The *Zephyr* Notification Service

C. Anthony DellaFera

Digital Equipment Corporation
Project Athena
Massachusetts Institute of Technology
Cambridge, MA 02139
tony@ATHENA.MIT.EDU

Mark W. Eichin
Robert S. French
David C. Jedlinsky
John T. Kohl
William E. Sommerfeld

Project Athena
Massachusetts Institute of Technology
Cambridge, MA 02139
{eichin,rfrench,opus,jtkohl,wesommer}@ATHENA.MIT.EDU

ABSTRACT

Zephyr is a notice transport and delivery system under development at Project Athena.[1] *Zephyr* is for use by network-based services and applications with a need for immediate, reliable and rapid communication with their clients. *Zephyr* meets the high-throughput, high fan-out communications requirements of large-scale workstation environments. It is designed as a suite of ''layered services'' based on a reliable, authenticated notice protocol. Multiple, redundant *Zephyr* servers provide basic routing, queueing, and dispatching services to clients that communicate via the *Zephyr* Client Library. More advanced communication services are built upon this base.

Introduction

This paper is a brief introduction to the concept of a notification service in general and to the design of the *Zephyr* Notification Service in particular. A notification service provides network-based services and their clients with immediate, reliable, and rapid communication for small quantities of time-sensitive information. The sections which follow address the following issues:

- Motivation for developing a notification service.
- Role of a notification service in networked workstation environments.
- Design constraints.
- Services provided for users by a notification service.
- Unsolved problems and topics for future development.

While this paper is about *Zephyr*, Project Athena's Notification Service, it is our belief that the concepts presented here can be generalized to fit a broad range of notification services and systems. A good understanding of a notification service can be acquired by comparing the *Zephyr* Notification Service and a more traditional method of workstation message delivery, electronic mail (specifically, *sendmail*). Table 1 makes this comparison.

Table 1.

A Comparison Between *Zephyr* And Mail		
Metric	*Zephyr*	Electronic Mail
Addressing	Implicit/Dynamic: All addressing is determined dynamically; an explicit ‘‘address’’ is not required. One-to-one addressing is supported by explicitly specifying of recipient ZID. The notice subscription layered service allows for self-selection by the recipient.	Explicit/Static: Sender must know and explicitly provide the name and address (except for ‘‘local’’ mail) of each recipient. Static mailing list support is provided. Only recipients explicitly named receive messages.
Delivery Method	Notices are delivered via dynamically routed reliable datagrams. No connections need be established or maintained. Multiple levels of notice acknowledgment are supported.	Mail is delivered using a point-to-point TCP/IP connection. Acknowledgments are not supported, per se. Return receipts may be requested but are not guaranteed to work.
Delivery Action	Asynchronous/Active: Notices arrive without user intervention. Notices to a particular user are delivered automatically and immediately to all of his or her current login sessions.	Synchronous/Passive: Mail must be sent and read manually by the user. Mail, in general, is delivered to one particular place (a ‘‘post office’’ or ‘‘mail drop’’) for each user; s/he must then actively retrieve it.
Message Length	Short, fixed, notice length with a maximum of approximately 10 lines of 80 characters each.	Long, typically unfixed, message length. Mail messages may be and often are extremely large, on the order of many pages. When message length is fixed it is usually done so by unpredictable rules that vary from site to site.
Message Persistence	Notices are considered time-sensitive; no queuing is provided. If the client recipient is not logged in, then the notice is flushed.	Long time-to-live. Messages typically remain in a mail drop until read.
Message Fan-out	High fan-out: Sending to large lists is efficient. Each client generates one copy of a notice regardless of the number of recipients. Each server receives one copy of a notice being routed regardless of the number of recipients.	Low fan-out. Sending to large lists can consume a great deal of the computing resource. If a message is sent to *n* users, *n* copies are generated by the sender, each of which is retained indefinitely by its recipient.
Traffic Performance	High volume/High throughput: Notices may be transmitted in large numbers due to the low overhead of dynamically-routed reliable datagrams.	Medium volume/Low throughput: Large mail messages can send a reasonable volume of data, but slowly (as connections need to be established and routes determined).

A Comparison Between *Zephyr* And Mail		
Metric	*Zephyr*	Electronic Mail
System Configurability	Dynamically reconfigurable: Dynamic resource allocation and configuration within the base notification services allows for automatic and simple user-level reconfiguration of layered services.	Statically reconfigurable: Reconfiguration of Unix mail systems is wizard-level work and has significant global impact. No utilities are provided for dynamic system modification or reconfiguration. All changes must be made centrally and atomically.
System Maintenance	Low maintenance: Layered services dynamically recover unused resources through time-outs and reference counting.	High maintenance: Mail requires a post office staff to maintain post office boxes (mail drops), mailing lists, the routing system, and to manually reroute ‘‘dead letters.’’

A Solution To Communication Needs

When services designed for use in a time-sharing environment are used for a very large system of networked workstations, certain communication services begin to fail.† They predominantly fail from to their inability to cope with large increases in network scale (i.e., an increase in both the number of workstations and the number of interconnected local area networks). After examining how certain of these services communicate with their clients, we have identified two primary failure modes. These are the inability of a service to cope with increasing server-to-client fanout and the inability of clients to deal by the replacement of a local service with a remote, distributed service. The *Zephyr* project was begun to provide a solution to these two failure modes in the context of time-sensitive communications. *Zephyr* has grown into a more powerful tool than was originally anticipated; what began as the development of a ‘‘desirable’’ service soon turned into the development of a ‘‘required’’ service.

The following services are candidate clients of a notification service. All need either the base level *Zephyr* service, which will deliver a message to an identifiable but unlocalized recipient, or the notice subscription layered service, which will deliver a message to the set of potential recipients interested in, i.e. subscribing to, messages of that class. These service levels are discussed more fully in the next section.

File Service
: A file server can send notices to the users and hosts that it knows would be affected by a change in file server status, e.g. a shutdown. If those users also register a subscription with the notice subscription layered service, other providers of information, such as operations staff, would also be able to reach the user.

Post Office Service
: Remote post offices can notify users about the arrival of new mail.

Electronic Meeting Service
: Electronic meeting services (conferencing systems) can notify interested users of new transactions, using the notice subscription layered service.

Print Service
: Print servers (and queuing services in general) can send job status information back to the job’s submitter.

MOTD Service
: Message-of-the-day information (system wide, service-specific, or local) can be sent to users, such as when they begin using a particular service in the case of a service-specific MOTD.

On-Line Consulting
: The notification service can be used as the underpinning of a dynamic on-line consulting service. The notice subscription layered service can be used to provide topic based information routing, user location, and consultant-to-client rendezvous.

Host Status Service
: Broadcast-based host status systems (e.g., *ruptime*) do not scale to a large workstation environment; disk usage grows linearly with network size and total packet computation time grows geometrically. The notification service can provide immediate host status and error log notification on selected hosts or servers.

User Location Service
: Broadcast-based user location systems (e.g., *rwho*) also do not scale to a large workstation environment for the same reasons noted above under *Host Status Service*. The notification service can provide asynchronous and immediate user location and state change (login/logout) notification on selected users or groups of users. This can facilitate communication among users. For example, within the limits of user permission and access control (described in the section *A User’s Overview*), students can watch for their friends or development team members for their co-workers.

Talk or Phone Service
: In a network of workstations, one must know the exact address of others in order to *talk* to them. A notification-based *talk* facility can be constructed that locates the party being called, transmits a talk request notice to that party and, if permission is granted, automatically establishes a *talk* connection.

Emergency Notification
: In the Athena environment, there is a requirement to provide a simple, asynchronous, and

†Actually, services in general begin to fail, but the scope of this discussion is primarily the realm of communication services. While some of the services, such as *On-Line Consulting*, may be unfamiliar to the reader, they are part of the Athena environment. See the companion paper on Athena Changes to Berkeley Unix, this volume, for an overview of that environment.[2]

secure means of sending urgent notices to all users on a workstation or in a particular group of workstations. Broadcast methods are not useful on a large scale and are by definition imprecise. In addition, the workstation user must trust the broadcasting host and the person issuing the message. Using *Zephyr*, emergency notices can be sent directly to all users on any specified host, with authenticity† guaranteed.

Message Service

Current *write* services suffer the same problems noted above under *Talk or Phone Services*. A notification-based *write* utility is trivial, since almost all the work is subsumed by the notification service. Write notices can go to individual users or to previously created subscription groups.

Other Service Events

The notification service can be used to reliably notify users of a wide range of asynchronous service events that occur in distributed workstation environments. This notification can be accomplished by using the base notification service, the emergency notification service, or one of the notification service layered services.

Fitting The Tool To The Job

Zephyr is designed around a system of dynamically-updated, locally-authoritative servers that provide centralized routing, queuing, and dispatching. Clients communicate with these servers via the *Zephyr* Client Library interface. The *Zephyr* Client Library implements the *Zephyr* Protocol, a reliable, authenticated notice transmission protocol. In our design we view the notification service as a suite of ‘‘layered services’’ built upon a base notice transport layer. Additional layers provide more advanced communications services. This design is analogous to that of the X Window System.[4]

Zephyr notices consist of two parts: a routing header and client data. It is the job of the *Zephyr* Servers to route notices between *Zephyr* clients based upon attributes specified in the notice's routing header. Servers do not examine a notice's client data; it is entirely possible that that data is encrypted or otherwise uninterpretable. By examining the attributes in a notice's routing header, a *Zephyr* Server computes the list of recipients of a notice. The most basic routing attribute that may be specified is a single recipient, named by a ZID.† More complex routing attributes are specified for the layered services. The notice classification information for the notice subscription layered service discussed above or specialized keywords for use with a rule-based routing service are examples of such complex attributes.

Determining notice recipients based upon routing header attributes is known as "subscription multicasting". Subscription multicasting is a passive routing technique; attributes not recognized by a service layer are simply ignored. This allows layered services to implement different notice routing methods that peacefully co-exist while using the same base notification service. In this way subscription multicasting differs from other message routing techniques such as network broadcast or *sendmail*. Because the set of recipients for any notice can always be determined, it is more efficient and less vulnerable to increases in network scale than broadcast techniques. Because additional resources, routing methods, and recipients may be dynamically added by almost any user, it requires less maintenance and incurs less overhead than traditional list based message transmission techniques (such as *sendmail*).

Zephyr clients determine what level of service they receive from the notification service by choosing the service layer with which they communicate. For example, certain system services have complete knowledge of their clients, and only need the notification service to route information to those clients. This is the most basic service layer. For example, a file server knows the particular users it is serving, and needs only the ability to reliably notify those users about service state changes. On the other hand, client services that cannot identify their clients (or may simply not know who is interested in such state information) may wish to notify ‘‘interested parties’’ about service state changes. A workstation error logger would fall into this category. This type of service would make use of the notice subscription service layer. Such a service layer provides the ability to store communication state information for client services external to those

---

†Since *Zephyr* relies on authentication, it is also suggested that you read the *Kerberos* paper in this volume.[3] This provides a general introduction the Project Athena *Kerberos* Authentication System.

†Recipients must be uniquely identifiable. *Zephyr* relies on *Kerberos* to both provide and guarantee these identifiers. So as to avoid confusion with the sense of a UID, we shall refer to the *Zephyr* identifier as a ZID.

services. This adds to the notification service the unique ability to provide to its clients status and availability information about other services even when those services are disabled and cannot communicate with their own clients.

Design Requirements And Constraints

The goal of the *Zephyr* development is to produce a 4.3BSD Unix implementation of a notification service useful to Athena. This implementation should consider the effects of:

Scale - Such a service must efficiently provide its capabilities with the highest possible fan-out (i.e., client to server ratio), without adversely affecting network load or server host performance. Additional redundant servers must be easy to install, and must provide load sharing immediately.

Evolution - Since *Zephyr* is an evolving service, it is must gracefully handle protocol compatibility from one version of the service to the next.

Management - It must be possible to perform all aspects of service maintenance and operation remotely.

Network Protocols - Since the notification service depends upon an underlying network transport mechanism, it accepts those design constraints imposed by that mechanism. The *Zephyr* Protocol, as currently implemented, is based on the Unreliable Datagram Protocol (UDP). As such, it is constrained to operate within the capabilities of UDP. However, there is nothing in the design of the protocol that would prevent its using other network transport mechanisms (such as a remote procedure call system). Constraints imposed by UDP are listed below, along with a brief description of how they affect *Zephyr* application programmers and end users.

Duplicate Notices
UDP does not provide any suppression of duplicate packets, *Zephyr* clients may receive duplicate *Zephyr* notices. *Zephyr* applications must be capable of dealing with this possibility.

Missequenced Notices
UDP does not provide packet sequencing. While *Zephyr* notices do contain timestamps, it is up to the application to check the timestamp and handle notices received out of sequence.

Flow Control
UDP does not provide any flow-control capability. *Zephyr* applications must be capable of dealing with notices at whatever rate they arrive or be willing to lose notices.

Unreliable Delivery
UDP does not provide a reliable delivery mechanism. *Zephyr* does provide several levels of acknowledgment processing, but it is up to the application to decide how much overhead it is willing to incur in order to guarantee notice delivery.

Packet Size
UDP packets have a relatively small, maximum size. Considering the amount of routing data and other information that *Zephyr* must store in each packet, this becomes a constraint on how much user data may be included with each packet.

How visible these constraints are to the end user is up to the *Zephyr* application programmer. For example, our *zwrite* application (which allows users to exchange *write*-like messages) only guarantees that the message was sent, not that it will actually arrive or how many copies will arrive.

In order to provide ZID-based communications, the notification service must dynamically maintain a database that maps ZID's to their current physical locations in the network. The only requirement on the ZID used in addressing is that it must be a network wide ZID and be ''registered'' as a client of the notification service (i.e., have its physical address(es) stored in the notification service database). In particular, *Zephyr* manages a location database that maps *Kerberos* ''principal names'' (authenticated user names) into a tuple of physical location information (geographic location, hostname, IP port number, and tty, among other things). This database is primarily used by *Zephyr* for notice routing, but is also made available to *Zephyr* clients via the User Location Layered Service discussed below.

The reliability of the information stored in the user location database imposes constraints upon applications that rely on *Zephyr*.

- User location information present in the database can be assumed to indicate that a user has logged in, because user logins that are reported to *Zephyr* must be *Kerberos* authenticated.
- User location information present in the database cannot be assumed to indicate that a user is still logged in, because there is no way to guarantee an orderly user logout.

(For example, a workstation may crash).

- The absence of user location information in the database cannot be assumed to indicate that a user has not logged in, because a user can choose to not make his or her login information publicly available.

Zephyr attempts to prevent user location data from persisting when it is no longer valid. If a workstation crashes, the user login sessions on that workstation are necessarily terminated without sending logout notices to a *Zephyr* server. This implies that logins in the database may not always be valid. To cope with this, a specialized *Zephyr* client runs during the workstation reboot sequence. This client simply tells a *Zephyr* server to flush any previous state information associated with the (rebooted) workstation.

A User's Overview

For this discussion, a ‘‘user’’ of *Zephyr* is either a user of a *Zephyr* client or an applications programmer who is using the *Zephyr* Client Library.

The *Zephyr* system can be viewed as divided into two parts, clients and servers. There must be at least one *Zephyr* Server (*zephyrd*) per *Kerberos* realm (realm of authority of a particular authentication service), one *Zephyr* HostManager Client (*zhm*) per active workstation and one *Zephyr* WindowGram Client (*zwgc*) per user login session. To ensure reliable service, there should be several *Zephyr* Servers per *Kerberos* realm.

When a workstation is reboots, a *zhm* is automatically started. The *zhm* serves two purposes. First, it acts as a reliable transmission tower for notices sent from local *Zephyr* clients. Second, the *zhm* acts as an emergency notice routing channel on the individual workstation. When *zhm* starts up it first seeks out a *Zephyr* Server and registers itself with that server. From then on, that *zhm* and all clients that communicate through it are ‘‘owned’’ by that server. Only that server will be considered to have ‘‘authoritative’’ information about *Zephyr* clients on the workstation managed by that *zhm*.

All *Zephyr* clients on a workstation use the *Zephyr* Client Library to send and receive *Zephyr* notices. The *Zephyr* Client Library routes all notice traffic leaving a workstation through that workstation's *zhm*. In this way, clients themselves are not required to have to locate and manage communications with a *Zephyr* Server. If the *zhm* loses contact with its *Zephyr* Server (i.e., the *Zephyr* server which owns it does not respond within a fixed but configurable safety margin) it is the *zhm's* job to seek out and contact a new *Zephyr* Server.

When a user logs into a workstation, a *zwgc* for that user is automatically started, provided that the user can provide an authenticator and that the user has not deliberately disabled *Zephyr*. If the user is not interested in using *Zephyr*, it is still important that s/he have a *zwgc* running. The most important reason is that *zwgc* is the contact point for *Zephyr* emergency notices. These notices are transmitted by certain privileged users (e.g. operations staff), *directly* to the workstation's *zhm*. The *zhm* then forwards these notices to all *zwgc's* currently running on the workstation.

When the *zwgc* starts up, it registers the user with *Zephyr* and, depending upon the setting of certain *Zephyr* control variables, may make other *Zephyr* requests. These variables may be modified using the *Zephyr* Control Utility (*zctl*). The most important of these variables is the *Zephyr* exposure level variable. This variable controls how much information about an individual user *Zephyr* will store and make available to requesting clients. There are currently six possible settings for this variable:

none - This completely disables *Zephyr*. The user is not registered with *Zephyr*. No user location information is retained by *Zephyr*. No login or logout announcements will be sent. No system default notice subscriptions will be entered for the user.

opstaff - The user is registered with *Zephyr*. Only system operation notices and emergency notices will be received. No user location information is retained by *Zephyr*. No login or logout announcements will be sent. System default notice subscriptions will be entered for the user.

realm-visible - This is the default exposure setting. The user is registered with *Zephyr*. All notices will be received. User location information is retained by *Zephyr* and made available only to users within the *Kerberos* realm. No login or logout announcements will be sent. System default notice subscriptions will be entered for the user.

realm-announced - The same as **realm-visible**, plus login/out announcements will be sent to users within the *Kerberos* realm who have explicitly requested them).

net-visible - The same as **realm-visible**, plus user location information is made available to any user who requests it.

net-announced - The same as **realm-announced**, plus login/out announcements will be sent to any user who has requested them.

zwgc is a vital client. For this reason *zwgc* has two primary interfaces. The first, and most powerful, is an X Window System interface referred to as a ''WindowGram browser.'' This browser allows the user to scroll through and perform certain operations (such as ''save'', ''delete'', ''cut'' and ''paste'') on all the notices that *zwgc* has received. If a user logs into an X display, *xwgc* selects this interface. If the user does not have access to an X Display, *xwgc* selects a simple terminal based interface.

In addition to the basic services described above, *Zephyr* provides additional layered services that are built on the base notification service.† The two most important of these are the *Zephyr* Notice Subscription Layered Service and the *Zephyr* User Location Layered Service.

The *Zephyr* Notice Subscription Layered Service provides a dynamic information dispersal service based upon a ''subscription list'' paradigm. The most important use for this layered service is by services that cannot directly identify their clients or may simply not know who is interested in the information they are providing. Such services may wish to simply send notices to ''all interested parties.''

The simplest function that the *Zephyr* Notice Subscription Layered Service provides is a method for users and groups of users to exchange notices. This is accomplished with the *zwrite* utility. In addition to person-to-person messages *zwrite* allows users to send notices to notice subscription lists.

Restricted access to user location information is made available to *Zephyr* clients by the *Zephyr* Client Library. This information is dispersed by the User Location Layered Service. The database which this layered service uses is maintained internally by *Zephyr* to track the existence of ZID's in the network. A user can be located through *Zephyr* by using the *zlocate* and/or *znol* utilities. These utilities make calls to the *Zephyr* User Location Layered Service for the user. The *zlocate* utility allows users to manually locate one or more users. The *znol* utility makes use of the *Zephyr* Notice Subscription Layered Service to subscribe to user login/out notices from a list of ZID's provided by the user.

†The architectural design details of the *Zephyr* Service Layers are discussed in the *Zephyr* design document.[5]

When a workstation crashes, all clients running on that workstation are lost. When this happensr, client state information that *Zephyr* has associated with that workstation must be flushed and any corresponding *Zephyr* system resources must be freed. This is done in two ways. The first occurs slowly while the workstation remains down, the second when it reboots. If the workstation remain down long enough all invalid state information will be incrementally detected and flushed. This incremental flushing occurs whenever a *Zephyr* Server attempts to send (or route) a notice to a client and discovers that the workstation on which that client is supposed to be running is not responding. When the workstation does eventually reboot, *zhm* is called with a special ''reboot flush'' flag. This causes *zhm* to run just long enough to transmit a special workstation state flush notice to the first available *Zephyr* Server. These two ''garbage collection'' techniques work together to keep the *Zephyr* system's database current.

The last phase of user interaction with *Zephyr* occurs at logout time. When the user logs out, *zwgc* notifies *Zephyr* to flush all state associated with that user's login and then exits.

The *Zephyr* system currently consists of the following suite of programs:

zctl(1)	*Zephyr* subscription control program
zinit(1)	*Zephyr* login initialization program
zleave(1)	Remind you when you have to leave
zlocate(1)	Find a user
znol(1)	Notify on login/out of specified users
zwgc(1)	*Zephyr* WindowGram Client
zwrite(1)	Write to another user
zhm(8)	*Zephyr* HostManager Client
zephyrd(8)	*Zephyr* Server daemon
zstat(8)	Display *Zephyr* statistics

Future Directions And Unsolved Problems

Once the basic notification service is in place it becomes a simple matter to provide many other layered services based upon it. The *talk* service mentioned above is a good example of a service that utilizes multiple *Zephyr* service layers. We envision *Zephyr* as a transport service that can incorporate new notice routing methods as they are developed. *Zephyr* is designed to allow communication development efforts to occur side-by-side with running production systems that utilize *Zephyr*. For example, researchers at M.I.T.'s Sloan School of Management have expressed interest in using *Zephyr* as the transport service layer for a rule based communication system.

The following is a list of some of the areas of future development. They are either unsolved problems or areas that need further investigation.

- Extend the *Zephyr* Protocol for use across long-haul networks.
- Modify the *Zephyr* Server to allow ZID registration from other *Kerberos* authentication realms.
- Modify the *Zephyr* Server to forward *Zephyr* notices to recipients in other *Kerberos* realms.
- Develop a more formal interface definition for use between the *Zephyr* notice transport layer and *Zephyr* Layered Services.
- Develop a more advanced user interface for the *Zephyr* WindowGram Client.
- Decouple *Zephyr* from the *Kerberos* system. In the current implementation, they are linked together too closely.
- Integrate with more clients.

Conclusions

Zephyr has proven useful in providing a mechanism for transporting time-sensitive information in a large-scale workstation environment. It not only permits existing services from the timesharing world to evolve toward the workstation world, but also permits new services to grow alongside. It makes reasonable compromises between reliability and complexity and is already of use to both users and operations staff. Indeed a major problem has been its popularity while still under development.

Acknowledgements

The authors would like to acknowledge the following people from M.I.T. Project Athena for their help in making *Zephyr* a reality. Michael R. Gretzinger, former Systems Programmer, and David G. Grubbs, former Manager of Systems Integration, for their suggestions on the initial concept of a Notification Service, and Daniel E. Geer, the Manager of Systems Development, for his undying support of our efforts. We also thank Katharyn L. Lieben and G. Winfield Treese for the improvements they made to this paper.

References

1. E. Balkovich, S. R. Lerman, and R. P. Parmelee, ''Computing in Higher Education: The Athena Experience,'' *Communications of the ACM* **28**(11), pp. 1214-1224, ACM (November, 1985).

2. G. W. Treese, ''Berkeley Unix on 1000 Workstations: Athena Changes to 4.3BSD,'' pp. 175-182 in *Usenix Conference Proceedings*, Dallas, Texas (February, 1988).

3. J. G. Steiner, B. C. Neuman, and J. I. Schiller, ''Kerberos: An Authentication Service for Open Network Systems,'' pp. 191-202 in *Usenix Conference Proceedings*, Dallas, Texas (February, 1988).

4. R. W. Scheifler and J. Gettys, ''The X Window System,'' *ACM Transactions On Graphics* **5**(2), pp. 79-109 (April, 1987).

5. C. A. DellaFera, M. W. Eichin, R. S. French, D. C. Jedlinsky, J. T. Kohl, and W. E. Sommerfeld, *Section E.4.1: Zephyr Notification Service*, M.I.T. Project Athena, Cambridge, Massachusetts (December 21, 1987).