels S_{opt} (details given in Section 3.5.4). As λ temporally varies, S_{opt} is adaptively changed. By proactively increasing S_{opt} before B_S exceeds $B_{\max}$, the system can decrease B_S and hence certainly prevent B_S from exceeding $B_{\max}$.

3.5.3. Smoothing and Prediction of Access Frequencies

Since λ tends to include accidental error, if we calculate B_S directly from λ for every observation, S_{opt} may become unstable. Accordingly, we deal with the periodically observed values of λ as time series data, and we can remove the observed error by smoothing the time series data. Moreover, through the smoothing, we calculate λ_{next} as the predicted value of λ at the next observation. S_{opt} can be stabilized by calculating B_S from λ_{next}. By using λ_{next}, we can proactively control S_{opt} before B_S exceeds $B_{\max}$.

We use exponential smoothing based on Brown's linear trend model [8], which has better pursuit performance against the trend variation of time series data than a simple moving average. We calculate λ_{next} by using Equation (1).

$$
\begin{aligned}
& f_1(1) = null, \quad f_1(2) = \lambda(1), \\
& f_1(t) = \alpha \cdot \lambda(t) + (1-\alpha) f_1(t-1), \\
& f_2(t) = \alpha \cdot f_1(t) + (1-\alpha) f_2(t-1), \\
& \lambda_{next} = 2 f_1(t) - f_2(t) + \frac{\alpha}{1-\alpha}\{f_1(t) - f_2(t)\},
\end{aligned}
\tag{1}
$$

where $\lambda(t)$ is the latest value of λ in the periodical observations and α is a smoothing factor ($0 < \alpha < 1$). The closer α approaches 1, the greater the influence of the recently observed value.

3.5.4. Optimal Number of Device Pairing Channels

We use the Erlang-B formula [9] to calculate the collision probability B_S (Equation (2)).

$$
B_S = \frac{\frac{(\lambda_{next} h)^S}{S!}}{\sum_{n=0}^{S} \frac{(\lambda_{next} h)^n}{n!}}, \tag{2}
$$

where the constant value h is the duration of the acceptance period. By using Equations (1) and (2), the parallel degree optimizer finds the minimum *S* that satisfies $B_S < B_{\max}$ as the optimal number of time-based pairing channels S_{opt}. Thus, our method dynamically adapts to the temporal variation in access frequency λ, and the number of time-based pairing channels, i.e., the number of pairing buttons shown on a PC, is always optimized to S_{opt}. Hence, scalability and ease of user operation can be ensured.

3.6. Implementation

We implemented a prototype device pairing system that uses our method. The DPS is implemented as a Web application server. We used voice-over-wireless-LAN (VoWLAN) mobile phones that each had a Web browser, e.g., N906iL [10], and PCs with Microsoft Internet Explorer as client devices. A snapshot of our prototype system is shown in Figure 3. Since a Web browser does not need to support JavaScript or cookies, various devices with Web browser other than mobile phones or PCs, e.g., PDAs, can be used in an out-of-box manner. Since HTTP is used as a network communication protocol, our system is independent of the type of network, such as wireless or wired. Hence, our system can be deployed with low cost into large-scale organizations.

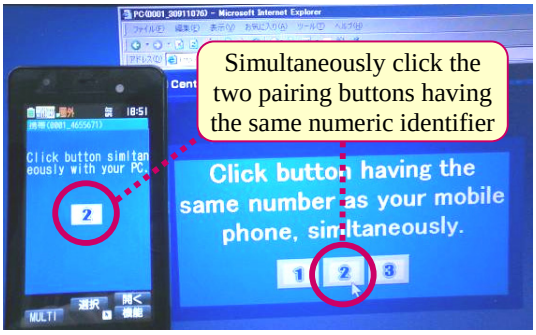

Figure 3: Snapshot of our prototype system.

4. Evaluation

4.1. DPS Scalability

We evaluated the scalability of the DPS by analyzing the relationship between S_{opt} and the maximum number of device pairings per day n. We assumed that λ did not temporally vary and that all users performed ten device pairings per 24 hours on average. We also assumed that $B_{\max}$ was set to 2.0% and h to 1.0 s as server-side settings. $B_{\max}$ = 2.0% means that a user will encounter one collision in 50 device pairings on average. As described in Section 3.1, even if a collision occurs, a desired pair could be created. Moreover, even if a mismatched pair is created as a result of the collision, the user can manually correct the mismatched pair to the desired one. Accordingly, we believe the value of $B_{\max}$ is reasonable. We restricted the maximum value of S_{opt} to 9 on the basis of Miller's experimental results for human short term memory [11].

The analytic results obtained using the relationship $\lambda_{next} = n \cdot h/(24 \times 3{,}600)$ and Equation (2) are shown in Figure 4.

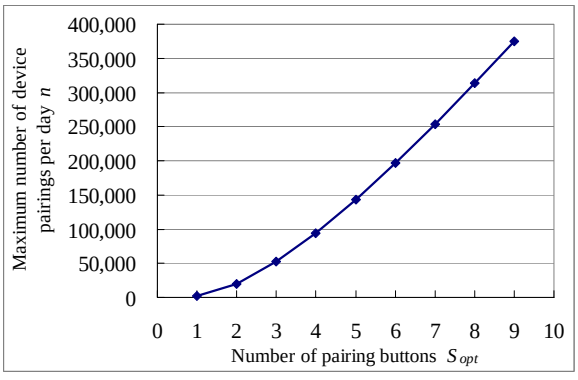

Figure 4: DPS scalability.

For S_{opt} = 9, the DPS can processes a maximum of about 375,000 device pairings per day; that is, about 37,500 users can be supported. Hence, we confirmed that the multi-channel-based device pairing control has sufficient scalability to handle organizations with several ten thousand users. Since the basic method was reported to be able to processes a maximum of about 1700 requests per day [2]——this numerical value is the same as with our method for S_{opt} = 1, we confirmed that our method's scalability is approximately 220 times that of the basic method.

4.2. Proactive Behavior of Adaptive Channel Fluctuation Control

We validated the proactive behavior of the adaptive channel fluctuation control described in Section 3.5. We analyzed the relationship among λ, S_{opt}, and B_S with a simulation. We increased λ from 0 time per day to 375,000 times per day. Note that λ is dynamically changed while λ was static in Section 4.1. We assumed that $B_{\max}$ was set to 2.0% as a server-side setting. The simulated results are shown in Figure 5. We confirmed that even if B_S rises as a result of an increase in λ, B_S is kept below $B_{\max}$ by proactively adding 1 to S_{opt} before B_S exceeds $B_{\max}$.

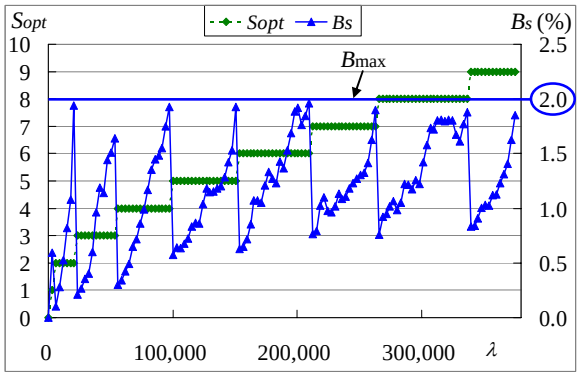

Figure 5: Proactive behavior of adaptive channel fluctuation control.

5. Discussion

In this section, we discuss several issues pertaining to our method and their solutions.

5.1. Network Delay and Jitter

Since large-scale organizations usually own a lot of interconnecting LANs, pairing request packets are exchanged via multiple routers between the DPS and client devices. Furthermore, the route between the DPS and a mobile phone and the route between the DPS and a PC are different in the case that the mobile phone connects to a wireless network while the PC connects to a wired network. Consequently, pairing request packets suffers the influence of diverse network delay and jitter.

Since a pair of devices is created on the basis of comparing the reception timestamps of two paring requests at the DPS, if the network delay and/or jitter

become sufficiently larger than the acceptance period, pair creation might become difficult.

This problem can be solved by introducing clock synchronization between the DPS and client device to absorb the influence of network delay and jitter. By measuring the time difference between the client device and itself, the DPS can calculate when the button was clicked at the client device by using the measured time difference as an offset applied to the request reception timestamp. Hence, a device pair can be created by comparing not reception timestamps, but click timestamps.

This solution can be implemented using an HTTP-based time synchronization protocol, e.g., SNTP [12] over HTTP or HTP [13]. While the client device is opening a pairing Web page, the DPS can measure the time difference between the client device and itself by using one of these protocols.

5.2. Protection from Attacks

To make device pairing secure, the device pairing system must be protected against attacks such as distributed denial-of-service (DDoS) attacks and man-in-the-middle (MitM) attacks. If attackers continually send massive numbers of requests from a large number of mobile phones and PCs, the collision probability could be extremely large, resulting in a successful DDoS attack. However, existing countermeasures for DDoS attacks, e.g., [14][15], can solve this problem.

In our method, MitM attacks can be prevented by simply using HTTPS [16] as a network communication protocol between the DPS (Web application server) and client device Web browser. Since the HTTPS server certificate issued by a trusted certificate authority can guarantee that the DPS is the genuine server, an MitM attacker's server cannot pretend to be the DPS. Thus, the MitM attacker's server cannot relay any device pairing packets between the DPS and a client device.

6. Conclusion and Future Work

We proposed a device pairing method which can keep the collision probability of pairing requests below a permissible limit by using multiple pairing buttons. Through an evaluation, we confirmed that this method has approximately 220 times the scalability of a conventional method and that it has sufficient scalability for deployment in large-scale organizations. The protocol used in our method is HTTP; therefore, arbitrary devices with only a Web browser can be used in an out-of-box manner and our system is independent of the type of network, such as wireless or wired. Hence, our system can be deployed at low cost in large-scale organizations.

We plan to prove the practicability of our method through a field trial in a real corporate environment. We also plan to improve the scalability further toward massive scale consumer services.

References

[1] Unisys News Release, "Forty-nine percent of employees now use multiple devices – primarily mobile ones – over the course of a typical workday," August 28, 2007, *http://www.unisys.com/about__unisys/news_a_events/08288810.htm*

[2] T. Oshiba et al., "Ad-hoc Endpoint Pairing based on Simultaneous Clicking for Ubiquitous Communications," *Proc. of SAINT 2008*, pp. 467–470, 2008.

[3] R. Marin-Perianu et al., "Movement-based Group Awareness with Wireless Sensor Networks," *Proc. of Pervasive 2007*, pp. 298–315, 2007.

[4] R. Mayrhofer et al., "Shake Well Before Use: Authentication based on Accelerometer Data," *Proc. of Pervasive 2007*, pp. 144–161, 2007.

[5] C. Soriente et al., "BEDA: Button-Enabled Device Association," *Proc. of IWSSI 2007*, 2007.

[6] J. Rekimoto, "SyncTap: Synchronous User Operation for Spontaneous Network Connection," *Personal and Ubiquitous Computing*, Vol. 8, Issue 2, pp. 126–134, 2004.

[7] J.M. McCune et al., "Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication," *Proc. of 2005 IEEE Symposium on Security and Privacy*, pp. 110–124, 2005.

[8] E.S. Gardner Jr., "Exponential Smoothing: The State of the Art," *Journal of Forecasting*, Vol. 4, Issue 1, pp. 1–28, 1985.

[9] A.K. Erlang, "Solution of Some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges," *Transactions of the Danish Academy of Technical Sciences*, No. 2, pp. 138–155, 1948.

[10] N906iL website, *http://www.nttdocomo.co.jp/english/product/foma/906i/n906il/index.html*

[11] G.A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Review*, Vol. 63, pp. 81–97, 1956.

[12] D. Mills, "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," *IETF RFC 2030*, 1996.

[13] E. Vervest et al., "HTTP Time Protocol (HTP)," open source software available at *http://www.clevervest.com/htp/*

[14] F. Kargl et al., "Protecting Web Servers from Distributed Denial of Service Attacks," *Proc. of WWW'01*, pp. 514–524, 2001.

[15] J. Mirkovic et al., "A Taxonomy of DDoS Attacks and Defense Mechanisms," *ACM SIGCOMM Computer Communications Review*, Vol. 34, No. 2, pp. 39–54, 2004.

[16] E. Rescorla, "HTTP over TLS," *IETF RFC 2818*, 2000.