UNIVERSITA' DEGLI STUDI DI PISA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

COMPUTING ASSETS CATEGORIZATION ACCORDING TO COLLECTED CONFIGURATION AND USAGE INFORMATION

Candidate

Milco Filoni

Supervisors :

Prof. Maurizio Bonuccelli , *University of Pisa*

Dott. Luca Deri, *Netikos S.p.a.*

Dieter Gantenbein, *IBM Zurich Research Laboratory*

Reviewer :

Prof. Dino Pedreschi, *University of Pisa*

Academic Year 2000/01

Abstract

Building an **accurate inventory of computing assets** in an unknown system and networking environment is a challenging task. Accounting records, physical inventories, computing service configurations, traffic monitors, network and applications server logs all represent valid sources of information. Their integration into a consistent model of the reality is a desirable goal, e.g. as basis for building commercial Information Technology (IT) Inventory systems, or as background information for Intrusion Detection Systems (IDS).

During this project, we will **study the optimal integration of relevant information sources,** aiming to generate an enhanced inventory of deployed computing assets categorized according to their usage. In particular, we will investigate the use of log information from HTTP and SMTP/POP servers, Socks/Proxy gateways, and network monitors, and then define a knowledge discovery in database (KDD) process to prepare data for subsequent data-mining tasks aiming to generate an enhanced inventory of the deployed computing assets and categorize them according to their usage.

The data integration and mining will be prototyped and validated in the context of the IBM Intelligent Device Discovery (IDD) project and ongoing activities related to network management and intrusion detection at the University of Pisa.

ACKNOWLEDGEMENTS

I wish to acknowledge all of those inside and outside of the IBM ZRL who contributed their time, effort, and expertise to help produce this thesis. In particular I want to thank Dieter Gantenbein for his valuable and detailed critiques, assistance, and constant support throughout the development of this publication, Luca Deri who provided several ideas for discussions, and Prof. Maurizio Bonuccelli for his help in accessing the data we used for the validation. Thanks also to Chris Giblin and Matthieu Verbert who helped developing the software and to my manager Lucas Heusler for giving me the possibility to join a so agreeable environment.

Since this document will complete my university studies I want also to thank my family, in particular my parents but also Pierangelo and Melina, for the provided non technical support. Special thanks go to all the friends of the "Three Cripples' Tavern" with whom I have spent most of my free time in the last years (sorry but I cannot list your names, you are too many!). Final thanks to Christian, Stefano, and Patrizio for all the cheerful moments spent together.

RINGRAZIAMENTI

Voglio innanzitutto ringraziare tutti coloro che fuori e dentro lo ZRL hanno speso parte del loro tempo per aiutarmi a portare a termine questa tesi di laurea. In particolare voglio ringraziare Dieter Gantenbein per le sue preziose e costruttive critiche e per il supporto datomi durante tutto lo svolgimento del lavoro, Luca Deri per le diverse idee proposte ed il Prof. Maurizio Bonuccelli per averci fornito i dati che ci hanno permesso di validare l'intero processo descritto in questo documento. Un grazie anche a Chris Giblin e a Matthieu Verbert per l'aiuto offertomi nello sviluppo del software.

Poiche' questo documento conclude i miei studi universitari, voglio anche ringraziare la mia famiglia, i miei genitori in particolare ma anche Pierangelo e Melina, per la loro costante presenza e per il supporto morale fornitomi in tutti questi anni. Ringrazio poi tutti gli amici de "La Taverna Dei Tre Storpi" con i quali ho passato la maggior parte del mio tempo libero negli ultimi anni (scusatemi se non vi elenco ma siete troppi). Per finire un grazie speciale a Christian, Stefano e Patrizio per tutti i momenti allegri passati insieme.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

In today's dynamic information society, organizations critically depend on the computing infrastructure. Tracking computing devices as assets and their usage helps in the provision and maintenance of an efficient, optimized service. A precise understanding of the operational infrastructure and its users also plays a key role during the negotiation of outsourcing contracts and for planning mergers and acquisitions. Building an accurate inventory of computing assets is especially difficult in unknown heterogeneous systems and networking environments without prior device instrumentation. User mobility and mobile, not-always-signed-on, computing devices add to the challenge.

1.1 Computing Infrastructure as a Critical Business Resource

Modern e-business environments tightly link the customer and supplier systems with the internal computing infrastructure. Hence the performance of the end-to-end business processes becomes critically dependent on the availability of the underlying computing infrastructure. From an economic perspective, the efficient cost-effective and resource-optimized provision of the required services is another argument in many organizations to justify the tight grip on the computing assets deployed and their usage [34].

Classical methods for inventory and asset management quickly reach their limit in today's dynamic environments: Periodic physical inventories ("wall-to-wall") have the clear advantage of identifying the actual location of the devices but require costly human visits ("sneaker net") and can detect neither mobile, currently out-

of-office equipment nor the existence and use of logical assets. Financial asset tracking, while being an accepted process in its own right, cannot detect additional equipment brought into or accessing the resources of an organization. Periodic self-assessment questionnaires to be filled out by individual end users or their cost-center managers are another and often complementary approach. Apart from the human effort they require and the inaccurate and incomplete data that results, most forms pose questions the answer of which could easily be read out of the infrastructure itself.

Well-managed computing infrastructures typically equip servers and end-user devices with software daemons for the tracking of resources and the system as well as for application performance monitoring [59], [60]. There are many situations, however, in which this cannot be assumed and used. In many organizations, there are a fair number of devices that are brought in ad-hoc and are not instrumented accordingly, for which instrumentation is not available, or on which instrumentation has been disabled. After a merger/acquisition, for example, we can hardly assume to find an encompassing management environment in place across the entire holding organization. However, a good understanding of the provided infrastructure and its users is essential, actually already prior to the acquisition or during the negotiation of an outsourcing contract. Such investigations to gather the data, required to allow more accurate service cost predictions and to reduce the risk of unforeseen contractual responsibilities, are often called Due Diligence or Joint Verification.

1.2 Proposal

In this work, we argue that it is no longer sufficient to keep a static inventory of information technology (IT) assets, but that the online tracking and categorization of resource usage based on found communication patterns provide a much richer information base, and enable faster and more accurate decisions in today's

evolving e-business environment. We propose therefore to complement basic network-based discovery techniques (described in detail in Chapter 2) with the combined log information from network and application servers to compute a aggregate picture of assets, and to categorize their usage.

The following statements summarize the benefits we expect to have with respect to both classical methods and network-based techniques for assets discovery and tracking:

- The main benefit is that log files provide historical information. They not only complement asset-tracking tools in building an accurate asset inventory but also provide information on how such entities interact during network usage. We argue that in order to better manage assets a good understanding of their usage is needed. In other words we believe that two hosts with exactly the same configuration but used in a different way also need to be managed differently.

- No daemon for collecting data has to be installed because each application server has its own logging mechanism. Most of the time log files already exist, ready to be used. In some cases the server needs to be configured explicitly configured but this is quite straightforward compared to installing new software that may also be system dependent.

- It is a totally passive approach. This means that no traffic has to be generated in order to collect data. This is another advantage that our approach has over most of the network-based discovery techniques.

- Application server log files can provide protocol-specific information that cannot be collected in lower layers of the network stack such as for example information about users.

Unfortunately data stored in log files are difficult to manage because they are completely flat and uncorrelated. A common integrated repository with a unified scheme in which such raw and uncorrelated data can be stored is needed. However this is only one face of the medal. We also need as flexible an approach as possible that allows data from new logs to be added easily and in a simple way. We will show that although the logging syntax is in most cases implementation-specific the semantic of such data is quite similar: a network activity is always started by an initiating host and/or by an initiating user, and it always has a target host and/or one or more target users. In this work we propose a solution to all these problems.

1.3 Document Organization

The next chapter reviews the used network-based discovery techniques currently used most often. Chapter 3 gives an idea of the overall approach followed in our work to complement these techniques with our proposal. Chapter 4 analyses information found in common network and application log files. Chapters 5 and 6 then propose how to warehouse and compute aggregated activities to prepare for the data mining to categorize assets and users. Chapter 7 shows a validation of the log analysis techniques on a small campus network, and Chapter 8 contains the conclusions and an outlook in terms of future work.

1.4 Summary

In this chapter we have mainly introduced the problem of assets tracking as a keyword for the provision and maintenance of an efficient, optimized network service. We have also given a short overview of the classical methods used for such purpose and described with their limitations. Then we stated our proposal, describing what benefits we expect to obtain respect to the other known methods.

Chapter 2

STATE OF THE ART

Besides the classical methods briefly described in Chapter 1 ("wall to wall", "financial assets tracking" etc.) an increasingly popular inventory method is to collect information using the network itself. Network-based inventories can significantly reduce the time and cost of an IT audit, and can also make regularly scheduled inventories feasible, providing more up-to-date information [7][54].

Network-based asset discovery and tracking techniques can be classified into "online" methods (to determine the actual state of end-systems, network and services) and "historic" log information processing (to analyze recorded network and service usage traces). Whereas online monitoring may also keep historic information, it cannot see into the past, i.e. into events that happened prior to its start. If the time period available for asset discovery is limited, i.e. too short to see all sporadically connecting devices, or if it is difficult to obtain sufficiently broad and deep administrative access to the overall infrastructure, it is attractive to reconstruct the global picture of computing assets and their usage from historic and redundant log information. Before focusing on the analysis of log information in the subsequent chapters, the following section surveys currently known techniques for online discovery.

As described in [1], there is no single technique that can perform an exhaustive network discovery, as every network has some peculiarity (e.g. some network parts are protected by firewalls, whereas others are accessible only from a few selected hosts) and also because not all networks run the same set of network protocols. In addition, network interface cards for mobile computers can easily be

shared (plug-n-play) and swapped, and more and more portables are already equipped with wireless network interfaces, making these devices difficult to track and identify as they can move and change link and network addresses during their lifetime. Network-based online discovery techniques can be further classified into methods that (i) passively listen and map the network, (ii) actively talk and walk the network and services, and (iii) interact and footprint individual hosts and target application services. The following sections highlight and position several such techniques.

2.1 Passive Network Mapping

Passive network mapping enables the discovery and identification of network assets in a purely passive fashion, i.e. without generating any kind of traffic that stimulates target machines in order to discover their presence [3].

Network Packet Sniffing

Packet sniffing consists of capturing packets that are received by one or more network adapters, and does not interfere with normal network operations as the packet capture application (network probe) generates no traffic whatsoever. As modern networks make intensive use of switches for filtering out unnecessary traffic, a probe can see only traffic directed to the host in which the probe is running and broadcast/multicast traffic. The network administrator's reaction is to adopt techniques such as ARP poisoning and port mirroring to avoid duplicating probes on each sub-network to be monitored [56]. Packet capture is location dependent, hence the probe should be placed where the traffic actually flows, which can pose quite a challenge. The probe needs to have decoders for each of the protocols the network administrator is interested in. As network traffic can be quite bursty, probes must be fast enough to handle all traffic in quasi real-time and to avoid loosing track of the ongoing traffic sessions.

Applications that belong to this category include ntop[13][14][15], and ethereal [53].

Subscription to Network and Syslogs

As most of the network devices and services store activity reports in log files, often even enabled for remote forwarding via syslog, it is quite common to subscribe to these log files for tracking network activities. In particular, log files can be very useful in the case of dial-in modem servers, corporate VPN gateways, mobile users, and WAP gateways. Some drawbacks of using logs are that their format is usually fixed and not customizable by network administrators, that it is necessary to periodically read the logs as they can wrap and hence overwrite historical information, and that access typically requires administrator privileges.

2.2 Active Network Mapping

There are several different techniques that can be employed for actively mapping network assets[5]. They all share the principle that the network needs to be exhaustively explored from a starting point using a repetitive algorithm that walks the entire network up to an endpoint or until the entire IP address range has been worked.

SNMP Walking of Network Topology

Starting point: the current default route of the host that performs the mapping. Recursive propagation algorithm: using SNMP [21] contact all adjacent routers, learn all their current interfaces, and read their ARP [62] table for learning all local hosts. Applications that belong to this category include NetView [55]. Specific MIBs can also provide hardware-related configuration information, e.g. allow the algorithm to drill down to racks, frames, and individual plugs in wiring closets. Termination condition: recurs until network closure or until a limited access authorization (SNMP community string) blocks the walking. The technique is

potentially hazardous for networks as SNMP traffic is not suitable for large networks and can interfere with normal operations. For security reasons, network administrators may deny remote SNMP GET operations. Moreover, SNMP can be either disabled or misconfigured.

Ping/Broadcast Ping

Starting point: the host that performs the mapping. Contact all hosts of the network being explored, e.g. using ICMP ECHO (a.k.a. ping) [20]. Every IP host is required to echo an *ICMP ping* packet back to its source. The ping tool therefore accurately indicates whether the pinged machine is alive or not (actually, since ping packets can get lost, we always ping an address twice, deeming it unreachable only if both do not elicit a reply). With suitably small packets, ping also has a low overhead. Pings to live hosts succeed within a single round-trip time, which is a few tens of milliseconds, so the tool is fast. Pings to dead or non-existent hosts, however, timeout after a conservative interval of 20 seconds, so pings to such hosts are expensive.

"Directed broadcast ping" refers to a ping packet addressed to an entire subnet rather than just one machine. This can be done by addressing either the '255' or the '0' node in the subnet (e.g. to broadcast to all nodes in the 128.84.155 subnet, ping 128.84.155.0 or ping 128.84.155.255—more generally, these two addresses corresponding to extending the subnet address either with all 0s or all 1s). A broadcast ping is received by all hosts in the subnet, each of which is supposed to reply to originator of the ping. This is useful in finding all the machines in a subnet. Ping broadcast however is not supported fully in all networks. In some networks, only the router responsible for that subnet responds to the broadcast ping (we refer to this as the *weak ping broadcast assumptio*n). In other networks, broadcast ping is not even responded to at all. These modifications prevent a denial-of-service attack called "smurfing" where a large subnet is broadcast with a

ping packet whose return address is set to that of the victim. The victim gets swamped with ICMP ping replies and soon dies.

Evaluation: End-to-end method that works in many situations where SNMP router walking is blocked. NAT and firewall devices block inbound IP ping sweeps, whereas unmanaged IP addresses can still talk to most servers, even in other network segments. Hence the starting point for ping sweeps should be selected carefully. Ping typically works as long as end systems have an inherent business need to communicate. Although the generation of ICMP traffic may interfere with normal operations, ICMP ECHO can be (partially) disabled for security reasons. Other techniques such as Nmap [56] or Traceroute [61] may produce better results in terms of efficiency and accuracy.

Traceroute

Traceroute [61] discovers the route between a probe point and a destination host by sending packets with progressively increasing TTLs. Routers along the path, on seeing a packet with a zero TTL, send ICMP *TTL-expired* replies to the sender, which tallies these to discover the path. Traceroute is usually accurate because all Internet routers are required to send the *TTL-expired* ICMP message. However, some ISPs are known to hide their routers from traceroute by manipulating these replies to collapse their internal topology. This reduces both the accuracy and the completeness of topologies discovered using traceroute. Traceroute sends two probes to every router along the path, so it generates considerably more overhead than ping. Since probes to consecutive routers are spaced apart to minimize the instantaneous network load, the time to complete a traceroute is also much longer than a ping.

DNS Network Domain Name-Space Walking

Starting point: the local DNS server [46]. Algorithm: walk the DNS space by performing a zone transfer in order to know all known hosts and DNS servers.

Recurs until network closure or until a DNS forbids the zone transfer. Evaluation: technique can produce misleading data as some DNS servers may be out of synchronization with the actual network state. Typically provides good information about stable network services and applications; naming conventions may allow further conclusions on intended main host usage (DNS, Notes, Mail Exchange, etc.). DNS walking is useless in non-IP networks of course, and fails on networks in which names have not been configured. Its results need to be carefully analyzed in particular when dynamic address protocols (e.g. BOOTP [64] and DHCP[65]) are in use.

DHCP Lease Information

Starting point: the local DHCP service or administrative access to the corresponding server. There is no standardized access across products. Microsoft's Win/NT Resource Kit contains utilities to find DHCP servers and list clients. Resulting data contains the currently assigned IP addresses with associated MAC address as key to client information. The value of this technique lies in particular in the tracking of devices connected only sporadically to the network.

Windows and Novell Network-Domains, LDAP, and Active Directory

Starting points: the local directory services of LDAP [66] and the corresponding Microsoft and Novell application networking. This technique nicely complements DNS walking on IP-based networks. On networks in which dynamic DNS is implemented results can partially overlap owing to misconfigurations; directories tend to have richer data [31].

2.3 Host and Service Mapping

Once a host has been discovered, we may want to employ specialized tools to learn about the operating system (OS) currently running and the services the host

provides. One of the principles of network mapping is that the more we know about a host the more we can find out. The easiest way to identify the OS is to parse the logon banners a server returns when opening TCP/IP ports. Unfortunately, not all hosts offer such services. However, a large class of workstations provides further Windows-specific data. If both approaches fail, the ultimate resort is to use advanced techniques to directly analyze the TCP/IP stack [5].

TCP/IP Stack Analysis and OS Detection

The standardized TCP/IP protocols allow a certain degree of local-system freedom. Such local choices may impact system tuning and application performance. Hence different stack implementations feature slightly differing behavior, especially when challenged with peculiar protocol situations such as bogus flags, out-of-scope packets, or windowing information. The current internet-shared database contains fingerprints for more than 500 IP implementations. The most popular tool is Nmap[56]. The results of stack-based OS analysis must be carefully interpreted as they are based on heuristics that can produce vague or even wrong results. Moreover, care must be exercised when interacting with some - typically old and badly maintained - hosts as the odd requests against the stack may crash the host currently being mapped.

UDP/TCP Port Scans

Port scanning is the technique that tries to communicate with remote ports, and map the TCP/IP services available from a host. Access can be tried by using either a small list of well-known ports (such as TELNET, FTP, HTTP and POP3), the entire range of named services, or by scanning the entire 64K-large port range. In addition, access can stop when granted (early close) or continue (full open) until the banners are displayed. The former can be an alternative to ping in network environments that block ICMP packets. There are also various

tools available to security analysts that further scan for possible security vulnerabilities [57]. To reduce the impact on end-systems and visibility in intrusion-detection systems, "stealth" modes are commonly available to sequence randomly across IP addresses and ports. Unfortunately port scan is a potentially hostile activity, hence it needs to be used carefully and only after the local network administrators have been informed. There is a growing list of personal tools that are able to detect port scans.

Remote Windows Fingerprinting

For Windows systems, there are specialized scanners that connect to the remote interfaces of Windows systems management. In particular, and especially with proper credentials, this yields a wealth of information on the hardware, networking, and software configuration of the remote host. An example is Winfingerprint [52].

2.4 Available Commercial and Freeware Products

We will survey now some of the most widely used commercial and freeware tools for network, systems, and inventory management with a particular focus on the methods they use to track assets.

2.4.1 Tivoli Net View

Tivoli NetView discovers TCP/IP networks, displays network topologies, correlates and manages events and SNMP traps, monitors network health, and gathers performance data [55]. For technical details on used network discovery technqiues, please refer to the following section 2.4.2.

2.4.2 HP OpenView

Since no one product can address all the aspects of network management, HP OpenView [67] consists of many products that address different aspects of the process.

Network Node Manager (NNM)

Network Node Manager is the foundation from which most of the HP OpenView products operate. When installed, the other HP OpenView products appear as added functionality within NNM. NNM not only functions as a solution on its own, but can collect data for, and forward data to, other HP OpenView products. Although NNM uses several protocols (such as TCP/IP, IPX/DMI, ICMP) to maintain communication channels with each managed device it is mainly based on SNMPv1 and SNMPv2 protocols [21].

When you start the NNM background processes, all IP and Level 2 devices (devices that support bridge, repeater, and MAU MIBs) on your network are automatically discovered and mapped out. If you are running NNM on a management station running the Windows NT operating system, IPX devices are also discovered and mapped out. This map is a visual representation of the communications channels established between NNM and the devices in your network. Be aware that this map is not a physical representation; rather, it is a logical representation. The accuracy of these communications channels between NNM and your network devices determine whether or not NNM can provide the information you need in order to manage your network.

The initial polling process may take several hours, or even over night, to discover all the devices on your network. The netmon service (background process) uses a combination of SNMP requests and ICMP pings transmitted over UDP and IPX to find out about the nodes on your network.

To discover the nodes on the network, netmon needs access to the following information:

- The subnet mask from the agent on the management station.
- The address of the default router in the management station's routing table.
- SNMP information from, at a minimum, the default router, and from other routers and nodes on the network.

For netmon to work, it requires the following:

- The management station must be running an SNMP agent.
- Nodes must be up and responding to ping requests to be discovered.
- All gateways/routers and the management station must have correctly configured subnet masks for all interfaces.

During IP discovery, netmon works best in these situations:

- The more routers in the network the better; and those routers should be running configured SNMP agents.
- The more nodes running configured SNMP agents the better.

Information about the discovered nodes is stored in NNM's databases and is used to automatically generate the network map.

Over time, NNM will discover new nodes on the network. However, if a new node never talks with a gateway or other nodes on the network that support SNMP, NNM may not find it. In this instance you can use menu items within NNM to send a low-level ICMP ping that forces NNM to discover the node; or you can add the node manually.

NNM takes advantage of information provided in three standard MIBs to discover bridges (switches) and hubs. These MIBs are the bridge MIB (RFC 1493), the repeater MIB (RFC 2108), and the 8023MAU MIB (RFC 1515). If a network device supports any of these MIBs, netmon will use the information reported to develop a model of the topology which better represents how and to what the device is interconnected. In the case of switches and bridges, additional information is gathered from the bridge MIB. In the case of hubs, addition information is gathered either from the repeater MIB or the 8023MAU MIB.

The status of non-IP or non-IPX interfaces on switches, bridges, and hubs is determined via SNMP based on the administrative and operating status of the port.

2.4.3 Peregrine InfraTools Network Discovery

InfraTools is a set of tools developed by Peregrine Systems which allow to track assets. The two most important discovery tools are the *InfraTools Network Discovery* and the *InfraTools Desktop Discovery* tools. InfraTools Network Discovery enables you to proactively manage the network resources and protect the business technology investment by providing accurate asset inventory [68]. As a vendor neutral solution, InfraTools Network Discovery identifies and continuously monitors all managed and unmanaged network devices without agents. The developers claim that it is able to identify all devices attached to the network with 99.99% accuracy, continuously exploring the network for every device by DNS name, IP address, MAC (Level 2), or SNMP field. The system evaluates a variety of data sources in order to maintain its real-time resource inventory. For managed devices it queries for standard MIB attributes such as manufacturer, model, URL, Y2K compliance, firmware version, and operational status. To assist with unmanaged devices or where incomplete or incorrect MIB data exists, device-specific scripts extract information in an effort to determine its traffic role. A

heuristic engine compensates for any false, incomplete, or incorrect information, determines the kind of device (such as router, switch, or workstation), and builds a data model of the device in the network.

Like OpenView NNM, InfraTools Network Discovery produces a physical topology network map.

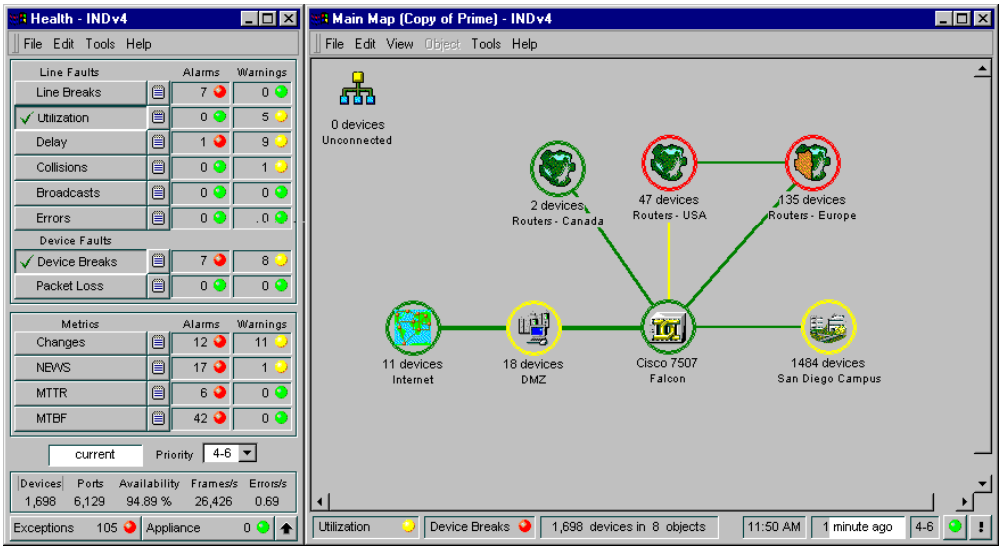

Figure 1: Infratools Physical Map

The system assigns specific device icons and line types to graphically represent the network in real-time, and adapts dynamically to stay current.

2.4.4 Microsoft Visio

Microsoft Visio 2000[63] delivers powerful, automated design and documentation tools for IT professionals who develop and maintain networks, databases, software applications, and web sites. The most significant feature for network managers is the auto-discovery and layout function. You can use this to automatically create logical diagrams of your network topology using representative icons for each network node--note that this only refers to IP-based

networks, so any IPX-only devices, some print servers for instance, won't be detected and won't show up on any discoveries. You're free to add such devices manually afterwards, but you shouldn't rely on Visio Enterprise as a comprehensive network inventory tool.

The auto-discovery feature uses a separate, but automatically launched program to search the network and find any nodes it can. You start by specifying whether the network is switched or routed. Since this distinction makes little sense on large networks, it's worth noting that the decision relates to the discovery method used. What it should ask is whether there is either one or more IP subnets on the network. In a routed network, the default gateway is used as a starting point for discovery. Its ARP cache is queried using SNMP to find any nodes on the network.

In a switched network, the PC will ping every possible address on the subnet of which it is a member. This is a more time-consuming method that can generate more traffic than an ARP cache discovery, and with several hundred nodes on a network, the discovery may take hours and the layout even longer. You can also do an advanced discovery, where a combination of both methods is used. You can also choose to ignore devices that aren't SNMP-enabled. Since most desktop PCs don't have SNMP responders installed, this will reduce the number of items that Visio tries to include on the diagram.

The database viewer application lists devices by type, showing the interfaces and attached networks for each device in the database. This can be used to prune the database of entries that aren't of interest.

2.4.5 Peregrine InfraTools Desktop Discovery

InfraTools Desktop Discovery [68] extends the breadth of asset inventory provided by the tools described above to include user-input (i.e. user name,

location, department, cost codes etc.) and specific data extracts (i.e. embedded asset information such as asset tag and serial numbers).

In regards to the hardware scan, InfraTools Desktop Discovery searches through the usual sources (i.e. Windows registry) like many inventory tools, but also couples information from these sources with its own hardware level scans. These scans utilize new and emerging asset management standards as well as existing technologies such as DMI and SMBIOS.

One of InfraTools Desktop Discovery's strongest differentiators is its inherent software scanning abilities. Two of the following distinct capabilities emerge within the software scan:

Application Recognition - This component is based on the leading application recognition technology, which utilizes heuristic algorithms to analyze file relationships and provide application and version recognition.

File Recognition - In addition to applications, InfraTools Desktop Discovery is able to recognize any file (user data or otherwise) from local hard disks, irrespective of any partition format or independent operating system.

2.4.6 Nmap

Nmap ("Network Mapper") is an open source utility for network exploration or security auditing[56]. It was designed to rapidly scan large networks, although it works fine against single hosts. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (ports) they are offering, what operating system (and OS version) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. Nmap runs on most types of computers, and both console and graphical versions are

available. Nmap is free software, available with full source code under the terms of the GNU GPL.

Nmap supports dozens of advanced techniques for mapping out networks filled with IP filters, firewalls, routers, and other obstacles. This includes many port scanning mechanisms (both TCP & UDP), OS detection, pings sweeps, and more. The result of running nmap is usually a list of interesting ports on the machine(s) being scanned (if any). Nmap always gives the port's "well known" service name (if any), number, state, and protocol. The state is either 'open', 'filtered', or 'unfiltered'. Open means that target machine will accept connections on that port. Filtered means that a firewall, filter, or other network obstacle is covering the port and preventing nmap from determining whether the port is open. Unfiltered means that the port is known by nmap to be closed and no firewall/filter seems to be interfering with nmap's attempts to determine this. Unfiltered ports are the common case and are only shown when most of the scanned ports are in the filtered state.

Depending on options used, nmap may also report the following characteristics of the remote host: OS in use, usernames running the programs which have bound to each port, the DNS name, whether the host is a smurf address, and a few others.

2.4.7 Winfingerprint

Winfingerprint [52] is a Win32 based security tool that is able to determine OS, enumerate users, groups, shares, transports, sessions, services, service pack and hotfix level, and data and time, and tcp ports.

Figure 2: WinFingerPrint Window

Winfingerprint is capable of querying/enumerating information without any logon credentials via the Windows null session. Essentially, the null session is an unauthenticated connection to an NT machine used for anonymous information gathering (user lists, for example). While the availability of null sessions eases some administrative burden by providing services to tools like Explorer, null sessions are similar to the Unix "finger" service. They are an intruder's dream: access to users, shares and other potentially useful information, remotely and anonymously. While null sessions can be disabled using a registry hack most organizations have not made these changes.

2.4.8 Tcpdump

The active (e.g. intrusive) performance monitoring conducted by previously described tools would be well complemented with an understanding of passive monitoring, that is genuine user traffic as it comes in and goes out of the network. Tcpdump is a powerful tool that allows people to sniff network packets and make some statistical analysis out of those dumps[48]. One major drawback to tcpdump

is the size of the flat file containing the text output (see section 4.7.1 for a detailed example). The simplest way to use Tcpdump is to run it with just an *-i* switch to specify which network interface should be used. This will dump summary information for every Internet packet received or transmitted on the interface. However, Tcpdump provides several important options, as well as the ability to specify an *expression* to restrict the range of packets you wish to study.

Rather than rehash here what is better documented elsewhere, I suggest you read Tcpdump's well written manual page [71].

2.4.9 IBM IDD Project

Intelligent Device Discovery (IDD) is an ongoing project at the IBM Zurich Research Laboratory. The IDD tool launches an encompassing low-intrusive network-based discovery of hardware and software in a heterogeneous unknown systems and networking environment [34].

The IDD tool discovers all IP network-attached devices through a multi-stepped, non-intrusive process:

- NetView is used to discover information about the network. The NetView IDD extractor tool extracts SNMP and non-SNMP gathered information from NetView to further help provide device information, active networks, and validate the IP ranges to be scanned by the IDD suite of collectors.

- IP Scanner discovers functioning IP addresses, to then be scanned in scanners below.

- Port Scanner combines port scanning (scans TCP/UDP ports: ftp, telnet, smtp, http, nameserver) and fingerprinting to determine OS.

- Winfingerprint is used to gain remote read-only access to the Windows registry and Netbios information. This tool provides specific configuration data of a Windows Server or Windows Desktop/Laptop. NO Port Scanning is involved.

- DIG performs DNS zone transfer to gather IP addresses and namespace info.

- DHCP Collector imports DHCP logs, capturing the latest dynamic IP addresses assigned to devices.

The integration of such tools with many others allows the discovery of PC's, Intel & Unix Servers, LAN printers, Routers, Hubs, the O/S, IP & Ethernet Addresses etc.

The tool is implemented on an open data-integration and automation platform and integrates the various network and host mapping tool sources of information into an aggregated consistent data model in a DB2 database.

2.4.10 Conclusions

Although previously described commercial and freeware products all represent powerful and valid solutions, none of them seems to use log files as source of information for asset tracking. The table below summarizes the characteristics of previous tools according to the category they belong:

Products Categorization	Network Management [55] [67]	Network Scanners [56]	OS specific Scanners [52]	Asset Management [70] [58]	Network Doc. Tools [58] [63]	Consulting tools [34]
Discovery:						
Network Topology	Yes	No	No	No	Yes	Yes
IP/Port scanning	No	Yes	No	No	Yes	Yes
Stack OS analysis	No	Yes	No	No	No	Yes
NetBIOS sysinfo	No	No	Yes	No	Yes	Yes
Win Tel Registry	No	No	Yes	No	Yes	Yes
Windows scanner	No	No	No	Yes	No	Yes
Unix scanner	No	No	No	Yes	No	Yes
Log Analysis	No	No	No	No	No	No

Figure 3: Product Discovery Categories

In terms of "historic" log information processing methods we did not find any interesting work in which these methods have been successfully applied to asset discovery and tracking. We believe instead that log data can be very useful in many situations. The Internet abounds with tools that parse individual log files (look at [68] [69] to have some examples) and produce a wealth of statistics on how the particular service has been used, but in our case we want something more. We want to integrate information coming from different protocols (e.g. different logs) into a consistent repository in order to build a more accurate inventory of computing assets, complementing the information collected using the other surveyed network based techniques with usage information, e.g. information about how assets (and not only protocols) are actually used.

2.5 Summary

The goal of this chapter was to survey both the currently most popular network-based techniques for asset discovery and tracking, and the actually most used commercial and freeware products that make use of them. We have seen that several tools are available but all of them have in common that they do not provide enough information about the usage of discovered assets. The challenge

in next chapters is now to convince the reader that logs data can be useful for such purposes in many situations.

Chapter 3

THE OVERALL APPROACH

In this chapter we will give a brief overview about how the process of transforming raw and uncorrelated log data into meaningful information has been organized. As the details will be given in the subsequent chapters we will merely describe here the main problems we encountered in trying to integrate data coming from different protocols into a common repository.

We have already said that our proposed overall approach is to complement basic network-based discovery with the combined log information from network and application servers, and then to compute an aggregate picture of assets and prepare data for categorization with data-mining techniques [11] [12]. The main benefit of merging different kinds of data is clearly that each log potentially represents a slice of both users and computing assets behavior. HTTP logs therefore provide information on how the users use the web whereas SMTP logs tell us how they use the email service etc., and together they contribute to a more complete user/asset behavior model. The more logs we can analyze, the more information we potentially have. For this reason we decided also to analyze logs such as the Socks, Gateways and Firewalls logs (generically called Net logs from now on) that are not specifically related to a particular protocol but that can help us to have more evidence on how users deploy the network.

3.1 Data Integration Architecture

Our first goal in selecting the data integration architecture is flexibility, in terms of the effort involved in introducing a new service. We have therefore chosen to split our system into two parts according to two different phases individuated:

- A first phase called *Data Parsing and Aggregation,* in which data from several different protocols are parsed and aggregated in order to be stored into a relational database.

- A second phase called *Data Integration,* in which data resulting from the first phase are cleaned and integrated into more meaningful units, called Activities.

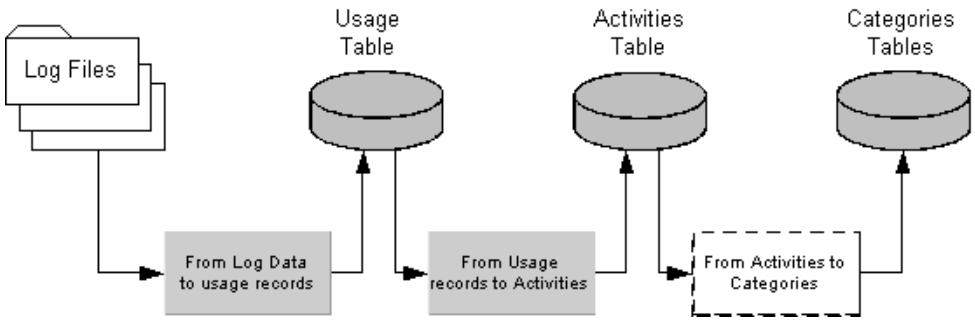

Figure 4: The Overall Approach

Figure 4 also shows a third phase in which Activities obtained as result of phase 2 are used as sources of categorization. Although this Categorization phase is not really a part of this work, we will show some preliminary categorization results in Chapter 7, which were obtained by simply querying the activities table, leaving the application of data-mining algorithms to discover hidden information to future work.

3.1.1 First Phase: Data Parsing and Aggregation

As stated, one of our goals is flexibility. For the moment this means that we want to have a structure such that if we decide to introduce data about a new service we do not have to modify the entire environment. Figure 5 shows the entities involved in this first phase.

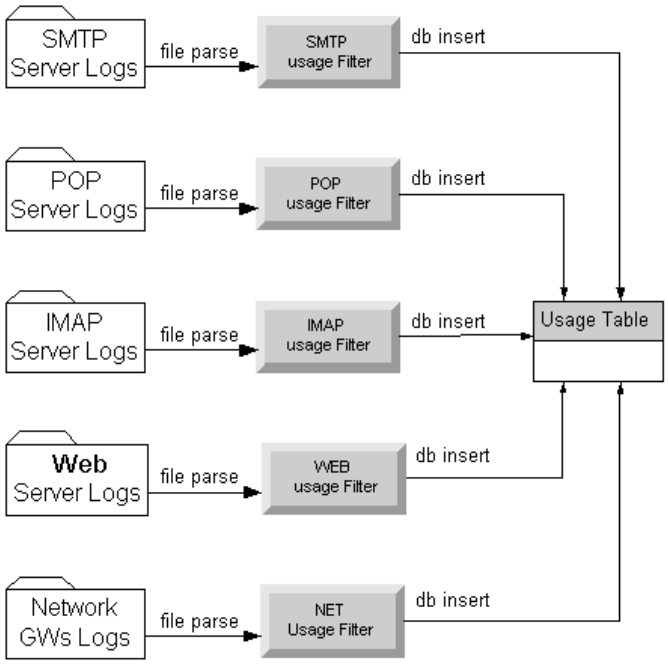

Figure 5: Data Collection Architecture

In our study, we will focus on SMTP, POP/IMAP, WEB, and NET log sources[18][19][25]. For each of these sources we built a parser that not only transforms text log entries into database records but also aggregates and reduces the data size in such a way that they can be stored in a database. The size of the data is in fact the biggest problem encountered in this phase. To get an idea, just observe that in our test data samples (one week of traffic) from a small/medium-sized network we handled about 100 millions of log entries.

During this first phase, we dealt with two contrasting requirements: on the one hand, it is not possible to deal with such big sets of data so that we must reduce it, whereas on the other hand we do not want to lose information in passing from log entries to usage records. A first degree of filtering has been applied in this phase to cleaning the logs from all those "dirty entries" such as errors messages or debug information that are not of interest to us but that log files unfortunately report quite often. The result of this phase should be as accurate a representation of aggregated logs data in a relational format as possible.

Some effort has also been dedicated to the code architecture such that it should not require numerous changes in order to parse logs coming from different servers implementing the same protocol (unless they use completely different logging procedures). For this we have chosen to make use of regular expressions. This means that if two logs differ only in the syntax used the code can be completely reused simply changing the regular expression that defines the syntax used for that particular log file. We do not want to go into too much detail here about the code, but refer the interested reader to Appendix A.

3.1.2 Second Phase: Data Integration

The input of this phase consists of the *Usage* table obtained as result of the *Data Parsing and Aggregation* phase. Starting from it, the goal is now to define more meaningful units by trying to eliminate any usage entries that are redundant and that could yield false results. The redundancy can be viewed as a consequence of the intrinsically distributed nature of network usage. In general, note the following:

- The same activity can be observed at several moments in the time.
- The same activity can be observed in several locations.

- The same activity can be observed by several protocols.

The first means that an activity generally consists of several usage records. We are no longer interested in knowing that a user has browsed a certain web page, but rather in information such as: "user X has surfed from host Y for W minutes, generating Z bytes of traffic". Similarly we are no longer interested in knowing that at a certain time an attempt to download emails has been started from a certain host by a certain user, but we would like to know, for example, that for a certain time a POP client demon has been running on that host polling the POP server for new emails every X seconds. Because of the two other bullets, this process of transforming Usage records into such units of information has to be done carefully because redundant data could falsify the result. The example below allows us to be more concrete by showing a situation in which the same activity has been logged in two different locations by two different protocols.

Example

Let us imagine a situation like that shown in Figure 4.

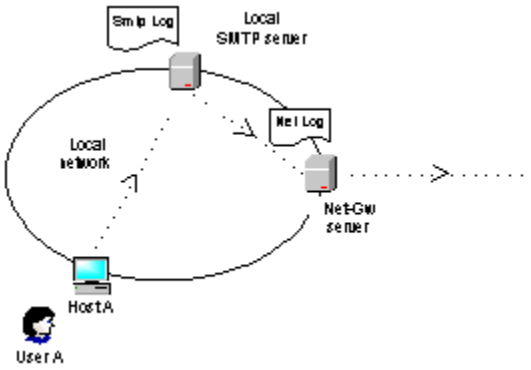

Figure 6: An example of redundancy

In such a situation the activity to be traced is the act of user A sending an email to a remote user. The user sends the email to the local SMTP server, which then forwards it to the remote SMTP server, passing through the local Net-Gw server.

In this example (e.g. with this network topology) this activity has been traced twice: first by the local SMTP server and then by the Net-Gw server. This means that (at least) two usage records document this fact. When we model activities only one of them has to be selected. However, we want emphasize the fact that the Net-Gw log still is necessary, for example, for tracking all attempts made by a local user to send emails using extranet SMTP server whose log files are not available.

As shown in Figure 7, a specific agent we called Activities Builder has been built for this purpose:

Figure 7: Data Integration Phase

The benefit of this multitiered architecture is that we can change the way we build activities by abstracting from the specific application server log-file formats and therefore without having to change anything in data collecting.

3.3 Summary

This chapter gave a short overview of the main problems encountered in the two main phases that led us from raw and uncorrelated log data to what we called activities records. The keywords in the first phase (described in more detail in Chapter 4) have been data parsing and data-size reduction. The result is a usage table that represents an easily manageable picture of log data in a relational format. In the second stage (described more fully in Chapter 5) related usage records from multiple network and application protocols and observation points are aggregated into a server-independent perception of activities. The main effort

in this second phase has focused on noise reduction. The result of this process is an activities table that will now be used as starting point for the categorization of computing assets.

Chapter 4

LOG FILES ANALYSIS

Our work starts by analyzing the various sources of data we selected. We will therefore show several examples of log files generated by different protocols as well as by different implementations of the same protocol. Although this work can be extended or adapted to a specific network, we decided to focus on the services most used by network users: web and email. In our University of Pisa data samples we observed that more than two thirds of the entire traffic is generated by these services. We will also try to describe the entities involved during the normal use of the services mentioned. We will see that in some cases there are standards in how servers log information whereas in some others this depends on the particular implementation. We will however show that although log entries can change depending on the particular implementation, the entities involved (people and assets) and the information about them is almost the same.

4.1 HTTP logs

HTTP log files are one of the most important sources of data in our study. Fortunately in this case the information that HTTP servers log has been standardized by the W3 Consortium, and today almost all the web servers adhere to these standards. There are two standards for http log entries. The first is called **Common Log File Format** [32], the second **Extended Common Log File Format** [33].

4.1.1 The Common Log File Format

The Common Log File Format has the following fields separated by a space:

- **RemoteHost**: The DNS name (or the corresponding IP address if no DNS service is available) of the host that made the request
- **RemoteUser**: The remote login name of the user (if not available, a dash is typically placed in the field).
- **AuthUser**: The username as which the user has authenticated himself. This is available when using password-protected WWW pages. (If not available, a dash is typically placed in the field.)
- **Timestamp**: Date and time of the request.
- **RequestType**: The request line exactly as it came from the client (i.e., the resource name and the method used to retrieve it [typically GET]).
- **Status**: The HTTP response code returned to the client; indicates whether the file was successfully retrieved, and if not, which error message was returned.
- **Bytes**: The number of bytes transferred.

Here are a few lines of a Common Log File created by an Apache web server version 1.3.19 running on a Windows2000 system:

```
131.114.4.xxx - - [25/Aug/2001:22:54:16 +0200] "GET /main.html HTTP/1.0"  200  4234
131.114.4.xxx - - [25/Aug/2001:22:54:16 +0200] "GET /images/header.gif HTTP/1.0"  200  9342
131.114.4.xxx - - [25/Aug/2001:22:54:16 +0200] "GET /images/dot.gif HTTP/1.0" 200  8765
131.114.4.xxx - - [25/Aug/2001:22:54:17 +0200] "GET /foto/iris.jpg HTTP/1.0" 200 7104
```

By default, Apache servers generate log files in the Common Log File Format.

4.1.2 The Extended Common Log File Format

The Extended Common Log File Format is obtained from the Common Log File Format simply by adding the following two fields separated by a space:

- **Referer**: The URL the client was on prior requesting the current URL. (If it cannot be determined a dash will be placed in this field.)
- **UserAgent**: The software the client claims to be using. (If it cannot be determined a dash will be placed in this field.)

Here are a few lines of the Extended Common Log File Format logged by an Apache web server version 1.3.19 explicitly configured to use this kind of log format:

131.114.4.69 - - [25/Aug/2001:22:54:16 +0200] "GET /main.html HTTP/1.0" 200 4234 "-" "Mozilla/4.71 [en] (WinNT; I)"

131.114.4.69 - - [25/Aug/2001:22:54:16 +0200] "GET /images/header.gif HTTP/1.0" 200 9342 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

131.114.4.69 - - [25/Aug/2001:22:54:16 +0200] "GET /images/dot.gif HTTP/1.0" 200 8765 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

131.114.4.69 - - [25/Aug/2001:22:54:17 +0200] "GET /foto/iris.jpg HTTP/1.0" 200 7104 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

4.1.3 E-R Diagram for HTTP Servers

We derive the entity-relationship diagram of http logs

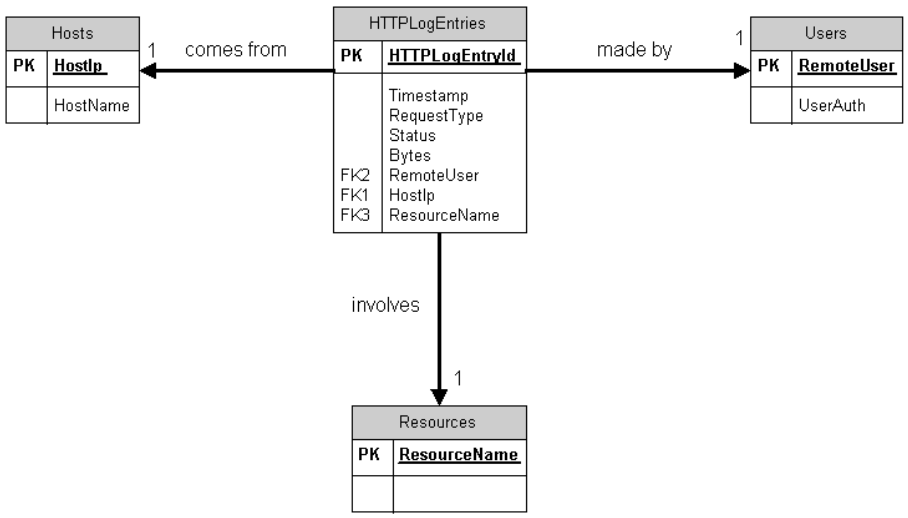

Figure 8: Web Log E-R Diagram

Figure **8** shows that by analyzing a generic http log we can learn that userX from

hostY requested resourceZ at a certain time. In fact this is not completely true because unless the resource or, in general the web site that contains it has some form of access restriction by means of explicit user authentication, the web log contains no information about user. Unfortunately this is true in the overwhelming majority of cases. On the other hand the diagram does not show that http logs also allow us to obtain information about the operating system and the browser used by the user. For example a *UserAgent* field with value *"Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"* tells us that the machine used runs the Windows NT version 5 operating system and that the user has used Microsoft Explorer version 5.01 to access the resource. This information can be very useful for example for asset and user inventory tools to integrate or complement information collected in other ways.

4.2 PROXY Logs

Web proxies [35][41] enable a more efficient access to popular websites. Instead of downloading data from a remote site, the page is held in a cache on the server. This means quicker access to the page and no overhead in bandwidth resources every time the same page is downloaded. Although there seems not to be a predominant proxy server on the market, the content of proxy logs is quite similar in the proxy servers of the various vendors. Proxy servers can be used with several protocols; however the one used most is the http protocol. Below I show few lines from a log file created by the Squid [42] http proxy server (Squid is a widely used proxy caching server for web clients, supporting FTP, gopher, and HTTP data objects):

```
975963427.854   2310 193.43.104.202 TCP_MISS/200 25976 GET http://quicken.excite.com/ -
DIRECT/quicken.excite.com text/html

975963428.621    477 193.43.104.202 TCP_REFRESH_HIT/200 6705 GET
http://a1896.g.akamai.net/7/1896/942/0004/quicken.excite.com/g/excitelogo.gif -
DIRECT/a1896.g.akamai.net image/gif
```

```
975963429.749   1295 193.43.104.202 TCP_MISS/200 12348 GET
http://image.eimg.com/ads/141650_am_ban_trvl_x7_x_468x60_3.gif - DIRECT/image.eimg.com
image/gif

975963434.902    538 193.43.104.202 TCP_REFRESH_HIT/200 3837 GET
http://quicken.excite.com/images/hp_chart_green.gif - DIRECT/quicken.excite.com image/gif

975965970.299   7524 193.43.104.202 TCP_MISS/404 1504 GET
ftp://ftp.dlux.net/software/windows/compression/winzip95.exe - DIRECT/ftp.dlux.net -
```

Here is the meaning of various fields:

- **Timestamp**: A temporal reference. The value is the raw Unix time (since Jan 1, 1970) plus milliseconds.
- **Elapsed**: The time elapsed (milliseconds) during the client connection.
- **FromHost**: The IP address of the client host.
- **Code**: The "cache result" of the request.
- **Status**: The HTTP status code (200 = Ok, etc.).
- **Bytes**: The number of bytes delivered to the client.
- **Method**: GET, HEAD, POST, etc. for HTTP requests.
- **URL**: The requested URL.
- **AuthUser**: Always NULL ("-") for Squid logs.
- **PeerStatus**: A status code that explains how the request was forwarded, either to your peer (neighbor) caches, or directly to the origin server.
- **PeerHost**: The host to which the request was forwarded.
- **PageType**: The page extension.

As we can see, they are very similar to those that we saw when we analyzed http logs in section 3.1. In particular, for http requests we can find all the information we would find into a Common Log File Format: we have a timestamp (although in a different format), the IP address of the host that made the request, the name of the resource requested, etc. The E-R diagram is the same as that for http logs and therefore will not be shown here.

4.3 SMTP Logs

Unfortunately, in this case there is not a standard log-file format. I will therefore examine the two most widely used implementations of this protocol [18] in order to find a common denominator of the information we can find in these logs. Although there is no standard, the two types of SMTP servers chosen should cover almost the totality of SMTP servers actually running on Internet.

4.3.1 Sendmail

Here are some lines generated by a Sendmail server [43]:

> Dec 13 05:28:27 mailhub sendmail[26690]: FAA26690: from=<user@has.a.godcomplex.com>, size=643, class=0, pri=30643, nrcpts=1, msgid=<19981032.CAA22824@has.a.godcomplex.com>, proto=ESMTP, relay=user@has.a.godcomplex.com [216.32.32.176]
>
> Dec 13 05:29:13 mailhub sendmail[26695]: FAA26695: from=<root@host.ccs.neu.edu>, size=9600, class=0, pri=39600, nrcpts=1, msgid=<199812131029.FAA15005@host.ccs.neu.edu>, proto=ESMTP, relay=root@host.ccs.neu.edu [129.10.116.69]
>
> Dec 13 05:29:15 mailhub sendmail[26691]: FAA26690: to=<user@ccs.neu.edu>, delay=00:00:02, xdelay=00:00:01, mailer=local, stat=Sent
>
> Dec 13 05:29:19 mailhub sendmail[26696]: FAA26695: to="|IFS=' '&&exec /usr/bin/procmail -f- ||exit 75 #user", ctladdr=user (6603/104), delay=00:00:06, xdelay=00:00:06, mailer=prog, stat=Sent

As outlined in Figure **9**, each line has at least one partner entry that shows the source and destination of each message. When a message enters the system it is assigned a unique "Message-ID", highlighted in Figure 9, that identifies the message while it is being processed. This Message-ID allows us to associate related lines in an interleaved log file.

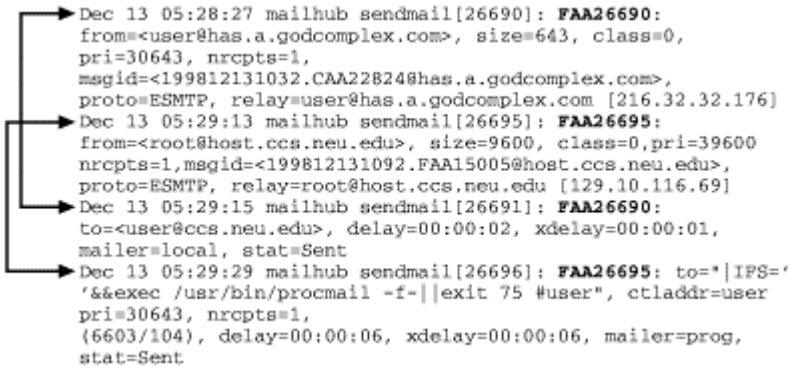

Figure 9: Sendmail Log Example

Here is an example of an entry logged when a message has entered the system:

Jun 3 09:00:13 localhost sendmail 9852]: JAA09852: from=<francisco@franciscoCompany.com>, size=955, class=0, pri=90955, nrcpts=3, msgid=<3CFE3C6E0508B0FFFD70BB4BF@SANTCO>, proto=ESMTP, relay=franciscoHost.franciscoCompany.com [194.113.245.71]

In this kind of entry we find the following information

- **Timestamp:** Month, day and time at which the message was received by the local smtp server.
- **HostRecorder:** The name of the host that has logged the entry.
- **InternalId:** An internal message ID that allows the system to associate related lines in an interleaved log file.
- **From:** The name of the sender in the form of *senderName@senderDomain.*
- **Size:** The size of the incoming message in bytes.
- **Class:** the numeric value defined in the sendmail configuration file for the keyword given in the *Precedence* header of the processed message.
- **Pri:** The initial priority assigned to the message.
- **Nrcpts:** The number of recipients for the message.
- **MsgId:** A unique message identifier defined as *local-part@domain* that allows linking the message across services.
- **Proto:** The protocol that was used when the message was received; this is either SMTP, ESMTP, internal, or assigned with the -p command-line switch.

- **Relay:** Where the message comes from. The semantic of this field can vary. In this case, assuming that the message has been sent from a local user, this field contains either the IP address or DNS name (sometimes both) of the host that forwarded the message to the local SMTP server (usually the host from which the local user has sent the message).

And here is the corresponding entry logged once the local SMTP server has forwarded the message to the remote SMTP server:

> Jun 3 09:03:00 localhost sendmail[9854]: JAA09852: to=<pablo@pabloCompany.com>, delay=00:02:47, xdelay=00:02:47, mailer=esmtp, relay=smtpServer.pabloCompany.com. [192.65.17.15], stat=Sent (2.0.0 f537mq511438 Message accepted for delivery)

This kind of entry contains the following information:

- **Timestamp:** Month, day and time at which the message was forwarded to the remote smtp server.
- **HostRecorder:** The name of the host that as logged the entry.
- **InternalId:** An internal message ID that allows the system to associate this line with the one showed above.
- **To:** The name of the recipient in the form *addresseeName@addresseeDomain.*
- **Delay:** The total message delay (the time difference between reception and final delivery).
- **Xdelay:** The total time the message took to be transmitted during final delivery. This differs from the delay= equate, in that the xdelay= equate only counts the time in the actual final delivery.
- **Mailer:** The symbolic name (defined in the sendmail configuration file) for the program (known as delivery agent) that performed the message delivery.
- **Relay:** Shows the name of the host to which the message has been forwarded (in this case the SMTP server of addressee).
- **Stat:** The delivery status of the message (Sent, Deferred, User unknown, etc.).

In conclusion, we say that for each message entering the system we will have an entry logged when the message was received and an entry for each delivery attempt (normally one).

4.3.2 Microsoft Exchange

Another widely used SMTP server is Microsoft Exchange [44]. In this case the information that is logged when the default message-tracking configuration is in use is the following (Exchange 2000) [45]:

- **Date**: Date when the entry was logged.
- **Time:** Time at which the entry was logged.
- **FromHostIP**: IP address of connecting client.
- **FromHostName**: Hostname of connecting client.
- **PartnerName**: Name of the messaging service the message is being handed off to. In Exchange 2000, the service could be SMTP, X400, MAPI, IMAP, POP3, or STORE.
- **RecorderIP**: IP address of the server making the log entry.
- **RecorderName**: Host name of the server making the log entry
- **ToUser**: Message recipient (SMTP or X.400 address).
- **EventID**: Integer corresponding to the event ID of the action logged, that is, sent, received, deleted, retrieved, and so on.
- **MSGID**: Message ID.
- **Priority**: -1,0,1 corresponding to low, normal, high., respectively.
- **Recipient Report Status**: A number representing the result of an attempt to deliver a report to the recipient.
- **TotalBytes**: Length of message in bytes.
- **NumberRecipients**: Total number of recipients.
- **Origination Time**: Timestamp of when the message first entered the Exchange 2000 organization (either via an external gateway or by creation through a client). Can be used in conjunction with the *Date* field to determine how long the message has been in the Exchange organization.
- **Encryption**: For the primary body part: 0 if no encryption, 1 if signed only, 2 if encrypted.
- **ServiceVersion**: Version of the service making the log entry.
- **LinkedMSGID**: If there is a MSG ID from another service, it is given here to link the message across services.

- **FromUser**: Primary address of the originating mailbox. This could be SMTP, X.400, or DN, depending on transport.

4.3.3 E-R Diagram for SMPT Servers

Figure **10** represents the relationships between the entities involved into the process of sending/receiving e-mail derived from the logs shown:

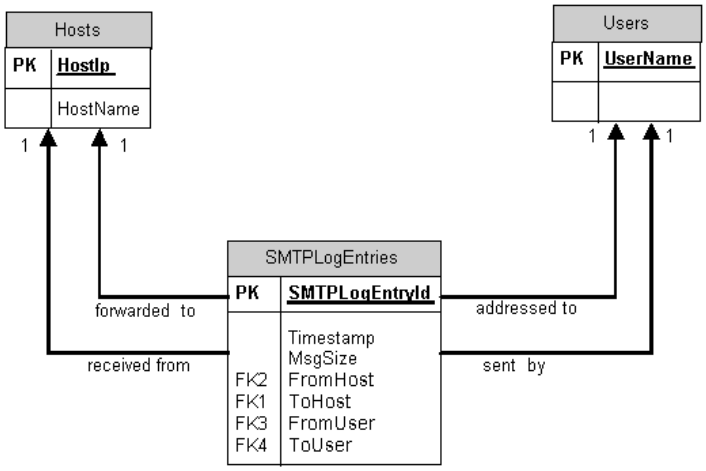

Figure 10: SMTP Log E-R Diagram

As you can see from the E-R diagram, also in this case we have selected a set of properties that we can reasonably assume to find in every SMTP server, and omitted all those details that are insignificant. The result is that by looking at a generic SMTP server log we can say that at a certain time *FromUser* has sent a message of *MsgSize* bytes from *FromHost* to *ToUser* passing via *ToHost*.

4.4 POP / IMAP Logs

Post Office Protocol [19] and Internet Mail Access Protocol [20] are Internet protocols that allow clients to download e-mails from their remote mailbox. The first aim in POP/IMAP definitions was in fact merely to permit dynamic access

to a maildrop on a server host in a useful fashion from various workstations. Usually, this means that POP/IMAP protocols are used to allow a workstation to retrieve mails the server holds for it.

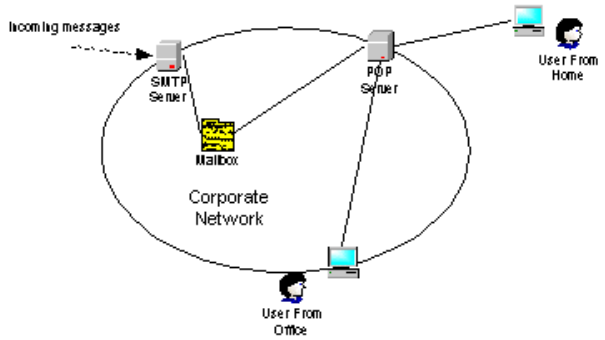

Figure 11: Pop/Imap Service

In a typical situation (showed in Figure **11**) a message is first stored into the filesystem by the SMTP server that received it and then, when the user requests to download new e-mails, it is extracted from the mailbox by the POP/IMAP server and sent to the user.

In this case there are not standards regarding the information that these kinds of servers log during their activity, but again our requirements will be sufficiently weak to be satisfied by almost all of the various implementations. Here are a few lines logged by UW-Imap server:

> Mar 24 00:32:15 aladdin imapd[5330]: imap service init from 9.4.16.57
>
> Mar 24 00:32:15 aladdin imapd[5330]: Authenticated user=mfi host=giuncarico [9.4.16.57]
>
> Mar 24 00:32:16 aladdin imapd[5330]: Moved 3864 bytes of new mail to /home/mfi/mbox from /var/spool/mail/mfi host= giuncarico [9.4.16.57]
>
> Mar 24 00:32:17 aladdin imapd[5330]: Logout user=mfi host=giuncarico [9.4.16.57]
>
> Mar 26 21:39:23 aladdin imapd[7467]: imap service init from 9.4.16.64
>
> Mar 26 21:39:23 aladdin imapd[7467]: Authenticated user=ydu host=jasmine [9.4.16.64]
>
> Mar 26 21:39:23 aladdin imapd[7467]: Moved 1478 bytes of new mail to /home/ydu/mbox from /var/spool/mail/ydu host= jasmine [9.4.16.64]

```
Mar 26 21:39:24 aladdin imapd[7467]: Logout user=ydu host=jasmine [9.4.16.64]
```

Below you find some examples of log entries generated by the Popd server used for POP service at the Pisa Computer Science Department:

```
Sep  4 12:01:08 apis ipop3d[13719]: pop3 service init from 131.114.4.xxx
Sep  4 12:01:08 apis ipop3d[13719]: Auth user=verdi host=pc-verdi [131.114.4.xxx] nmsgs=8/8
Sep  4 12:01:08 apis ipop3d[13719]: Logout user=verdi host=pc-verdi [131.114.4.xxx] nmsgs=8 ndele=0
Sep  4 12:01:13 apis ipop3d[13727]: pop3 service init from 131.114.2.yyy
Sep  4 12:01:13 apis ipop3d[13727]: Login user=neri host=pc-neri [131.114.2.yyy] nmsgs=0/0
Sep  4 12:01:13 apis ipop3d[13727]: Logout user=neri host=pc-neri [131.114.2.yyy] nmsgs=0 ndele=0
```

In the first case, four log entries are logged for each downloading attempt. The first when the server receive the request from the client (init service), the second when the user authenticates itself by sending its credentials (Authenticated), the third when new messages have been copied to the local mailbox (Moved…), and the fourth when the client logs out from the service (Logout). The second case is more or less similar to the first except that the "Moved…" entry is lacking. Although the two logs use different ways to log data (the syntax is not exactly the same and the number of entries generated for each download attempt is different) the semantic is almost the same in terms of resulting information: in both cases we can learn that at a certain time a certain user has downloaded new e-mails from a certain host, plus some information about the session (number of bytes moved in the first case, number of messages found/deleted/left in the second case).

Figure 12 graphically represents the above statements:

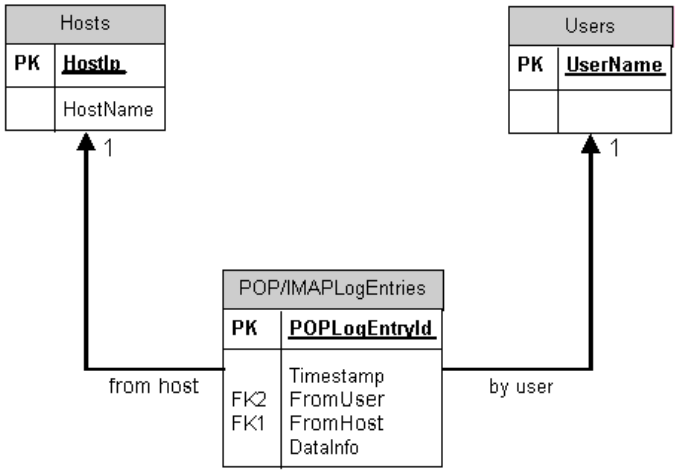

Figure 12: E-R Diagram for Pop/Imap Logs

In this case, a typical Pop/Imap object can be seen as consisting of the following fields:

- **Timestamp:** a temporal reference (when the entry was logged).
- **FromUser:** the login name of the user who has connected to the POP/IMAP server.
- **FromHost:** the name or IP address of the host to which messages have been moved.
- **DataInfo:** Information about emails downloaded.

How these Pop/Imap objects have actually been built will be explained in Chapter 4.

4.5 FTP Logs

Although numerous different FTP servers are available on the market, the information logged by them seems to be quite the same. Here are some sample lines from a wu-ftpd version 2.6.x server transfer log:

Sun Dec 27 05:18:57 1998 1 nic.funet.fi 11868 /net/ftp.funet.fi/CPAN/MIRRORING.FROM a _ o a cpan@perl.org ftp 0 * c

Sun Dec 27 05:52:28 1998 25 kju.hc.congress.ccc.de 269273 /CPAN/doc/FAQs/FAQ/PerlFAQ.html a _ o a mozilla@ ftp 0 * c

Sun Dec 27 06:15:04 1998 1 rising-sun.media.mit.edu 11868 /CPAN/MIRRORING.FROM b _ o a root@rising-sun.media.mit.edu ftp 0 * c

Sun Dec 27 06:15:05 1998 1 rising-sun.media.mit.edu 35993 /CPAN/RECENT.html b _ o a root@rising-sun.media.mit.edu ftp 0 * c

Wuarchive-ftpd [49] is a replacement ftp daemon for Unix systems developed at Washington University. Wu-ftpd probably is the most popular ftp daemon on the Internet, used on many anonymous ftp sites throughout the world. A wu-ftp (version 2.6) log entry has the following fields:

- **Timestamp:** a temporal reference when the entry was logged.
- **TransferTime:** the duration of the transfer.
- **RemoteHost:** the name of the remote client.
- **FileSize:** the size of the file transferred.
- **FileName:** the name of the file transferred.
- **Transfer-type:** a single character indicating the type of transfer. Can be
  - a for an ASCII transfer, or
  - b for a binary transfer.
- **Flags:** one or more single-character flags indicating any special action taken. Can be one or several of
  - C file was compressed,
  - U file was uncompressed,
  - T file was tar'ed, or
  - _ no action was taken
- **Direction:** the direction of the transfer. Can be either
  - o outgoing, or
  - i incoming.
- **AccessMode:** the method by which the user is logged in:
  - a (anonymous) for an anonymous guest user,
  - g (guest) for an pass-worded guest user, or
  - r (real) for a local authenticated user.

- **Username:** the local username, or if guest, the ID string given.
- **ServiceName:** the name of the service being invoked, usually FTP.
- **AuthenticationMethod:** the method of authentication used, either

  0 none, or

  1 RFC931 authentication.
- **AuthenticatedUserId:** the user id returned by the authentication method. A * is used if no authenticated user id is available.
- **CompletionStatus:** a single character indicating the status of the transfer:

  c complete transfer, or

  i incomplete transfer.

Here is the derived E-R diagram:

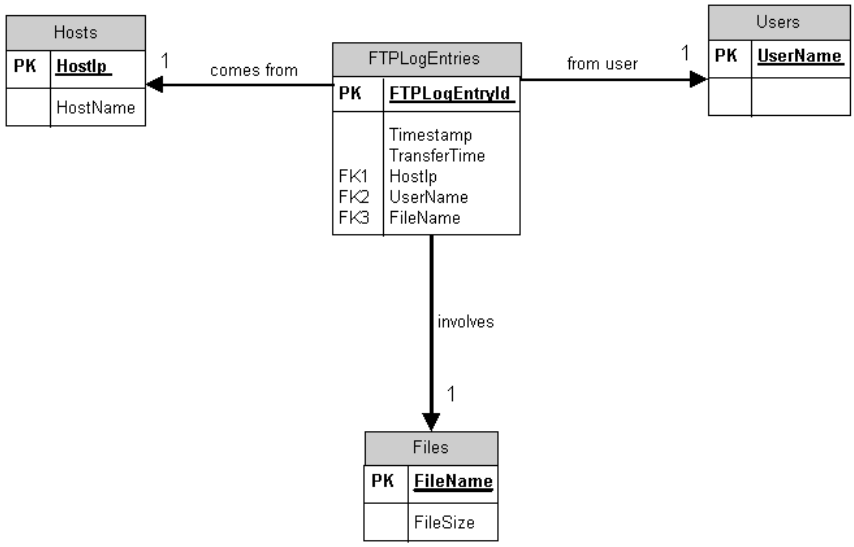

Figure 13: E-R Diagram for FTP logs

4.6 DNS Logs

For this kind of logs we will take the log file created by the BIND version 8 [47] server which is the most widely used DNS [46] server, as model. Here is an example:

10-Apr-2000 00:01:20.307 XX /10.2.3.4/1.2.3.in-addr.arpa/SOA/IN

10-Apr-2000 00:01:20.308 XX+/10.4.3.2/host.foo.com/A/IN

A typical entry has the following fields:

- **Timestamp:** A temporal reference when the entry was logged.
- **Recursive:** Whether the request was recursive: + means recursive, a blank means non-recursive.
- **RequestingHost:** The IP address of the host, which sent the request
- **RequestedName:** The name to be resolved
- **Type:** The type of request, such as SOA, IN, PTR, MX etc.
- **Protocol:** The protocol being requested. Nearly always, this is IN.

This is the E-R diagram:

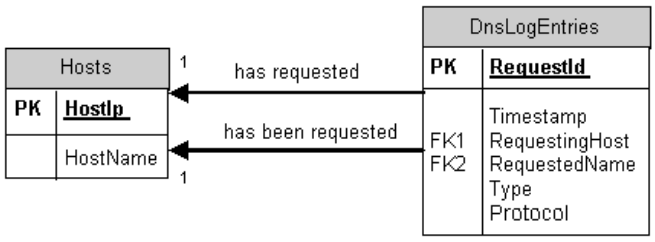

Figure 14: E-R Diagram for DNS logs

4.7 NET Logs

We have already mentioned that in order to model users/hosts behavior we will also look at logs that are not specifically related to a particular protocol but that

can be used to complement protocol-specific log information. Again the set of available formats is quite big but we will show that our requirements are sufficiently weak to assume that each such log provides the information we need.

4.7.1 TcpDump Logs

Tcpdump allows you to dump the traffic on a network. It can be used to print out the headers of packets on a network interface that matches a given expression[48]. Although it is mainly intended for Unix systems, it has been ported on almost every platform.

The general format of a tcpdump log line is

Timestamp > src > dst: flags data-seqno ack window urgent options

Src and dst are the source and destination IP addresses and ports. Flags are some combination of S (SYN), F (FIN), P (PUSH) or R (RST) or a single "." (no flags). Data-seqno describes the portion of sequence space covered by the data in this packet (see example below). Ack is sequence number of the next data expected from the other direction on this connection. Window is the number of bytes of available receive buffer space in the other direction on this connection. Urg indicates that the packet contains "urgent" data. Options are tcp options enclosed in angle brackets (e.g., <mss 1024>). Src, dst and flags are always present. The other fields depend on the contents of the packet's tcp protocol header, and are output only if appropriate.

Here is the opening portion of an rlogin from host rtsg to host csam extracted from an example reported in the tcpdump man page:

```
999598392.567349  >  rtsg.1023 > csam.login: S 768512:768512(0) win 4096 <mss 1024>
999598392.726146  <  csam.login > rtsg.1023: S 947648:947648(0) ack 768513 win 4096 <mss 1024>
999598392.730587  >  rtsg.1023 > csam.login: . ack 1 win 4096
```

```
999598392.731026  >  rtsg.1023 > csam.login: P 1:2(1) ack 1 win 4096
999598392.908579  <  csam.login > rtsg.1023: . ack 2 win 4096
999598392.927531  >  rtsg.1023 > csam.login: P 2:21(19) ack 1 win 4096
999598392.958701  <  csam.login > rtsg.1023: P 1:2(1) ack 21 win 4077
999598392.955132  <  csam.login > rtsg.1023: P 2:3(1) ack 21 win 4077 urg 1
999598393.012867  <  csam.login > rtsg.1023: P 3:4(1) ack 21 win 4077 urg 1
```

The first line says that tcp port 1023 on rtsg sent a packet to port login on csam. The S indicates that the SYN flag was set. The packet sequence number was 768512 and it contained no data (the notation is "first : last (nbytes)" which means "sequence numbers first up to but not including last which is nbytes bytes of user data"). There was no piggybacked ack, the available receive window was 4096 bytes, and there was a max-segment-size option requesting an mss of 1024 bytes. Csam replies with a similar packet except that it includes a piggybacked ack for rtsg's SYN. Rtsg then acks csam's SYN. The "." means that no flags were set. The packet contained no data so there is no data sequence number. The first time tcpdump sees a tcp "conversation", it prints the sequence number of the packet. On subsequent packets of the conversation, the difference between the current packet's sequence number and this initial sequence number is printed. This means that all sequence numbers after the first one can be interpreted as relative byte positions in the conversation's data stream (with the first data byte in each direction being "1"). On the 6th line, rtsg sends csam 19 bytes of data (bytes 2 through 20 in the rtsg -> csam side of the conversation). The PUSH flag is set in the packet. On the 7th line, csam says it has received data sent by rtsg up to but not including byte 21. Most of this data is apparently sitting in the socket buffer because the size of csam's receive window has decreased by 19 bytes. Csam also sends one byte of data to rtsg in this packet. On the 8th and 9th lines, csam sends two bytes of urgent, pushed data to rtsg.

4.7.2 SOCKS Logs

SOCKS [23] [24] is a networking proxy protocol that enables hosts on one side of a SOCKS server to gain full access to hosts on the other side of the SOCKS server without requiring direct IP reachability. A SOCKS redirects connection requests from hosts on opposite sides of a SOCKS server. The SOCKS server authenticates and authorizes the requests, establishes a proxy connection, and relays data. SOCKS is commonly used as a network firewall that enables hosts behind a SOCKS server to gain full access to the Internet, while preventing unauthorized access from the Internet to the internal hosts. There are two main versions of SOCKS: SOCKS V4 and SOCKS V5. Also in this case, without standards, we need to study some specific implementations of the protocol. In particular we will analyze the log files generated by the NetProxy version 3.0 server [35]. NetProxy is a proxy server and firewall system for Windows 95, Windows 98 and Windows NT. It allows many users to access the Internet simultaneously using a single connection of almost any kind (modem, ISDN, cable modem, leased line, etc.). NetProxy provides several proxy services, as well as a configurable firewall, access logging, and Socks (both version 4 and 5) service.

SOCKS 4

This is an entry that indicates SOCKS v4.0 traffic:

Timestamp	OriginIp	OriginPort	DestinationIp	DestinationPort	UserName
31/Aug/1997:12:09:48 +0100	10.84.128.147	60403	208.48.218.33	80	fred

If no username was specified, the field contains a single hyphen character (-).

SOCKS 5

This is an entry that indicates SOCKS v5.0 traffic. The format is exactly the same as the SOCKS 4 entry, except for the fact the text "UDP" may appear at the end of the line to indicate an UDP port association rather than a mapped TCP port.

Timestamp	OriginIp	OriginPort	DestinationIp	DestinationPort	UserName	Protocol
31/Aug/1997:12:09:48 +0100	10.84.128.147	60403	208.48.218.33	80	fred	UDP

where

- **Timestamp**: when the entry was logged
- **OriginIP**: the IP address of the host that started the session
- **OriginPort**:: the port number used by the application that started the session
- **DestinationIP**: The IP address of the destination host
- **DestinationPort**: The port number on which the destination application will receive data
- **UserName**: The name of the user that made the request
- **Protocol:** The protocol used (only for SOCKS version 5 traffic)

4.7.3 E-R Diagram for NET Logs

Although the information provided by Net logs can be very specific and detailed we will continue our attempt to find a subset that can reasonably be assumed to be as provided by all these log types.

Here is the proposed E-R schema:

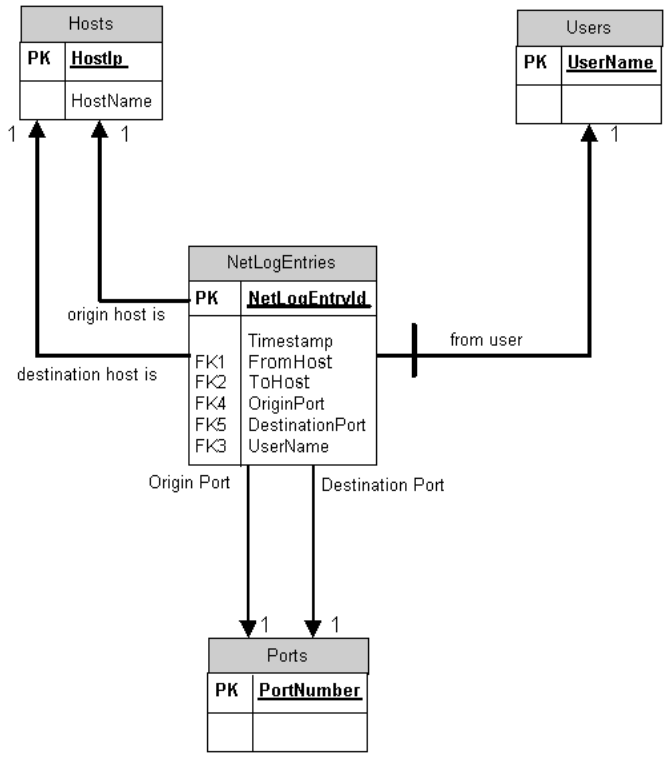

Figure 15: E-R Diagram for Net logs

To be as general as possible we merely require the following:

- **Timestamp:** a temporal reference.
- **FromHost:** the IP address or DNS name of the host that sent the packet.
- **FromPort:** the origin port from which the packet was sent.
- **ToHost:** the IP address or DNS name of the destination host.
- **ToPort:** the destination port to which the packet has been.
- **Volume:** the number of bytes sent.
- **UserName:** the name of the user that started the session (optional: only for socks entries).

More details about the implementation will be given in Chapter 4.

4.8 Activity-Entity Synthesis

We conclude the chapter with the complete E-R diagram that results by merging all the "individual E-R diagrams" presented in preceding paragraphs.

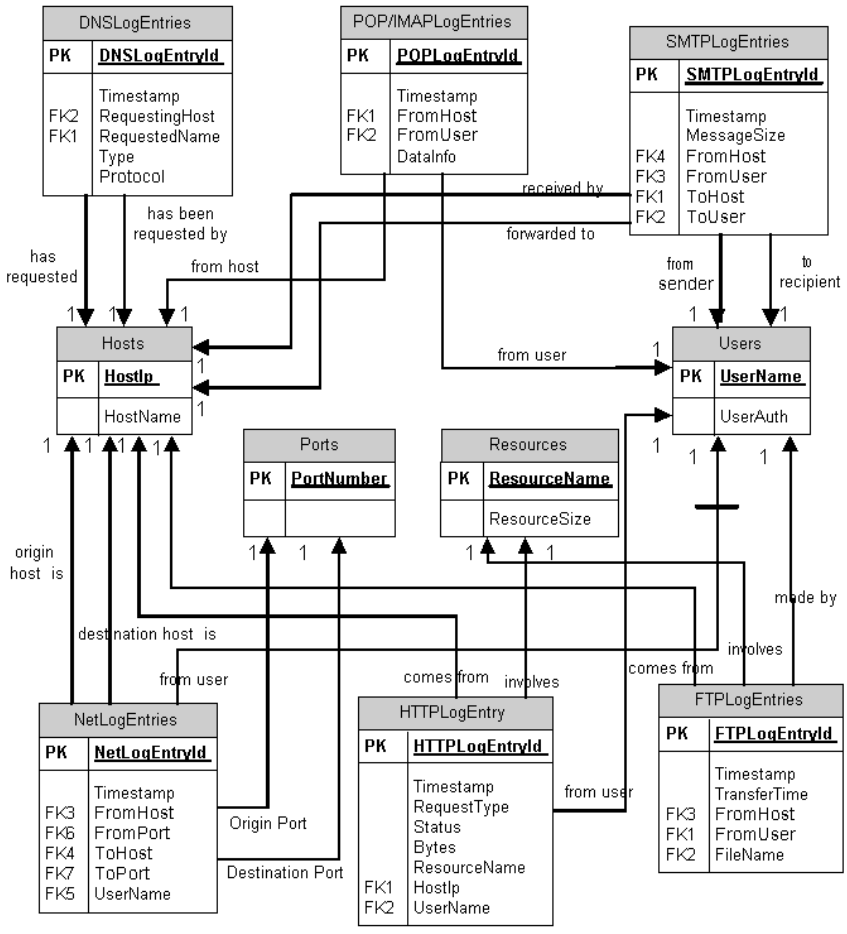

Figure 16: A Complete Picture

The main intent of this E-R diagram is to have an overall picture that summarizes everything we said during the analysis made in preceding paragraphs. This picture shows on the upper and lower borders all the various sources of information that we assume to have used during the entire process (except for FTP and DNS

logs). In the center of the diagram we can see the main entities involved in this process (hosts and people), and further down, other "observable" entities (ports and files) that we have reported for completeness, but that will be not considered in detail from now on.

4.9 Summary

In this chapter we have first described the services that we have chosen to analyze in our study. Although these protocols do not cover the complete network activity, they represent a large slice of the whole. Then we have analyzed several logs from servers that implement these protocols. We showed that, except for the http protocol, the format of these logs has not been standardized. However although the content of these logs varies, the semantic of the information is almost the same for the various implementations of the same protocol. For each protocol analyzed we presented an E-R diagram pointing out relationships between the entities involved in such a protocol. Finally, we concluded the chapter with a diagram resulting from the merging of all these "single-service diagrams", to provide a complete picture of the information that we can assume to be available henceforth.

Chapter 5

DATA PARSING AND AGGREGATION

In this chapter we will describe in a more details how the Data Parsing and Aggregation phase has been implemented. The main goal of this phase is to obtain a consistent repository in which data coming from several different logs will be stored according to our proposed integrated logic scheme. This is needed mainly because our data sources have too many entries so that the data cannot be easily used unless aggregated. Consider that the fact that for a week of logged traffic (which we consider to be the minimum period that has to be analyzed in order to obtain significant data) would lead to at least 100 millions log entries on a medium-sized network like the one that is object of our study. Here is our Usage schema:

ID	Category	RecordType	StartTime	EndTime	InitiatingUser	InitiatingHost	TargetUser	TargetHost

GlobRef	LocRef	Description	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev

Table 1: The Usage Schema

The semantic of some fields is slightly different depending on the RecordType value, and will therefore be explained separately in next few subsections; we can however observe that in general a usage record potentially has a StartTime and an EndTime, a initiating user and/or host as well as a target user and/or host plus some information (Tmin, Tmax, Tavg, TstdDev) about the distribution over the time of log entries used to generate the usage record.

5.1 Web Log File

Let us start with the web log file. In this case our approach consists of aggregating all entries that refer to web requests coming from the same host within a certain time in one individual usage record. Below we show an example of such an aggregation.

Example 4.1.1

These entries were logged when host 131.114.4.6x accessed the index.html page of the Computer Science Department web site of the University of Pisa:

131.114.4.6x - - [25/Aug/2001:22:54:16 +0200] "GET / HTTP/1.0" 200 4234 "-" "Mozilla/4.71 [en] (WinNT; I)"

131.114.4.6x - - [25/Aug/2001:22:54:16 +0200] "GET /images/header.gif HTTP/1.0" 200 9342 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

131.114.4.6x - - [25/Aug/2001:22:54:16 +0200] "GET /images/dot.gif HTTP/1.0" 200 8765 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

131.114.4.6x - - [25/Aug/2001:22:54:17 +0200] "GET /foto/iris.jpg HTTP/1.0" 200 7104 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

131.114.4.6x - - [25/Aug/2001:22:54:17 +0200] "GET /foto/palloncini1.jpg HTTP/1.0" 200 4549 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

131.114.4.6x - - [25/Aug/2001:22:54:17 +0200] "GET /foto/cortile.jpg HTTP/1.0" 200 8483 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

131.114.4.6x - - [25/Aug/2001:**22:54:17** +0200] "GET /foto/ingresso2.jpg HTTP/1.0" 200 3224 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

.

. **Assuming that host 131.114.4.6x generates no other log entries for**

. **more than one minute**

.

131.114.4.6x - - [25/Aug/2001:**22:55:20** +0200] "GET ~prof1/index.html HTTP/1.0" 200 3224 "http://www.di.unipi.it/" "Mozilla/4.71 [en] (WinNT; I)"

The HTTP server logs a separate entry for each file requested from the client browser. Therefore, a user's request to view a particular page (the index.html page in this example) often results in several log entries because graphics and scripts are downloaded in addition to the HTML file (next 6 entries in this example)

without an explicit user request. Here is the corresponding usage record created and stored into the database:

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser	TgtHost
1	Web Log	Http Session	2001-08-25 22:54:16	2001-08-25 22:55:20		131.114.4.6x		131.114.4.11

GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
		Mozilla/4.71 [en] (WinNT; I)	7	45701	0	1000	167	372

where the meaning of the fields is

ID: A unique usage record identifier.

Category: The log from which the usage record has been generated. In this case "Web Log".

RecordType: The type of the record. In the case of web log files its value is always "Http Session".

StartTime: Timestamp of the first web log entry that belongs to this usage record.

EndTime: Timestamp of the last web log entry that belongs to this usage record.

InitiatingUser: This field should contain the user credentials but it is almost always empty (unless authentication is required to access a particular web page.

InitiatingHost: The IP address of the host that made the request.

TargetUser: Always empty.

TargetHost: The IP address of the web server.

GlobalRef: Always empty.

LocalRef: Always empty.

Description: Information about the user agent (Netscape Navigator, Internet Explorer, etc.) as well as about the operating system of the host.

DataPackets: The total number of web log entries that have contributed to create this usage record.

DataVolume: The sum of all bytes of each log entry that belongs to this usage record.

The ideal situation would correspond to have a usage record for each page explicitly requested by the user, aggregating all those entries that are generated

automatically because of HTML tags. This can be done with some degree of approximation using the *Referrer* field. However this computation requires an overhead that is not justifiable in our situation, because we are not doing an ad hoc web site study that requires such fine a granularity, which in any case would be lost when, in the next phase, several usage records will be collapsed into one activity[6][8][10].

5.2 SMTP Log File

The SMTP log file is the one that has been the least filtered of the ones we parsed. This is due to the fact that, as we showed in Section 3.3, in general there is a relation 1 to *n* that relates the sender to the *n* recipients of an email. We would like to retain this information without complicating the database logic schema, meaning that at this point we do not want to add new tables to model the relation. Therefore we just filtered any data we do not need from each log entry without collapsing more entries into one usage record.

Example 4.2.1

Let us have a look at SMTP log entries generated when user **Rossi@di.unipi.it** sends an email to local user **Verdi@di.unipi.it** and to remote user **Bianchi@informatik.uni-freiburg.de**. These are the log entries generated:

> Jun 18 09:26:37 apis sendmail[30933]: JAA14975: from=<rossi@di.unipi.it>, size=1038, class=0, pri=61038, nrcpts=2, msgid=<005101c0f7ee$54e36640$5b027283@kdd>, proto=SMTP, relay=pc-rossi [131.114.2.91]
>
> Jun 18 09:27:06 apis sendmail[30934]: JAA14975: to=<verdi@di.unipi.it>, ctladdr=<rossi@di.unipi.it> (15124/110), delay=00:00:29, xdelay=00:00:00, mailer=local, stat=Sent
>
> Jun 18 09:27:06 apis sendmail[30934]: JAA14975: to=<bianchi@informatik.uni-freiburg.de>, ctladdr=<rossi@di.unipi.it> (15124/110), delay=00:00:29, xdelay=00:00:28, mailer=esmtp, relay=mailgateway1.uni-freiburg.de. [132.230.1.211], stat=Sent (OK id=15BxcV-0003Xy-00

Here are the corresponding usage records created

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser
1	Smtp Log	Message Sending	2001-06-18 09:26:37		Rossi@di.unipi.it	131.114.4.xxx	
2	Smtp Log	LclForwarding	2001-06-18 09:27:06	2001-06-18 09:27:06			Verdi@di.unipi.it
3	Smtp Log	RmtForwarding	2001-06-18 09:27:06	2001-06-18 09:27:xx			Bianchi@informatik.uni-freiburg.de

TgtHost	GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
	Msg1@di.unipi.it	JAA14975		1	1038				
		JAA14975		1					
mailgateway1.uni-freiburg.de		JAA14975		1					

The meaning of each field is the following:

ID: A unique usage record identifier.

Category: Always "Smtp Log".

RecordType: The type of the record. It can assume one of these four values: "MsgSending", "MsgReceiving", "LclForwarding", "RmtForwarding":

- *MsgSending* refers to emails sent by local users,
- *MsgReceiving* refers to emails sent by remote users,
- *LclForwarding* refers to the act of forwarding an email to a local user, and
- *RmtForwarding* refers to the act of forwarding an email, sent by a local user, to a remote SMTP server.

StartTime: Timestamp that records when the corresponding log entry has been logged.

EndTime: Empty for "MsgSending" and "MsgReceiving" records, StartTime + xdelay for the others.

InitiatingUser: The sender's name (empty for "LclForwarding" and "RmtForwarding" records).

InitiatingHost: The IP address of the host that made the request.

TargetUser: The recipient's name (unfilled for "MsgSending" and "MsgReceiving" records).

TargetHost: The IP address of the host to which the message was forwarded (the remote SMTP server for outgoing messages; the host holding the recipient's mailbox for the others).

GlobalRef: The global message id (see Section 3.3.1).

LocalRef: An id that states that the three entries refer to the same email-sending activity.

Description: Always unfilled.

DataPackets: Always equal to 1.

DataVolume: The size of the email. Empty for "LclForwarding" and "RmtForwarding" records.

Tmin, Tmax, Tavg, TstdDev: Always empty.

The example above describes the case in which the message is sent from inside the network by a local user. In this case the InitiatingHost field refers to the host from which the Smtp server received the message. Most of the time this host is the host from which the sender has sent the email, but there are some cases (depending on how the network/email-service is configured) in which the semantic of this field is different. For example in a situation like that showed Figure 17 below where the SMTP server is in a DMZ and the firewall acts as IP masquerador.

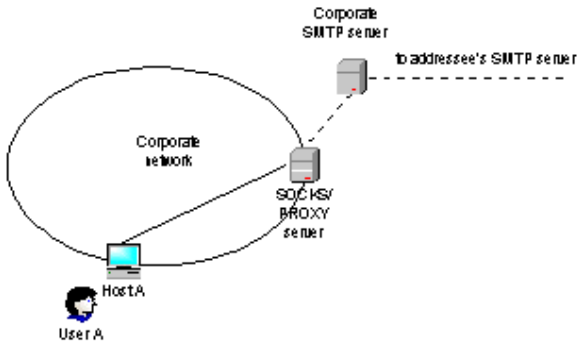

Figure 17: SMTP Server in DMZ

Here there is no way for the SMTP server to know the IP address of the host from which the email was sent because it is hidden by the firewall. In such a case the usage records created would be the following:

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser
1	Smtp Log	Message Sending	2001-06-18 09:26:37		Rossi@di.unipi.it	**net-gw.di.unipi.it**	
2	Smtp Log	LclForwarding	2001-06-18 09:27:06	2001-06-18 09:27:06			Verdi@di.unipi.it
3	Smtp Log	RmtForwarding	2001-06-18 09:27:06	2001-06-18 09:27:xx			Bianchi@informatik.uni-freiburg.de

TgtHost	GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
	Msg1@di.unipi.it	JAA14975		1	1038				
		JAA14975		1					
mailgateway1.uni-freiburg.de		JAA14975		1					

As we can see, now the InitiatingHost field refers to the firewall host. Still another situation occurs if the message is sent by a remote user. Let us imagine that the same email was sent by the remote user Bianchi@informatik.uni-freiburg.de to the local user Rossi@di.unipi.it. The corresponding usage records created would be the following:

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser
1	Smtp Log	**MsgReceiving**	2001-06-18 09:26:37		Bianchi@informatik.uni-freiburg.de	**mailgateway1.uni-freiburg.de**	
2	Smtp Log	LclForwarding	2001-06-18 09:27:06	2001-06-18 09:27:06			Rossi@di.unipi.it

TgtHost	GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
	Msg1@di.unipi.it	JAA14975		1	1038				
		JAA14975		1					

Now the InitiatingHost field has another meaning: it refers to the host that has forwarded the email to the local SMTP server (most of the time the remote user's SMTP server).

All these differences must be taken into account when usage records will be used for activities modeling, but this will be discussed later, in the next chapter.

5.3 Pop Log File

As shown in Section 3.4 the information that can reasonably assumed to be available in this kind of logs includes the user name, the host from which the pop session has been executed, and some information about the emails downloaded (number of emails downloaded or number of bytes moved, etc.). Our approach consists of creating one usage record for each email-downloading attempt. This choice allows us to not lose any data in passing from log entries to usage records and at the same time it allows us to reduce the size of data set normally by a factor 3 although this source of data does not present particular problems in terms of compression as it generally represents only a small slice of the total. This means that in a normal situation one usage record is created for every three log entries.

The next example shows what are the log entries logged and the corresponding usage record created when user Rossi@di.unipi.it starts a Pop session from his host pc-rossi@di.unipi.it (131.114.2.9x).

Example 4.3.1

User Rossi starts a Pop session from his host pc-rossi.di.unipi.it to download any emails received since the last Pop session. These are the log entries logged:

```
Jun 18 09:26:49 apis ipop3d[733352]: pop3 service init from 131.114.2.9x
Jun 18 09:26:50 apis ipop3d[733352]: Auth user=Rossi host=pc-rossi.di.unipi.it [131.114.2.9x]
nmsgs=32/32
Jun 18 09:26:51 apis ipop3d[733352]: Logout user=Rossi host=pc-rossi.di.unipi.it [131.114.2.9x]
nmsgs=27 ndele=5
```

The first line is logged when the service request arrives at the pop server, indicating that host 131.114.2.9x has started a Pop session. The second line is logged when the Pop client authenticates itself sending user credentials. In this line we can find information about the user and about new messages in the form

of new-messages-received/total-messages-in-mailbox. Finally, the third line is logged when the client closes the session. This line reports (among the other things) how many messages were left in the mailbox and how many were deleted. As you can see these three entries are related by the [733352] field, which is the number of the thread that served the session. In practice for each session started the pop server creates a new thread that serves the session and then expires. Although this number is not unique, it can be supposed to be in a quite short interval and therefore has been used to merge information coming from different log entries of the same session into the following *Email Downloading* usage record:

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser
1	Pop Log	Pop Session	2001-06-18 09:26:49	2001-06-18 09:26:51	Rossi@di.unipi.it	131.114.2.9x	

TgtHost	GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
131.114.4.6				27	32	1000	1000	1000	0

The meaning of each field is the following:

ID: A unique usage record identifier.

RecordType: Always "POP Session".

StartTime: Timestamp of the first POP log entry that belongs to the corresponding pop session.

EndTime: Timestamp of the last POP log entry that belongs to the corresponding pop session.

InitiatingUser: Name of the user whose credentials were used to validate session.

InitiatingHost: The IP address of the host that started the POP session.

TargetUser: Always unfilled.

TargetHost: The POP server IP address.

GlobalRef: Always empty.

LocalRef: Always empty.

Description: Always empty.

DataPackets: Number of messages left in the mailbox.

DataVolume: Number of messages downloaded.

Again, the Tmin, Tmax, Tavg and TstdDev fields contain information about the time distribution of the log entries that generated this usage record.

5.4 Net Log File

The largest compression effort is needed for NET logs. This is due to the fact that NET logs are by far the largest source of information we have. Therefore we need some criteria that allow us to appreciably reduce the number of usage records to be created but that at the same time prevent the loss of too much information in passing from log entries to database records. The first idea was to try to aggregate all log entries that refer to the same tcp connection as a unique usage record, but then we realized that this was not sufficient in the sense that the compression factor was still not sufficiently high and resulted in a number of records still of the order of millions. In particular most of them (about 95%) were http connections, thus the next step for this kind of connections was to collapse more than one connection into one usage record. We will go into more details in next few subsections.

5.4.1 SMTP and POP/IMAP Traffic

For Net log entries that refer to SMTP and POP/IMAP traffic, the aggregation algorithm works in the same way. It makes use of a hash table where one bucket is reserved for each SMTP or POP/IMAP connection open at a certain time. The key that identifies the connection consists of *ClientIP.ClientPort.ServerIP.ServerPort,* where ServerPort has to be either 25 (default SMTP service port) or 110/143 (default POP/IMAP service port). For each net log entry the algorithm simply checks to which connections it belongs, updating its fields such as end time, data volume, number of packets and so on. If the net log entry refers to the first

packet of a new connection, a new bucket is created. Each connection is considered closed (and correspondingly a new usage record is created) when the time elapsed from last packet received is greater than a fixed (GAP) parameter. Tests with different GAP parameters showed that if we choose a sufficiently high GAP (e.g. 1 min) we can be reasonably sure that all packets belonging to one such connection will be part of the same usage record. This is because normally clients who eventually close the current and open a new connection with the same server before the GAP has elapsed, use a different port, allowing us to distinguish the current connection from the previous one.

Example 5.4.1.1

This example shows a situation in which the local host 131.114.2.11x starts a POP session with an Internet POP server [217.58.130.18].

Here are the corresponding entries logged:

999598392.171337 > 131.114.2.11x.45316 > 217.58.130.18.pop3: S 3331168056:3331168056(0) win 5840 <mss 1460,sackOK,timestamp 364005685 0,nop,wscale 0> (DF)
999598392.338684 < 217.58.130.18.pop3 > 131.114.2.11x.45316: S 3319002880:3319002880(0) ack 3331168057 win 32120 <mss 1460,nop,nop,sackOK,nop,wscale 0> (DF)
999598392.339018 > 131.114.2.11x.45316 > 217.58.130.18.pop3: . 1:1(0) ack 1 win 5840 (DF)
999598392.562244 < 217.58.130.18.pop3 > 131.114.2.11x.45316: P 1:42(41) ack 1 win 32120 (DF)
999598392.562617 > 131.114.2.11x.45316 > 217.58.130.18.pop3: . 1:1(0) ack 42 win 5840 (DF)
999598392.567349 > 131.114.2.11x.45316 > 217.58.130.18.pop3: P 1:7(6) ack 42 win 5840 (DF)
999598392.726146 < 217.58.130.18.pop3 > 131.114.2.11x.45316: . 42:42(0) ack 7 win 32120 (DF)
999598392.730587 < 217.58.130.18.pop3 > 131.114.2.11x.45316: P 42:94(52) ack 7 win 32120 (DF)
999598392.731026 > 131.114.2.11x.45316 > 217.58.130.18.pop3: P 7:19(12) ack 94 win 5840 (DF)
999598392.908579 < 217.58.130.18.pop3 > 131.114.2.11x.45316: P 94:114(20) ack 19 win 32120 (DF)
999598392.940426 > 131.114.2.11x.45316 > 217.58.130.18.pop3: P 19:33(14) ack 114 win 5840 (DF)
999598393.107232 < 217.58.130.18.pop3 > 131.114.2.11x.45316: P 114:130(16) ack 33 win 32120 (DF)
999598393.108099 > 131.114.2.11x.45316 > 217.58.130.18.pop3: P 33:43(10) ack 130 win 5840 (DF)
999598393.296012 < 217.58.130.18.pop3 > 131.114.2.11x.45316: . 130:130(0) ack 43 win 32120 (DF)
999598399.074977 < 217.58.130.18.pop3 > 131.114.2.11x.45316: P 130:160(30) ack 43 win 32120 (DF)
999598399.075676 > 131.114.2.11x.45316 > 217.58.130.18.pop3: P 43:49(6) ack 160 win 5840 (DF)

999598399.247457 < 217.58.130.18.pop3 > 131.114.2.11x.45316: P 160:169(9) ack 49 win 32120 (DF)
999598399.247994 > 131.114.2.11x.45316 > 217.58.130.18.pop3: P 49:55(6) ack 169 win 5840 (DF)
999598399.424459 < 217.58.130.18.pop3 > 131.114.2.11x.45316: . 169:169(0) ack 55 win 32120 (DF)
999598401.423016 < 217.58.130.18.pop3 > 131.114.2.11x.45316: P 169:183(14) ack 55 win 32120 (DF)
999598401.425679 > 131.114.2.11x.45316 > 217.58.130.18.pop3: F 55:55(0) ack 183 win 5840 (DF)
999598401.429685 < 217.58.130.18.pop3 > 131.114.2.11x.45316: F 183:183(0) ack 55 win 32120 (DF)
999598401.430058 > 131.114.2.11x.45316 > 217.58.130.18.pop3: . 56:56(0) ack 184 win 5840 (DF)
999598401.614889 < 217.58.130.18.pop3 > 131.114.2.11x.**45316**: . 184:184(0) ack 56 win 32120 (DF)

No Packets between 131.114.2.11x and 217.58.130.18 for 20 seconds

999598421.515854 > 131.114.2.11x.**45320** > 217.58.130.18.pop3: S 3369225773:3369225773(0) win 5840 <mss 1460,sackOK,timestamp 364008620 0,nop,wscale 0> (DF)

As you can see the client 131.114.2.11x starts a tcp connection on port **45316** with the pop server 217.58.130.18 (port 110 is the default Pop service port). On this connection several packets are sent and received by both hosts with a maximum delay of less than 6 seconds between two successive packets. When GAP has elapsed since the last packet received, the connection bucket will be removed and the corresponding usage record stored into the database. This is the usage record resulting from collapsing all previously shown log entries:

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser	TgtHost
1	Net Log	Pop Session	2001-09-04 12:13:12	2001-09-04 12:13:21		131.114.2.11x		217.58.130.18

GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
		2	24	236	0	5778	410	1212

The meaning of each field is the following:

StartTime: The timestamp of the first packet of the connection.

EndTime: The timestamp of the last packet of the connection.

Description: This field can be either "One" or "Two". "One" means that data have been sent only in one direction, whereas "Two" means that data have been sent in both directions.

DataPackets: Number of packets (e.g. net log entries) that have contributed to the usage record.

DataVolume: Sum of bytes exchanged between the two hosts during the session.

The Description value allows us to discriminate between connections in which client and server have really interacted by exchanging data and connections in which data have been sent only in one direction (an example of such bad data are packets sent for port scanning). The example also shows that when the client starts a new session with the same pop server, it uses a new, different port (45320 vs. 45316). This will result in a new bucket being created into the hashtable, and consequently in a new usage record stored into the database.

5.4.2 HTTP traffic

The situation with HTTP traffic differs somewhat. For this kind of traffic, connection level aggregation is not sufficient because the total number of usage records created would be too high. We must collapse more than one connection into one usage record. The algorithm behavior is very similar to the previous one, with the difference that now all simultaneous connections between two hosts are aggregated. Http protocol [25] allows clients to have multiple connections open at the same time. This means that normally when a user surfing the web downloads a web page, more than one connection are open between the browser and the web server in order to download all the resources composing the page. This behavior is slightly limited with http version 1.1, where connection persistence allows web browsers to send multiple requests in pipeline on the same connection so that the number of simultaneous connections usually is not more than two.

Example 5.4.2.1

This example shows a situation in which local host 131.114.4.17x starts an http session with a web server running on remote host 212.48.9.22. The NET log

entries below show that the client opens two simultaneous connections with the web server, the first on port 2099, the second on port 2100.

999597426.543181 > 131.114.4.17x.2099 > 212.48.9.22.www: S 2586282406:2586282406(0) win 16384 <mss 1460,nop, nop, sackOK> (DF)

999597426.729384 < 212.48.9.22.www > 131.114.4.17x.2099: S 1463449978:1463449978(0) ack 2586282407 win 8760 <nop,nop,sackOK,mss 1460> (DF)

999597426.729871 > 131.114.4.17x.2099 > 212.48.9.22.www: . 1:1(0) ack 1 win 17520 (DF)

999597427.952043 > 131.114.4.17x.2099 > 212.48.9.22.www: P 1:325(324) ack 1 win 17520 (DF)

999597427.955892 > 131.114.4.17x.2100 > 212.48.9.22.www: S 2586682374:2586682374(0) win 16384 <mss 1460,nop,nop,sackOK> (DF)

999597428.162305 < 212.48.9.22.www > 131.114.4.17x.2099: . 1:1(0) ack 325 win 8760 (DF)

999597428.164911 < 212.48.9.22.www > 131.114.4.17x.2100: S 3857102380:3857102380(0) ack 2586682375 win 8760 <nop,nop,sackOK,mss 1460> (DF)

999597428.166708 > 131.114.4.17x.2100 > 212.48.9.22.www: . 1:1(0) ack 1 win 17520 (DF)

999597428.166737 < 212.48.9.22.www > 131.114.4.17x.2099: P 1:134(133) ack 325 win 8760 (DF)

999597428.167321 < 212.48.9.22.www > 131.114.4.17x.2099: P 134:559(425) ack 325 win 8760 (DF)

999597428.168232 > 131.114.4.17x.2100 > 212.48.9.22.www: P 1:329(328) ack 1 win 17520 (DF)

999597428.168286 > 131.114.4.17x.2099 > 212.48.9.22.www: . 325:325(0) ack 559 win 16962 (DF)

999597428.375082 < 212.48.9.22.www > 131.114.4.17x.2100: . 1:1(0) ack 329 win 8760 (DF)

999597428.390022 < 212.48.9.22.www > 131.114.4.17x.2100: P 1:134(133) ack 329 win 8760 (DF)

999597428.390564 < 212.48.9.22.www > 131.114.4.17x.2100: F 523:523(0) ack 329 win 8760 (DF)

999597428.390693 < 212.48.9.22.www > 131.114.4.17x.2100: P 134:523(389) ack 329 win 8760 (DF)

999597428.391304 > 131.114.4.17x.2100 > 212.48.9.22.www: . 329:329(0) ack 134 win 17387 <nop,nop, sack 1 {523:524} > (DF)

999597428.391436 > 131.114.4.17x.2100 > 212.48.9.22.www: . 329:329(0) ack 524 win 16998 <nop,nop, sack 1 {523:524} > (DF)

999597433.786891 > 131.114.4.17x.2100 > 212.48.9.22.www: F 329:329(0) ack 524 win 16998 (DF)

999597434.028688 < 212.48.9.22.www > 131.114.4.17x.2100: R 3857102904:3857102904(0) win 0 (DF)

999597471.802370 > 131.114.4.17x.2099 > 212.48.9.22.www: R 2586282731:2586282731(0) win 0 (DF)

The algorithm will consider these two connections as only one fact, thus generating just one usage record:

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser	TgtHost
21	Net Log	Http Session	2001-09-04 11:57:06	2001-09-04 11:57:51		131.114.4.17x		212.48.9.22

GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
		2	21	1732	0	37784	2262	8232

The meaning of each field is the same as before.

For HTTP 1.0, the number of simultaneous connections is even greater so that one can have five or more simultaneous connections open between one client and one server. Moreover, some analysis of web log data showed that although HTTP 1.0 is older it is still used at least as much as 1.1 is. We have calculated that on average about 4 connections have been collapsed into one usage record.

5.5 Summary

In this chapter we have shown how the Data Parsing and Aggregation phase has been implemented. For each source of data we have presented several examples about how log entries have been parsed and collapsed into usage records. As already said, the greatest effort has been devoted to data size reduction. The Usage table resulting from this phase will be the starting point of the next phase described in the next chapter.

Chapter 6

COMPUTING AGGREGATED ACTIVITIES

In this chapter we will explain how aggregated activities have been built in our specific test case. We want to point out that this process of merging usage records into more meaningful Activities records is strongly dependent on how the network is structured. This means that different network topologies and different services configurations could lead to different activities or at least to different merging procedures. We will now focus on a particular case, the Computer Science departmental network of the University of Pisa. The topology of the Computer Science departmental network is shown below:

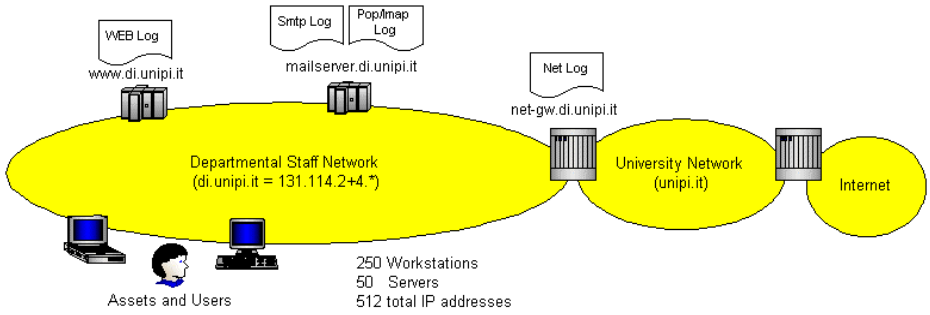

Figure 18: Actual Network at University of Pisa

As you can see the network has its own web and mail (SMTP, POP, IMAP) servers that can be accessed by any local host without any intermediation. All these entities are protected from the outside world by a firewall that represents the only way to cross the perimeter of the network: all incoming and outgoing traffic must pass through it. As shown in Figure 18, five different logs are supposed to be available: the internal public WEB server log, the POP and IMAP

servers logs for emails downloads, the SMTP server log for emails sending and receiving, and the NET-GATEWAY log for all the outgoing and incoming tcp traffic. Such a configuration, which is very common in the real world, allows us to use local server logs for tracking internal activities whereas the net-gateway (firewall) can be used for tracking all the activities that represent interactions with the external world.

6.1 Web Surfing Activities

At this point we will assume that all web and net log entries that refer to http traffic have already been parsed and aggregated into the corresponding *Http Session* usage records as described in Chapter 5. Now we want to collapse these different usage records into more meaningful units that we call *Local Web Surfing* activities. Here is the pseudocode of the algorithm used by the Activities Builder to model *Local Web Surfing* activities:

```
For each (LocalHost) {

    SELECT   *
    FROM  usage table
    WHERE RecordType = Http Session
    AND InitiatingHost = CrntHost
    ORDER BY StartTime

    CurrentActivity = empty activity

    For each of such usage records {
        if the time gap since previous is less than GAP {
            include it into the current activity
        } else {
            close current activity
            store it into the activities table
            start a new activity
        }
    }
}
```

A *Local Web Surfing* activity is obtained using the Web usage records (Category = *Web Log* AND RecordType = *Http Session*) for internal http traffic and the Net usage records (Category = *Net Log* AND RecordType = *Http Session*) for outgoing http traffic. For each local host the algorithm selects all *Http Session*-type records from the Usage table, and orders them according to the time they happened. Then it starts aggregating such records until the time gap between two successive usage facts is greater than a fixed GAP interval. Clearly the GAP setting reflects the degree of granularity of the resulting activities, and also influences the number of activities created: a GAP equal or close to zero would lead to one activity per usage record; a GAP of one week would lead to at most one activity per host. Of course interesting GAP values are between these two extremes. We obtained the best results in terms of number of Web Surfing activities created and activities duration for GAP values of approx. 1 hour.

We now try to show more clearly how a Web Surfing activity is built using an example.

Example 6.1.1

Let us imagine that professor Doe arrives in his office late in the morning and starts his web browser, which is configured in such a way that it downloads as first URL the main page of the local web server. Then it surfs the Sun's web site (192.18.97.241) for half an hour, and IBM's web site (129.42.18.99) for five minutes. Then he realizes that it is time to have lunch so he leaves his office for an hour and half. When he comes back, he starts surfing again. This situation results in these four usage records:

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser	TgtHost
21	Http Log	Http Session	2001-09-04 10:57:52	2001-09-04 10:57:55		131.114.4.xxx		131.114.4.11
22	Net Log	Http Session	2001-09-04 11:03:01	2001-09-04 11:04:20		131.114.4.xxx		192.18.97.241
23	Net Log	Http Session	2001-09-04 11:33:12	2001-09-04 11:33:20		131.114.4.xxx		129.42.18.99
24	Net Log	Http Session	2001-09-04 13:00:51	2001-09-04 13:01:27		131.114.4.xxx		216.239.35.100

GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
\Index.html	Http/1.1	Mozilla/4.71[en](WinNT;I)	9	16859	0	2000	375	695
		Two	156	64184	0	8102	117	659
		Two	81	26552	0	2423	97	314
		Two	77	27884	0	29781	476	3386

As already mentioned, starting from first usage record, the *Activities Builder* algorithm collapse into the same *Local Web Surfing* activity all those records for which the time elapsed between the EndTime of the current record and the StartTime of the successive record is less than the fixed one-hour-long GAP. In this example it first collapses three usage records into the same activity because when it checks the fourth it finds that the time elapsed between the third and the fourth record is one hour and half. Therefore it closes the current activity, stores it into the Activities table, and starts a new one with the fourth usage record. This is the resulting activity record:

ID	RecordType	StartTime	EndTime	InitUser	InitHost	Target(s)
55	Local Web Surfing	2001-09-04 10:57:52	2001-09-04 11:33:20		131.114.4.xxx	131.114.4.11||192.18.97.241||129.42.18.99

Description	NumUsages	Data1	Data2	Data3	Tmin	Tmax	Tavg	TstdDev
Mozilla/4.71[en](WinNT;I)	3	246	107595		306000	1732000	1019000	

The StartTime of the activity corresponds to the StartTime of the first usage record collapsed into this activity; the EndTime corresponds to the EndTime of the last usage record that belongs to this activity (the third in our example). The Target(s) field contains a concatenation of all target hosts surfed during the activity, and the Description field contains the web browser used or a concatenation of them if more than one web browser have been used. The latter case corresponds to a situation in which two or more different web browsers

have been used at the same time on the same host for web surfing. The NumUsages field shows how many usage records have been used for generating the activity, while data1 and data2 corresponds to the number of packets (log entries) that contributed for activity creation and the volume of traffic generated during the activity, respectively. Finally we have information on the distribution of usage records during the entire activity: Tmin and Tmax are the minimum and maximum gap, respectively, between two successive usage records, and Tavg and TstdDev are the average and the standard deviation of such a distribution, respectively.

The example above corresponds to a web-surfing activity generated by a local host. Another quite frequent situation is that often also some other local (e.g. intranet) web servers are installed apart from the official(s) one(s). The difference is that no logs are available from them because they are not managed by the network administrators staff, who may not even know that they exist. Such activities are therefore built by looking for valid Net-Http usage records whose TargetHost is a local host. This is the pseudo code:

```
For each (LocalHost) {

        SELECT   *
        FROM  usage table
        WHERE RecordType=Http Session
        AND TargetHost = CrntHost
       ORDER BY StartTime

        CurrentActivity = empty activity

        For each of such usage records {
              if the time gap since previous is less than GAP {
                    include it into the current activity
              } else {
                    close current activity
                    store it into the activities table
                    start a new activity
              }
         }
}
```

The result will be an activity record like the following:

ID	RecordType	StartTime	EndTime	InitUser	InitHost	Target(s)
55	Remote Web Surfing	2001-09-04 10:57:52	2001-09-04 11:33:20		192.16.6.3	131.114.4.xxx

Description	NumUsages	Data1	Data2	Data3	Tmin	Tmax	Tavg	TstdDev
	3	123	87632		306000	1732000	1019000	

Here no information about the browser used is available because only Net-Http usage records have been used to generate this activity. The record type now is *Remote Web Surfing*, meaning that a local host has been surfed by a remote one. The remaining fields have the same meaning as in the previous example.

A final remark regards the concept of valid usage records. We said in Chapter 4 that the usage table is mainly intended as a more flexible way to use log data. We also said that in passing from log files to usage records as little filtering as possible must be applied. This requirement implies that the usage table still contains noise and some not significant data that could yeld false results. Passing from usage to activities records we want to eliminate this noise. Therefore in this specific case, for example, we do not consider all those Net usage records that refer to *one-direction* traffic (maybe due to a Port scanning).

Example 6.1.2

If an external host sends a web packet on port 80 of a local host that does not run any web server or that does not even exist, this will result in a *Http Session*-type usage record where the local host is the TargetHost of the record, meaning that some kind of Http interaction between the two hosts has taken place.

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser	TgtHost
21	Net Log	Http Session	2001-09-04 10:57:52	2001-09-04 10:57:55		192.16.6.3		131.114.4.xxx

GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
		One	9	16859	0	2000	375	695

Table 2: Example of an invalid usage record

But this does not correspond to what actually happened: the external host tried to access the local host, the net gateway has logged this attempt but there was no interaction between external host and local host simply because no web server was running on the local host, and hence this record does not have to be taken into consideration when building the activity.

6.2 Email Downloading Activities

In this case the sources are the POP/IMAP and NET-POP/IMAP usage records. The first are used for *Local Email Downloading* activities meaning all those activities, started by users or by pop-client demons in order to download any emails eventually received from the local official POP/IMAP intranet server. NET-POP/IMAP usage records are used for *Remote Email Downloading* activities, meaning those activities that refer to attempts made by local users to download emails from an extranet mailbox. Again we will assume that all POP/IMAP log entries and all NET log entries that refer to POP/IMAP traffic have already been parsed and aggregated into the corresponding *Pop Session* usage records as described in Chapter 4.

6.2.1 Local Email Downloading

For *Local Email Downloading* activities we will use only POP/IMAP usage records (Category = *Pop Log* or Category = *Imap Log*). Although these records already provide information about the individual attempts to download emails from the official intranet POP/IMAP servers, they are too numerous and difficult to

manage because almost every POP/IMAP client is configured in such a way that it polls the server at regular intervals for new emails. We therefore need a reasonable way to collapse several of them into just one activity without losing important information. Below I reported some POP usage records showing individual interactions between user Verdi's POP client and the intranet POP server:

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser	TgtHost
21	Pop Log	Pop Session	2001-09-07 11:17:01	2001-09-07 11:17:07	Verdi	131.114.2.xxx		131.114.4.6
22	Pop Log	Pop Session	2001-09-07 11:26:56	2001-09-07 11:26:56	Verdi	131.114.2.xxx		131.114.4.6
23	Pop Log	Pop Session	2001-09-07 11:36:56	2001-09-07 11:36:56	Verdi	131.114.2.xxx		131.114.4.6
24	Pop Log	Pop Session	2001-09-07 11:46:56	2001-09-07 11:46:56	Verdi	131.114.2.xxx		131.114.4.6

GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
			5	15	1000	5000	3000	2000
			7	3	0	600	300	300
			7	1	1000	1000	1000	0
			9	0	0	1000	500	500

All four of these records will be merged into one *Local Email Downloading* activity as shown below:

ID	RecordType	StartTime	EndTime	InitiatingUser	InitiatingHost
55	Local Email Downloading	2001-09-07 11:17:01	2001-09-07 11:46:56	Verdi	131.114.4.xxx

Target(s)	Description	NumUsages	Data1	Data2	Data3	Tmin	Tmax	Tavg	TstdDev
131.114.4.6		4	20	19	9	595000	600000	598333	244

Also here the StartTime and EndTime are, respectively, the StartTime of the first usage record and the EndTime of the last usage record that is considered as aving generated such activity. InitiatingUser is the name of the user whose credentials were used for authentication, and InitiatingHost and Target(s) are, respectively, the host from which the download started and the IP address of the host on which the POP server is running. Data1 corresponds to the total number of

messages found into the mailbox when the activity started, Data2 corresponds to the total number of messages deleted from the mailbox during the entire activity, and Data3 represents the number of messages left into the mailbox at the end of the activity. The difference between (Data2 + Data3) – Data1 gives the number of new messages received during the activity. Similarly to *Web Surfing* activities, Tmin and Tmax here represent the minimum and maximum time elapsed between two successive downloading attempts, while Tavg and TstdDev correspond to the average and standard deviation of the usage records distribution over time. In this particular case, we can observe a very low TstdDev value. This means that interactions between pop client and server happened at regular intervals, possibly because they are automatically started by a POP client demon. High TstdDev values could mean that during the activity the user was physically sitting at the InitiatingHost and that ho or she has explicitly started an interaction with the POP server. Here is the pseudo code for the piece of the *Activities Builder* that generates *Local Email Downloading* activities:

```
SELECT DISTINCT InitiatingUser, InitiatingHost
FROM  usage table
WHERE Category = "Pop Log"

For each of such pairs {

    SELECT   *
    FROM  usage table
    WHERE InitiatingUser = CrntUser
    AND InitiatingHost = CrntHost
    ORDER BY StartTime

    While (there are more usage records) {
        If the time elapsed since the previous is less than GAP {
            Include this usage record into the activity
        } else {
            close current activity
            store it into the activities table
            and start a new one with current usage record
        }
    }
}
```

Note that the overall structure is almost the same used that for modeling Web Surfing activities. Of course, here, the usage records selected are different.

6.2.2 Remote Email Downloading

Besides the official company email account most users nowadays have other email accounts in various locations on the Internet. Although we cannot assume the availability of log files from such extranet servers, we can use local NET-GW information to capture such activities. At this stage each individual remote email download attempt has already been stored into one usage record like the following (see Chapter 4):

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser	TgtHost
21	Net Log	Pop Session	2001-09-04 12:22:29	2001-09-04 12:22:35		131.114.4.xxx		192.18.97.241

GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
			41	19141	0	5997	261	935

A Remote Email Downloading activity will be the result of the aggregation of one or more of these usage records between the same pair of origin and destination host. The activity is supposed to be finished, and a new one is started when the time elapsed between two successive usage records is greater than a fixed REMOTE_POP_IMAP_GAP parameter. Such an activity record looks as follows:

ID	RecordType	StartTime	EndTime	InitiatingUser	InitiatingHost
55	Remote Email Downloading	2001-09-06-08.19.50	2001-09-06-12.06.51		131.114.4.xxx

Target(s)	Description	NumUsages	Data1	Data2	Data3	Tmin	Tmax	Tavg	TstdDev
192.18.97.241		232	4650	50946		3000	61000	58965	1853

Note that no information about the user is available. In this case this activity, which took almost 4 h, has been built starting from 232 single email downloading attemps, resulting in 4650 tcp packets for a total of 50946 bytes exchanged. As the TstdDev value is quite low, this traffic could have been generated by a demon with a refresh interval of about 60 seconds. This is the pseudo code:

```
SELECT DISTINCT InitiatingHost, TargetHost
FROM  usage table
WHERE Category = "Net Log"
AND RecordType="Pop Session"

For each of such pairs {

    SELECT  *
    FROM  usage table
    WHERE InitiatingHost = CrntLocalHost
    AND TargetHost = CrntRemoteHost
    ORDER BY StartTime

    While (there are more usage records) {
        If the time elapsed since the previous is less than GAP {
            Include this usage record into the activity
        } else {
            close current activity
            store it into the activities table
            and start a new one with current usage record
        }
    }
}
```

6.3 Email Sending Activities

An *Email Sending* activity models the action of a user sending e-mail. We can distinguish two types of such activities: *Local Email Sending* and *Remote Email Sending*. The first corresponds to sending an email by means of the intranet SMTP server, whereas the second corresponds to sending an email using an extranet SMTP server whose log files we cannot assume to be available. They will be described separately in next two paragraphs.

6.3.1 Local Email Sending Activities

Local Email Sending activities have been modeled starting from SMTP usage records (Category = Smtp Log). At this stage we assume that each email sent from a local user to *n* recipients is described by *n* + 1 usage records. So, reusing Example 4.2.1, the act of sending an email by local user Rossi@di.unipi.it to the two recipients Verdi@di.unipi.it and Bianchi@informatik.uni-freiburg.de is described by the following three usage records:

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser
1	Smtp Log	Message Sending	2001-06-18 09:26:37		Rossi@di.unipi.it	131.114.4.xxx	
2	Smtp Log	LclForwarding	2001-06-18 09:27:06	2001-06-18 09:27:06			Verdi@di.unipi.it
3	Smtp Log	RmtForwarding	2001-06-18 09:27:06	2001-06-18 09:27:15			Bianchi@informatik.uni-freiburg.de

TgtHost	GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
	Msg1@di.unipi.it	JAA14975		1	1038				
		JAA14975		1					
mailgateway1.uni-freiburg.de		JAA14975		1					

Now we want to collapse all three records into one *Local Email Sending* activity like the following:

ID	RecordType	StartTime	EndTime	InitiatingUser	InitiatingHost
55	Local Email Sending	2001-06-18 09:26:37	2001-06-18 09:27:15	Rossi@di.unipi.it	131.114.4.xxx

Target(s)	Description	NumUsages	Data1	Data2	Data3	Tmin	Tmax	Tavg	TstdDev
Verdi@di.unipi.it || Bianchi@informatik.uni-freiburg.de		3	1038	2					

Here is the pseudocode used for modeling this activity.

```
SELECT DISTINCT LocalRef
FROM  usage table
WHERE Category = "Smtp Log"
AND RecordType="Message Sending"

For each different LocalRef {

    SELECT   *
    FROM  usage table
    WHERE Category = 'SMTP Log'
    AND LocalRef = CrntRef
    ORDER BY StartTime

    create a new activity using results of last query
    and store it into the activities table

}
```

We note that it is always possible to relate usage records that refer to the same email by looking at the LocRef field. This LocRef id is assigned to the message when it enters the system and can be considered unique for a reasonably long period of time. For each different email (e.g. LocRef value), the algorithm will build a new Local Email Sending activity, merging information from all usage records that refer to that particular email. The result is that we can now merely look at such an activity record to learn that at a certain time a particular user has sent an email of size Data1 to Data2 recipients whose names are reported in the Target(s) field.

6.3.2 Remote Email Sending Activities

It happens quite often that users use an extranet SMTP server to send emails. This kind of activity, which we call *Remote Email Sending,* can be modeled starting from NET-SMTP usage records (Category = *Net Log* AND RecordType = *Smtp session*).

Each such activity is described by a usage record like the following

ID	Category	RecordType	StartTime	EndTime	InitUser	InitHost	TgtUser	TgtHost
1	Net Log	Smtp Session	2001-09-08-10.45.17	2001-09-08-10.45.20		131.114.2.8x		146.48.65.88

GlobRef	LocRef	Descr	DataPkts	DataVol	Tmin	Tmax	Tavg	TstdDev
		Two	89	27366	0	199	38	62

Such a usage record actually is already an activity: it means that at a certain time a 27-Kb email was sent from the local host 131.114.2.8x using the extranet SMTP server running on remote host 146.48.65.88. The *Activities Builder* merely has to copy such a record into the activities table according to the activities schema. Here is the pesudocode:

```
SELECT *
FROM  usage table
WHERE Category = "Net Log"
AND RecordType = "Smtp Session"
AND InitiatingHost <> IntranetMailserver
AND TargetHost <> IntranetServer

For each of such records{

    create a new activity using current record
    and store it into the activities table

}
```

As you can see the Activities Builder does not select every usage record that represents an interaction between a local host and a remote SMTP server because some of them are redundant. In particular, note that the usage table contains also all those records that refer to local emails (emails sent using the intranet SMTP server) forwarded by the intranet mail server to the remote recipients' SMTP servers.

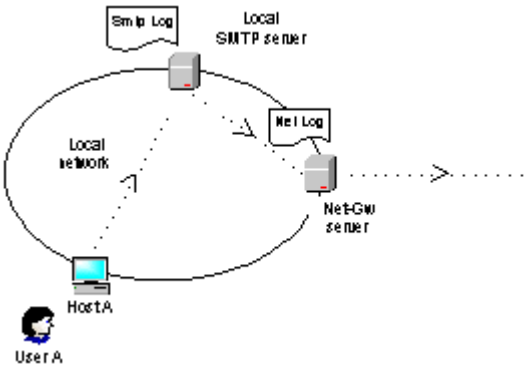

Figure 19: Email Service Configuration

Figure 19 shows the path followed by a generic email sent by a local user using the intranet SMTP server: UserA first sends the email to the intranet SMTP server, which then forwards it to the recipient's SMTP server. In such a situation this activity has already been considered as *Local Email Sending* activity. This is the reason why the *Activities Builder* omits from its selection all those usage records that have the intranet SMTP server as InitiatingHost or TargetHost. Here is the resulting *Remote Email Sending* activity.

ID	RecordType	StartTime	EndTime	InitiatingUser	InitiatingHost
27	Remote Email Sending	2001-09-08-10.45.17	2001-09-08-10.45.20		131.114.2.8x

Target(s)	Description	NumUsages	Data1	Data2	Data3	Tmin	Tmax	Tavg	TstdDev
146.48.65.88		1	27366	89		0	199	38	62

The information provided by this kind of record is different from the preceding one because now we know neither the sender nor the recipients of the email sent.

6.4 Summary

In this chapter we described in detail how an activities table is filled. We have individuated six kinds of activities: Local Web Surfing, Remote Web Surfing, Local Email Downloading, Remote Email Downloading, Local Email Sending, and Remote Email Sending. They have been obtained by integrating different types of usage records resulting from the *Data Parsing and Aggregation* phase. The result consists of a table of meaningful entries that can now be used for OLAP analysis or as starting point for the application of data-mining tasks.

Chapter 7

VALIDATION ON A CAMPUS NETWORK

To validate the process of combining log information from network and application servers to compute an aggregate picture of computers and users according to detected communication patterns, we tested it with the Computer Science departmental staff network of the University of Pisa. According to a human source, there exist a total of approx. 300 hosts (50 servers and 250 workstations). Under a confidentiality agreement we were able to get access to a full 7-day week of real traffic logs from the departmental web and mail servers and the gateway to the backbone of the university network. The validation playground is depicted in Figure 20.

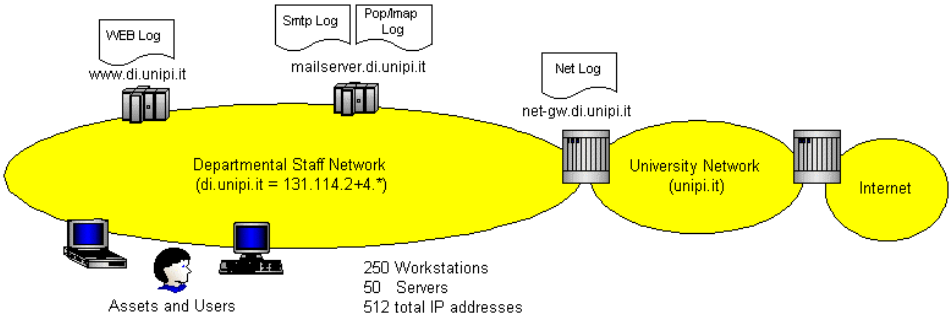

Figure 20: Actual Network and Log-File Sources at the University of Pisa

7.1 Data Sources Integration

As already stated, a significant problem tackled by the author is the large amount of data to parse. In the setup described, the first week in September corresponds to a total of 13 Gbytes uncompressed raw TCPDUMP, 46 MB HTTP, 8/2 MB POP/IMAP, and 7 MB SMTP log files.

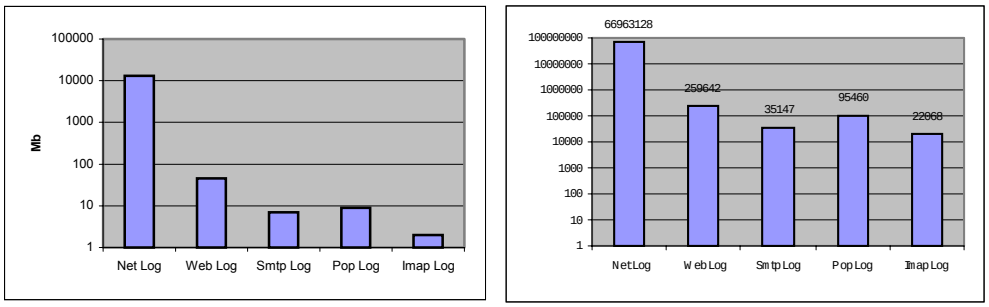

Figure 21: Log File Sizes and Log Entries

Although the first releases of our parsers needed more than two days to complete the *Data Parsing and Aggregation* phase, the actual warehousing of usage records take less than two hours, parsing approx. 100,000,000 log entries, directly filtering out 30% as not-studied protocols, and creating 335,000 usage records. The two charts in Figure show from which log file usage records have been generated and their distribution over the different protocols. The usage records are distributed as follows over the protocols studied: 64% HTTP, 19% SMTP, 14% POP, 3% IMAP. The distribution of the usage-record log source is 68% Net, 10% Web, 22% Mail.

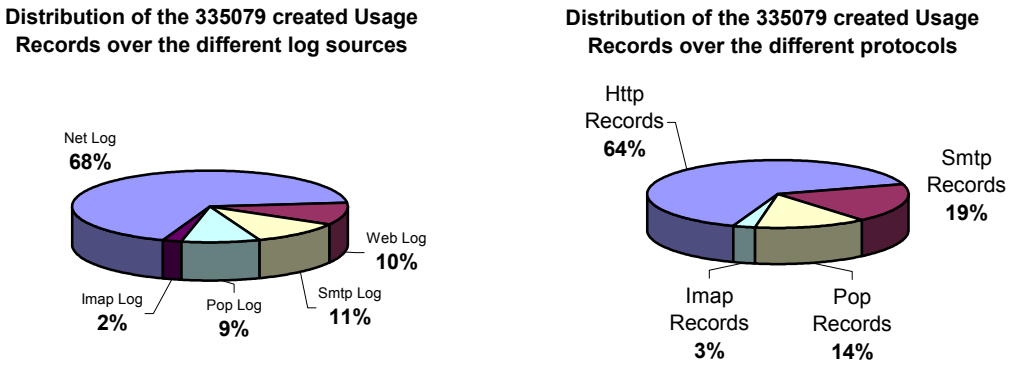

Figure 19: Usage Records Distributions

In other words, most usage records correspond to Internet web surfing logged by the network gateway. The protocol used most in terms of usage records (with 1 min. as usage gap) is definitively HTTP with 64%.

Finally, the usage records were aggregated into 9,100 activities, as consolidated input for the subsequent data analysis. In the form of aggregated activities (with 60 min as activity gap) web surfing still represent 29%, whereas email sending and downloading activities are at 50% and 21%, respectively.

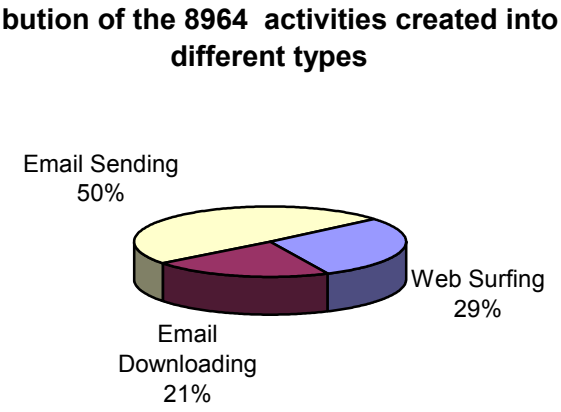

Figure 22: Activity Records Distribution

Note that it should not be surprising that Email Sending activities have a much higher share than the others: The reason is that one such activity corresponds to one email sent whereas the other types regard activities that can last several hours.

7.2 Some Results

We will show now some preliminary results obtained merely by querying the activities table. Clearly it is always possible to calculate a lot of statistics about the traffic generated by each single host (we will give some examples at the end of the chapter) but here we want to do something more.

Figure 23 shows that 56% of the discovered hosts that were analyzed are used by a single user, hence can be considered private workstations.

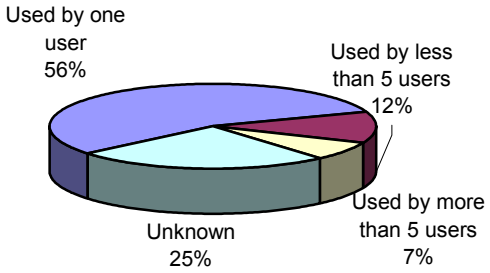

Figure 23: Users on Hosts

This chart has been obtained looking at POP/IMAP and SMTP activities (which provide user information) started from each host. The 25% of unknown means that no POP/IMAP or SMTP activities were started from those hosts during the week, whereas 13 hosts (7% of 182) have been used by more than 5 different users to download emails; this is not completely true because 8 of them are IP addresses that are dynamically assigned by a dial-up service when people connect to the departmental network via a modem, whereas the others are servers that users can use via telnet to download emails from outside. The others (hosts with fewer than 5 users) are shared machines that can be used for example by students for their thesis work.

Hosts are mainly used for both web and mail but there are 32 of the discovered hosts that are used only for web surfing, and 5 hosts that are used only for email.

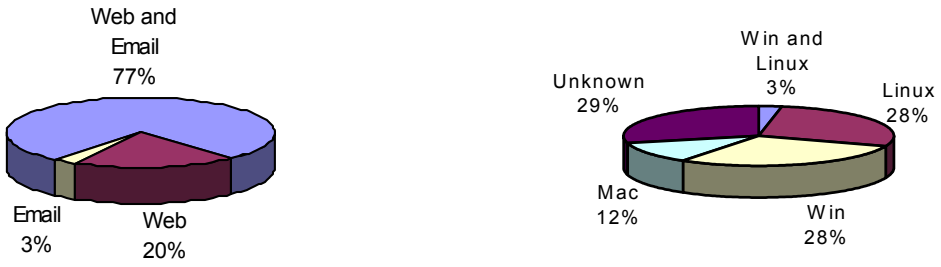

Figure 24: Hosts Utilization and Configuration

The 56% of the discovered hosts are Windows and Linux machines; about 12% are Macs. Finally, 3% of discovered hosts run both Windows and Linux operating systems. This information has been collected by looking at the *description* field of the *Local Web Surfing* activities. The 29% unknown means that about 50 hosts never surfed the intranet web server during the week, and that therefore no information about the operating system is available.

We now want to focus on a particular user or host, and try to gather information about its behavior.

Example 7.2.1

This example shows the results obtained by querying the activities table for all activities started from host 131.114.4.XXX on September 10. Looking at *Local Email Downloading* and *Local Email Sending* activities, we learn that host 131.114.4.XXX, which is a Linux machine, is shared by users Jordan and Bird. Figure 25 reveals that on September 10 both users used the host.

TYPE	START	END	INITUSER	DESCR	NUSG	VOLUME	AVG	SDEV
Local Email Downloading	9:33:59 AM	6:02:25 PM	Jordan		542	151	58	19
Local Web Surfing	9:36:07 AM	11:01:19 AM		Mozilla/4.76 [en] (X11; U; Linux 2.4.2-2smp i686)	16	426448	340	354
Local Email Sending	9:41:59 AM	9:41:59 AM	Jordan		1	1174		
Local Email Sending	9:42:35 AM	9:42:35 AM	Jordan		1	1235		
Local Email Sending	9:50:30 AM	9:50:54 AM	Jordan		2	529508		
Local Email Sending	9:55:27 AM	9:56:32 AM	Jordan		1	2322		
Local Email Sending	10:36:04 AM	10:36:04 AM	Jordan		1	1486		
Local Email Sending	10:57:13 AM	10:57:14 AM	Jordan		1	2396		
Local Email Sending	11:31:51 AM	11:31:51 AM	Jordan		2	1646		
Local Email Sending	11:34:57 AM	11:35:13 AM	Jordan		1	3996		
Local Email Sending	11:36:46 AM	11:36:48 AM	Jordan		1	1142		
Local Email Sending	11:38:02 AM	11:38:02 AM	Jordan		1	1579		
Local Web Surfing	1:52:25 PM	1:52:26 PM		Mozilla/4.76 [en] (X11; U; Linux 2.4.2-2smp i686)	1	43867	0	0
Local Web Surfing	3:19:32 PM	3:19:33 PM		Mozilla/4.76 [en] (X11; U; Linux 2.4.2-2smp i686)	1	43867	0	0
Local Email Sending	4:12:35 PM	4:12:35 PM	Jordan		1	2281		
Local Email Sending	4:16:30 PM	4:16:52 PM	Jordan		1	3544		
Local Web Surfing	4:33:58 PM	5:31:07 PM		Mozilla/4.76 [en] (X11; U; Linux 2.4.2-2smp i686)	11	298857	342	688
Local Email Downloading	6:46:25 PM	6:46:32 PM	Bird		1		0	0
Web Surfing	6:48:03 PM	7:06:19 PM			64	901660	17	25
Local Email Downloading	8:49:29 PM	8:54:37 PM	Bird		2		81	0
Local Web Surfing	8:55:16 PM	9:31:19 PM		Mozilla/4.76 [en] (X11; U; Linux 2.4.2-2smp i686)	21	885162	108	405
Local Web Surfing	10:49:51 PM	11:27:57 PM		Mozilla/4.76 [en] (X11; U; Linux 2.4.2-2smp i686)	92	3306098	24	46
Local Email Downloading	11:05:29 PM	11:08:43 PM	Bird		1		0	0
Local Email Sending	11:05:55 PM	11:05:56 PM	Bird		1	549		

Figure 25: Example of a Host Behavior

In particular we can say that a POP client daemon (started by user Jordan at 9:33:59 AM) has been running until 6:02:25 PM, with a refresh range of about 60 seconds. Then host 131.114.4.XXX has been used for web surfing for about one and a half hour using a Mozilla 4.76 web browser. During this time, several emails were sent to several recipients (not shown in Figure 25, but available into the database) from the host by user Jordan. The host was not used between 11:38:02 AM and 1:52:26 PM, but the POP client daemon was still running and polling the intranet POP server every minute. Then several web and email activities were started in the afternoon. At 6:46:25, the host started to be used by another user (user Bird), which generated activities until 11:05:55 PM for a total of 5MB of traffic exchanged. In this example some information such as the number of messages downloaded or the name of the hosts surfed has been omitted.

In next example the focus is no longer on the host but on the user instead.

Example 7.2.2

By looking at *Local Email Downloading* and *Local Email Sending* activities, we learnt that user Platini has two private workstations: host 131.114.1X (a Windows 95 machine) and host 131.114.2Y (a Windows NT 4 machine). Figure 26 below shows the activities started by user Platini on September 7.

TYPE	START	END	INITHOST	DESCR	USG	VOLUME	AVG
Web Surfing	9:01:43 AM	9:04:18 AM	131.114.4.1X	Mozilla/4.0 (compatible; MSIE 5.5; Windows 95)	2	49029	149
Local Email Downloading	9:04:32 AM	1:44:54 PM	131.114.4.1X		30	2	579
Web Surfing	9:05:12 AM	11:51:29 AM	131.114.4.2Y	Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 4.0)	151	6795963	66
Remote Email Downloading	9:07:12 AM	10:22:29 AM	131.114.4.2Y		13	214890	375
Web Surfing	10:31:14 AM	1:10:11 PM	131.114.4.1X		84	3544627	114
Local Email Sending	10:34:01 AM	10:34:02 AM	131.114.4.1X		1	25194	

TYPE	START	END	INITHOST	DESCR	USG	VOLUME	AVG
Local Email Sending	10:37:58 AM	10:37:58 AM	131.114.4.1X		1	2282	
Local Email Sending	10:56:45 AM	10:56:46 AM	131.114.4.1X		1	8486	
Remote Email Downloading	11:16:15 AM	11:55:11 AM	131.114.4.2Y		9	262047	292
Remote Email Downloading	11:16:15 AM	1:35:23 PM	131.114.4.2Y		29	6318	298
Remote Email Sending	11:20:35 AM	11:20:35 AM	131.114.4.2Y		1	352	0
Remote Email Sending	11:23:20 AM	11:23:20 AM	131.114.4.2Y		1	352	0
Remote Email Sending	11:37:58 AM	11:37:59 AM	131.114.4.2Y		1	279	0
Remote Email Downloading	11:56:41 AM	1:35:23 PM	131.114.4.2Y		20	6830	311
Local Email Sending	12:31:54 PM	12:31:55 PM	131.114.4.1X		1	202657	
Local Email Sending	12:59:36 PM	12:59:37 PM	131.114.4.1X		1	4689	
Local Email Sending	1:00:36 PM	1:00:37 PM	131.114.4.1X		1	1053	
Web Surfing	1:04:53 PM	1:25:45 PM	131.114.4.2Y	Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 4.0)	9	2905667	152
Remote Email Downloading	2:46:21 PM	3:06:01 PM	131.114.4.2Y		4	11204	393
Remote Email Downloading	2:46:22 PM	3:06:02 PM	131.114.4.2Y		4	876	393
Web Surfing	3:07:52 PM	4:51:34 PM	131.114.4.2Y		123	3406742	50
Remote Email Downloading	3:09:55 PM	4:40:31 PM	131.114.4.2Y		13	94541	452
Local Email Downloading	3:54:50 PM	5:35:20 PM	131.114.4.1X		9	1	751
Web Surfing	3:59:59 PM	5:33:56 PM	131.114.4.1X		30	655751	191
Remote Email Downloading	4:51:49 PM	5:32:45 PM	131.114.4.2Y		7	25848	409
Remote Email Downloading	4:51:50 PM	5:34:35 PM	131.114.4.2Y		8	1740	366
Local Email Sending	5:05:34 PM	5:05:54 PM	131.114.4.1X		1	1068558	
Remote Email Downloading	5:34:32 PM	5:34:32 PM	131.114.4.2Y		1	843	0

Figure 26: Example of a User Behavior

As you can see user Platini has started many activities throughout the day starting at 9:01:43 AM: he sent several emails using both the local SMTP server and two different extranet SMTP servers, he downloaded emails using both intranet and extranet servers, and he surfed the web generating about 20 MB of traffic. We can also say that he used a POP client daemon on host 131.114.4.1X for polling the intranet POP server and a POP client daemon running on host 131.114.4.2Y for polling the other extranet POP servers to which he is subscribed. Another interesting observation regards the fact that he uses the two hosts differently: host 131.114.4.2Y is used for all interactions with external mail servers, whereas all activities involving intranet mail servers are started from host 131.114.4.1X.

As already mentioned, it is always possible to obtain a lot of statistic about the protocols studied merely by using SQL queries or scripts. Some examples are shown in Figure 27 and Figure **28** below.

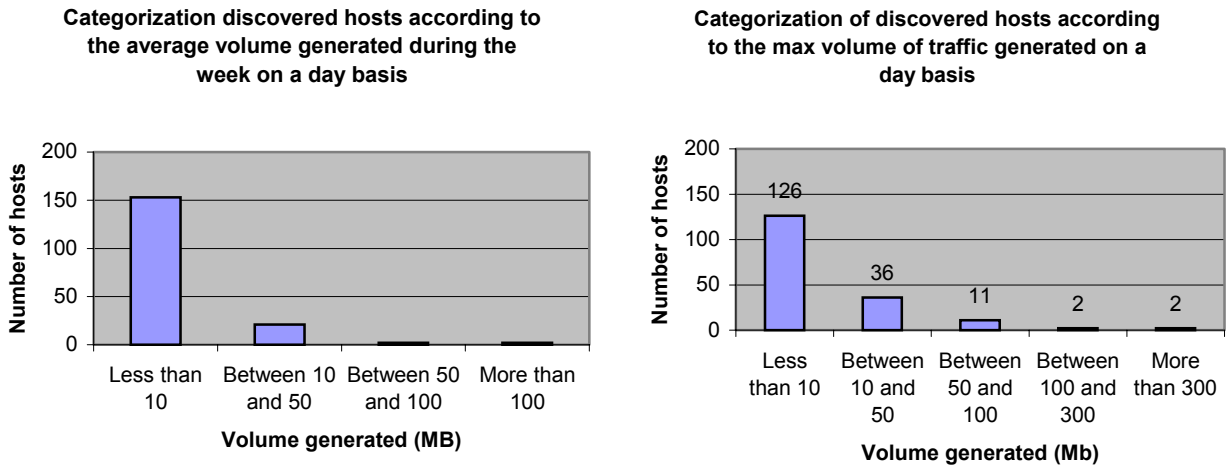

Figure 27: Some Statistics I

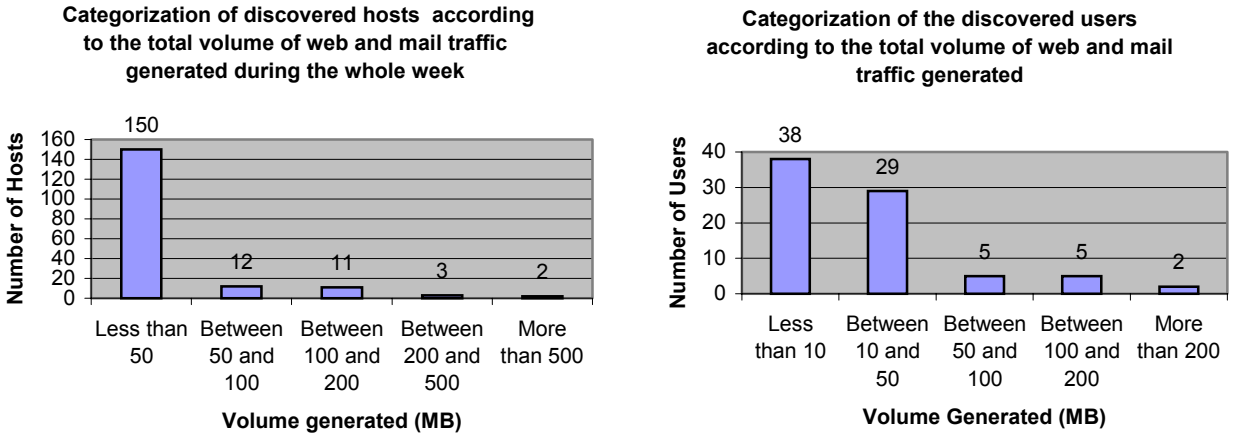

Figure 28: Some Statistics II

The aim of showing all these examples is to convince the reader that now a lot of information is available to the analyst. Data now are easily manageable, clean, and stored in a relational database that can be used as starting point for further specific analysis by means of either SQL queries or more advanced data-mining tasks to discover hidden information.

7.3 Summary

In this chapter we summarized the validation process of our approach using data from the Computer Science departmental network. We started with about 13 GB of log data, and finished with about 9K of activity records after a process of both merging data from different sources and cleaning them from inconsistencies and redundancy. The chapter is concluded with several examples on how such meaningful entries can be used to model asset and user behavior.

Chapter 8

CONCLUSIONS AND FUTURE WORK

In today's dynamic information society, organizations critically depend on the underlying computing infrastructure. Tracking computing devices as assets and their usage helps in the provision and maintenance of an efficient, optimized service. Building an accurate inventory of computing assets is especially difficult in unknown heterogeneous systems and networking environments without prior device instrumentation. User mobility and mobile, not-always-signed-on, computing devices add to the challenge. We therefore propose to complement basic network-based online discovery techniques with the combined historic log information from network and application servers to compute an aggregate picture of assets, and to categorize their usage with data-mining techniques according to detected communication patterns[7].

In this work we outlined the process of warehousing and analyzing network and application server logs to track assets and their usage. Given our initial validation, we hope to establish the potential of integrating the consolidated historic knowledge residing in access-specific gateways, firewalls, VPN servers, and network proxies, and the growing wealth of application-specific servers. We anticipate that in the long run the gathering of usage information from the various network and application subsystems will prove to be the most cost-effective asset and usage management scheme. Having already been established as the best method for pervasive devices, it may actually become standard usage for unknown heterogeneous environments and nicely complement the management of other, more static hosts, always-on workstations and servers.

The authors are aware that this work is not complete as there are some open problems and challenges in categorizing computing assets. Referring to the lessons learnt during the validation process, the value of the warehoused data increases with more careful cleaning and coding. In particular, this work also needs to be extended in order to categorize accurately mobile and pervasive devices that can roam across different networks and be active only for a limited period of time.

Network intrusion-detection and customer-relationship management are established fields that benefit from OLAP techniques. We hope to promote the use of similar forms of data mining also as techniques for corporate asset management and to establish a flexible and dynamic management infrastructure for e-business services[11][12]. Figure 29 proposes a chain of processing steps, starting with the classical network discoveries, adding log analysis for usage categorization, that may eventually allow questions about the cost, utility, and risk associated with individual assets to be answered on the one hand, and the computation of associated values on the other.

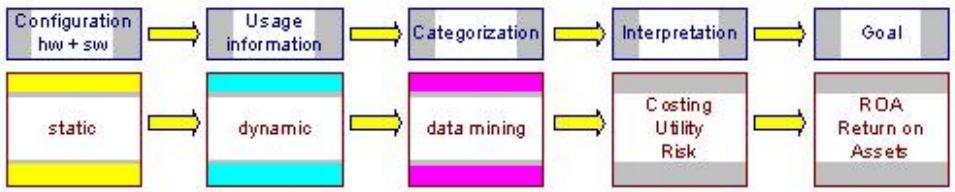

Figure 29: Processing Stages for the Computation of Business Values

Additional ideas include the following:

- Enlarge the set of data sources studying and integrating new protocols to discover chat[22] and Instant Messaging sessions[37], streaming audio and

video, ftp and similar files downloading protocols (such as Napster and Gnutella), telnet, ssh, https etc.

- Enhance the proposed system with an accounting application that allows tracking the service usage, its users, and its availability.

- Study how to generate alarms (e.g. SNMP traps) when an asset modifies its behavior (e.g. if a computer that is known not to handle mail at some point routes emails, it means that something has changed or that a virus is running on the asset).

- Apply data-mining algorithms to activities resulting from the warehousing process of log files data in order to discover hidden regularities among assets and users behavior[11].

Eventually, we would like to derive conclusions such as: "Computer A is used by a secretarial person 5/7 days a week in the morning. Computer B probably is a student-lab workstation, shared by users X, Y and Z." The lessons learned during the validation process are that (i) there is hope to achieve this – eventually – but (ii) it is of utmost importance to minutely parse, clean, code, and compress the original data sources, and (iii) there is no way around having a well-known sample population of users and computers to establish a data-mining model allowing OLAP predictions in newly discovered environments.

BIBLIOGRAPHY

[01] R.Siamwalla, R.Sharma, and S.Keshav, *Discovering Internet Topology.* Cornell Network Research Group Department of Computer Science Cornell University, Ithaca, NY 14853

[02] D.J. Beckett, *Combined log system* Computer Networks and ISDN Systems Proceedings of the Third International World-Wide Web Conference*, April 1995*

[03] R.H. Katz, S. Seshan, and M. Stemm, SPAND: *Shared Passive Network Performance Discovery,* Proc 1st Usenix Symposium on Internet Technologies and Systems (USITS '97), Monterey, CA, December 1997.

[04] M.S. Chen, J.S. Park, and P.S. Yu. *Data Mining for path traversal patterns in a Web environment.* In Proceedings of the 16th International Conference on Distributed Computing Systems, pages 385-392, 1996.

[05] Dethy. *Advanced Host Detection: Technques to validate Host-Connectivity,* http://www.synnergy.net/downloads/papers/host-detection.txt

[06] R. Cooley, B. Mobasher, and J. Srivastava. *Grouping Web Page References into Transactions for Mining World Wide Web Browsing Patterns.* In Proceedings of KDEX'97, Newport Beach, CA, 1997

[07] D. Gantenbein, M. Filoni, L. Deri. *Categorizing Computing Assets According to Communication Patterns*, submitted to Networking 2002, Pisa, May 2002.

[08] R. Cooley, B. Mobasher, and J. Srivastava. *Data Preparation for Mining World Wide Web Browsing Patterns.* Knowledge and Information Systems Conference. Springer-Verlag 1999

[09] S.Ruggieri et al., *Web Log Data Warehousing and Mining for Intelligent Web Caching.* Data and Knowledge Engineering. Vol 32, Issue 2, October 2001, 165-189.

[10] L. Catledge and J.Pitkow. *Characterizing browsing behaviors on the World Wide*

Web. Computer Networks and ISDN Systems, 1995

[11] J. Han and M. Kamber, *Data Mining: Concepts and Techniques,* Academic Press, 2001, ISBN 1-55860-489-8

[12] P. Adriaans, D. Zantinge, *Data Mining,* Addison-Wesley, 1996, ISBN 0-201-40380-3

[13] L.Deri, S.Suin. *Monitoring Networks Using Ntop.* In Proceedings of the 2001 IEEE/IFIP International Symposium on Integrated Network Management. Seattle, 2001.

[14] L. Deri and S.Suin, *Effective Traffic Measurement using ntop*, IEEE Communications Magazine, May 2000.

[15] L. Deri and S.Suin, *Ntop: beyond Ping and Traceroute*, DSOM, 1999,

[16] P. Pirolli, J. Pitkow, and R. Rao. *Silk from a sow's ear: Extracting usable structures from the Web.* In proc. of 1996 Conference on Human Factors in Computing Systems, Vancouver 1996.

[17] J. Pitkow. *In search of reliable usage data on the www.* In Sixth International World Wide Web Conference, pages 451-463, Santa Clara, CA, 1997.

[18] Jonathan B. Postel. *Rfc 821: Simple Mail Transfer Protocol.* August 1982

[19] J.Myers, M.Rose. *Rfc 1939: Post Office Protocol - Version 3*, May 1996

[20] M. Crispin. *Rfc 1730: Internet Message Aceess Protocol – Version 4,* December 1996

[21] J. Case et.al., *Rfc 1157: Simple Network Management Protocol (SNMP).*

[22] J. Oikarinen, D. Reed. *Rfc 1459: Internet Relay Chat Protocol.*

[23] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones *Rfc 1928: SOCKS Protocol Version 5.*

[24] Ying-Da Lee. *SOCKS: A protocol for TCP proxy across firewalls.*

[25] Fielding, et al. *Rfc 2616: Hypertext Transfer Protocol -- HTTP 1.1.*

[26] J. Postel. Rfc 792: *Internet Control Message Protocol.* September 1981.

[27] C. Kalt. *Rfc 2810: Internet Relay Chat: Architecture.* April 2000.

[28] C. Kalt. *Rfc 2811: Internet Relay Chat: Channel Management.* April 2000.

[29] C. Kalt. *Rfc 2812: Internet Relay Chat: Client Protocol.* April 2000.

[30] C. Kalt. *Rfc 2813: Internet Relay Chat: Server Protocol.* April 2000.

[31] A. G. Lowe-Norris, *Windows 2000 Active Directory*, O'Reilly, ISBN 3-89721-171-8, 2001

[32] W3 Consortium: *Logging Control in W3C httpd*

[33] W3 Consortium: *Extended Common Log File Format.* Working Draft

[34] *Intelligent Device Discovery (IDD) Project*, IBM Zurich Research Laboratory, http://www.zurich.ibm.com/csc/ibi/idd.html

[35] *NetProxy Home Page.* http://info.grok.co.uk/index.html

[36] *ICQ web site.* http://www.icq.com/products/

[37] *ICQ Protocol Specification.* http://omega.uta.edu/~tom/ICQ/

[38] *AOL Instant Messenger* http://www.aol.com/aim/

[39] *AIM/OSCAR protocol specification.* www.zigamorph.net/faim/protocol/

[40] *The Apache Software foundation.* http://www.apache.org/

[41] *Squid Home Page.*http://www.squid-cache.org

[42] *Squid Log Files.* http://www.squid-cache.org/Doc/FAQ/FAQ-6.html

[43] *Sendmail Home Page.* http://www.sendmail.org

[44] *Microsoft Exchange Home Page.* http://www.microsoft.com/Exchange

[45] *Microsoft Exchange: Tracking Log*, http://www.microsoft.com/Exchange/en/55/help/default.asp

[46] *DNS Resources Directory.* http://www.dns.net/dnsrd/

[47] *Internet Software Consortium: BIND.* http://www.isc.org/products/BIND/

[48] V. Jacobson, C. Leres, and S. McCanne, *tcpdump*, Lawrence Berkeley National Labs, ftp://ftp.ee.lbl.gov/, 1989

[49] *Wuarchive-ftpd Home Page.* http://www.wu-ftpd.org

[50] Baromedia 2001: *Resultats du barometre des medias suisses.*

[51] *Access Log Analyzers.* http://www.uu.se/Software/Analyzers/Access-analyzers.html

[52] *Winfingerprint* Windows Information Gathering Tool,

http://winfingerprint.sourceforge.net/

[53] *Ethereal free network protocol analyzer for Unix and Windows,* http://www.ethereal.com/

[54] Centennial, *Network Inventory Audit*, http://www.intertechnology.ca/soft-1.htm

[55] *NetView: Tivoli product*, http://www.tivoli.com/products/index/netview/

[56] *NMAP Free Security Scanner*, http://www.insecure.org/nmap/index.html

[57] *NSA Firewall Network Security Auditor*, http://www.research.ibm.com/gsal/gsal-watson.html

[58] *Infratools: Network and Desktop Discovery* http://www.peregrine.com

[59] Tivoli, *Inventory management*, http://www.tivoli.com/products/index/inventory/

[60] Tivoli, *Performance solutions*, http://www.tivoli.com/products/solutions/availability/news.html

[61] S. Branigan et al., *What can you do with Traceroute?,* http://www.computer.org/internet/v5n5/index.htm

[62] David C. Plummer. Rfc 826: *An Ethernet Address Resolution Protocol ,* Nov 1982

[63] Microsoft Office - *Microsoft Visio Home Page* www.microsoft.com/office/visio/default.htm

[64] B. Croft, and J. Gilmore. *Rfc 951: Bootstrap Protocol (BOOTP),* Stanford and SUN Microsystems, September 1985.

[65] R. Droms *Rfc 2131: Dynamic Host Configuration Protocol,* March 1997

[66] W. Yeong, T. Howes, S. Kille. *Rfc 1777: Lightweight Directory Access Protocol,* March 1995

[67] *HP OpenView homepage*. http://www.openview.hp.com

[68] *Extreme Tracking* http://www.extreme-dm.com/tracking/?home

[69] *Log Analyzers* http://www.uu.se/Software/Analyzers

[70] *Tivoli Asset Management Architecture* http://www.tivoli.com/support/public/Prodman/public_manuals/td/TSD/TSD6_techref/en_US/HTML/rtamarch.htm

Appendix A

THE AGGREGATORS

A.1 Net Log Aggregator

```
/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'NetLogParser' class generates usage records aggregating
 * Net log entries
 *
 * File     : NetLogParser.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

import java.io.*;
import java.sql.*;
import com.ibm.idd.db.*;
import gnu.getopt.Getopt;

import com.ibm.idd.lib.conv.Regex2DB;
import com.ibm.idd.lib.AppFileProperties;

/* Sample Net log line:
*
*
* 998736153.079056 < 216.218.255.232.www > 131.114.2.26.3669: . 32121:33581(1460) ack 326 win
6432 (DF)
*
*
*/

class NetLogParser{

  static File logFile = null;

  public static void main(String args[]) {

    Connection        dbCon = null;
    BufferedReader    logReader = null;
    String            user = "", localHostName = "", recorder = "";
    String            application = "Net-Gw", schema = "CADLI";
```

```
String            server = "", source, sqlStatement;
long              time1 = 0;
int               c;

try {
  // have to do this to init log4j:
  AppFileProperties.loadProperties();

  // Default
  localHostName = java.net.InetAddress.getLocalHost().getHostName();

  // COMMAND LINE PARSING
  Getopt g = new Getopt("CommandLineParser", args, "u:h:a:s:l:");
  while ((c = g.getopt()) != -1) {
    switch(c) {
    case 'u':
      user = g.getOptarg();
      break;
    case 'h':
      localHostName = g.getOptarg();
      break;
    case 'a':
      application = g.getOptarg();
      break;
    case 's':
      schema = g.getOptarg();
      break;
    case 'l':
      server = g.getOptarg();
      break;
    case '?':
      break;
    default:
      System.out.println("getopt() returned " + c + "\n");
    }
  }

  if (user.equals("")) {
    recorder = localHostName;
  }
  else {
    recorder = user + " on " + localHostName;
  }
  source = args[g.getOptind()];
  logFile = new File(source);

  // TABLES AND CONNECTIONS CREATION
  TableAccessProperty tap =
      new TableAccessProperty (DatabaseAccessProperty.getDefault(), schema, "USAGE");
  SimpleTableLocalAccess.assertTableCreated(tap);
  dbCon = DatabaseLocalAccess.getConnection();
  dbCon.setAutoCommit(true);

  // LOG PARSING
  if (logFile.exists() == true) {
    FileReader fr = new FileReader(logFile);
    logReader = new BufferedReader(fr);

    // define SQL template
    sqlStatement = "INSERT into " + schema + ".USAGE " +
      "values (DEFAULT,'Net Log','" + server + "','" + source + "','{3}'," +
      "'{0}','{1}','','{2}','','{4}','','',"+
      "'{11}',{6},{5},{7},{8},{9},{10},'" + recorder + "',DEFAULT)";
```

```
        // define regex:
        String regex = "([.\\d]*) [<>] ([\\d]*.[\\d]*.[\\d]*.[\\d]*).([\\w]*) > " +
          "([\\d]*.[\\d]*.[\\d]*.[\\d]*).([\\w]*): [.FPRS]* [:\\d]*\\(([\\d]*)\\) .*";

          Regex2DB conv = new Regex2DB(logReader,regex,dbCon,sqlStatement);
        conv.setParsingHandler(new NetParsingHandler());

        // ... and then convert:
        conv.convert();

        // that's it
        System.exit(0);
      }
      else System.out.println("Error: File " + args[0] + " doesn't exist");

    } catch (Exception e) {
      System.out.println("Error: Exception occurred");
      e.printStackTrace();
    } finally {
      if (dbCon != null) {
        try {
          dbCon.close();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
      if (logReader != null) {
        try {
          logReader.close();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
    }
  }
}


*******************************************************************************



/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'NetParsingHandler' class handles Net log entries
 *
 *
 * File     : NetParsingHandler.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

import java.sql.*;
import java.util.*;
import com.ibm.idd.lib.parser.BasicParsingHandler;
```

```
import com.ibm.idd.lib.parser.GeneratorException;


public class NetParsingHandler extends BasicParsingHandler {

  final  int         RANGE = 60000; // 1 minute
  final  int         REFRESH_RANGE = RANGE * 1;
  static Hashtable   hashCon = new Hashtable();
  static long        prvTime = 0;


  public NetParsingHandler() {
    super();
  }


  /*
  * This method is called when a line of text has
  * successfully parsed. The parsed tokens are passed as
  * an array of String. The line which was
  * parsed is given for information purposes.
  */

  public void processTokens (String line, String[] tokens) throws GeneratorException  {

    String[]  ntokens = new String[12];
    String    hashKey = null, crntKey;
    String    msString, secString;
    long      msLong, msTemp, secLong;
    long      crntTime, lastTime, crntGap, numBytes;
    int       direction = 0;
    double    avg = 0, stdDev = 0;
    NetObj    hashEl;

    try {
      if ((tokens == null) || (tokens.length == 0)) {
      return;
    }


    // right now we handle only {www,pop,imap,smtp,chat,ssh,telnet,ftp} sessions
    // direction = 1 : client -> server
    // direction = 2 : server -> client

    if (tokens[2].equals("www")) {

      hashKey = tokens[3] + tokens[1] + tokens[2];
      direction = 2;

    } else if (tokens[4].equals("www")) {

      hashKey = tokens[1] + tokens[3] + tokens[4];
      direction = 1;

    } else if (tokens[2].equals("pop3") || tokens[2].equals("imap2") ||
               tokens[2].equals("smtp") || tokens[2].equals("5190")  ||
               tokens[2].equals("ssh")  || tokens[2].equals("telnet")||
               tokens[2].equals("ftp"))  {

      hashKey = tokens[3] + tokens[4] + tokens[1] + tokens[2];
      direction = 2;

    } else if (tokens[4].equals("pop3") || tokens[4].equals("imap2") ||
               tokens[4].equals("smtp") || tokens[4].equals("5190")  ||
```

```
           tokens[4].equals("ssh")  || tokens[4].equals("telnet")||
           tokens[4].equals("ftp"))  {

 hashKey = tokens[1] + tokens[2] + tokens[3] + tokens[4];
 direction = 1;

} else {

 return;

}

// Timestamp creation from raw log time field
secString = tokens[0].substring(0,tokens[0].indexOf("."));
secLong = new Long(secString).longValue();
msLong = secLong * 1000;
msString = tokens[0].substring(tokens[0].indexOf(".")+1, tokens[0].length());
msTemp = new Long(msString).longValue();
msLong = msLong + (msTemp/1000);

if (prvTime == 0) {
  prvTime = msLong;
}

crntTime = msLong;
numBytes = new Integer(tokens[5]).intValue();

// if this session is already present into the hashtable
if (hashCon.containsKey(hashKey)) {

  hashEl = (NetObj)hashCon.remove(hashKey);
  lastTime = hashEl.getEndTime();
  crntGap = crntTime - lastTime;

  // if the elapsed time since last packet of this session is greater than RANGE
  if ((crntGap) >= RANGE) {

    // I first store the corresponding bucket into the DB
    ntokens[0] = new Timestamp(hashEl.getStartTime()).toString();// StartTime
    ntokens[1] = new Timestamp(lastTime).toString();             // EndTime
    ntokens[2] = hashEl.getOrHost();                             // InitiatingHost
    ntokens[3] = hashEl.getType();                               // RecordType
    ntokens[4] = hashEl.getDstHost();                            // TargetHost
    ntokens[5] = new Long(hashEl.getVolume()).toString();        // DataVolume
    ntokens[6] = new Integer(hashEl.getPackets()).toString();    // DataPackets
    ntokens[7] = new Long(hashEl.getMin()).toString();           // Tmin
    ntokens[8] = new Long(hashEl.getMax()).toString();           // Tmax

    avg = (hashEl.getAvgCounter()
        / Math.max((hashEl.getPackets() - 1),1));
    ntokens[9] = new Double(avg).toString();                     // Tavg

    stdDev = Math.sqrt((hashEl.getStdDevCounter()
        / Math.max((hashEl.getPackets() - 1),1)) - (avg * avg));
    ntokens[10] = new Double(stdDev).toString();                 // TstdDev

    if (hashEl.getDir() == 3) {

      ntokens[11] = "Two";

    } else {

      ntokens[11] = "One";
```

```
    }
    processTokens(ntokens);

    // and now I store the new packet into the hash table
    hashEl.setStartTime(crntTime);
    hashEl.setEndTime(crntTime);
    hashEl.setVolume(numBytes);
    hashEl.setPackets(1);
    hashEl.setMin(Integer.MAX_VALUE);
    hashEl.setMax(0);
    hashEl.setAvgCounter(0);
    hashEl.setStdDevCounter(0);
    hashEl.setDir(direction);
    hashCon.put(hashKey, hashEl);

  } else {
    //otherwise I just update the current connection
    hashEl.setEndTime(crntTime);
    hashEl.setVolume(hashEl.getVolume() + numBytes);
    hashEl.setPackets(hashEl.getPackets() + 1);
    hashEl.setMin(Math.min(crntGap,hashEl.getMin()));
    hashEl.setMax(Math.max(crntGap,hashEl.getMax()));
    hashEl.setAvgCounter(hashEl.getAvgCounter() + crntGap);
    hashEl.setStdDevCounter(hashEl.getStdDevCounter() + (crntGap * crntGap));

    if ((hashEl.getDir() != direction) && (numBytes > 0)) {
      hashEl.setDir(3);
    }
      hashCon.put(hashKey, hashEl);

    }

  } else {

    // I create a new session object and I insert it into the hash table
    if (tokens[2].equals("www")) {

      NetObj hashSlot = new NetObj(crntTime,tokens[3],tokens[1],"Http Session",numBytes,2);
      hashCon.put(hashKey, hashSlot);

    }else if (tokens[4].equals("www")) {

      NetObj hashSlot = new NetObj(crntTime,tokens[1],tokens[3],"Http Session",numBytes,1);
      hashCon.put(hashKey, hashSlot);

    }else if (tokens[2].equals("pop3")) {

      NetObj hashSlot = new NetObj(crntTime,tokens[3],tokens[1],"Pop Session",numBytes,2);
      hashCon.put(hashKey, hashSlot);

    }else if (tokens[4].equals("pop3")) {

      NetObj hashSlot = new NetObj(crntTime,tokens[1],tokens[3],"Pop Session",numBytes,1);
      hashCon.put(hashKey, hashSlot);

    }else if (tokens[2].equals("imap2")) {

      NetObj hashSlot = new NetObj(crntTime,tokens[3],tokens[1],"Imap Session",numBytes,2);
      hashCon.put(hashKey, hashSlot);

    }else if (tokens[4].equals("imap2")) {

      NetObj hashSlot = new NetObj(crntTime,tokens[1],tokens[3],"Imap Session",numBytes,1);
      hashCon.put(hashKey, hashSlot);
```

```
  }else if (tokens[2].equals("smtp")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[3],tokens[1],"Smtp Session",numBytes,2);
    hashCon.put(hashKey, hashSlot);

  }else if (tokens[4].equals("smtp")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[1],tokens[3],"Smtp Session",numBytes,1);
    hashCon.put(hashKey, hashSlot);

  }else if (tokens[2].equals("5190")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[3],tokens[1],"Chat Session",numBytes,2);
    hashCon.put(hashKey, hashSlot);

  }else if (tokens[4].equals("5190")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[1],tokens[3],"Chat Session",numBytes,1);
    hashCon.put(hashKey, hashSlot);

  }else if (tokens[2].equals("ssh")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[3],tokens[1],"Ssh Session",numBytes,2);
    hashCon.put(hashKey, hashSlot);

  }else if (tokens[4].equals("ssh")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[1],tokens[3],"Ssh Session",numBytes,1);
    hashCon.put(hashKey, hashSlot);

  }else if (tokens[2].equals("telnet")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[3],tokens[1],"Telnet Session",numBytes,2);
    hashCon.put(hashKey, hashSlot);

  }else if (tokens[4].equals("telnet")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[1],tokens[3],"Telnet Session",numBytes,1);
    hashCon.put(hashKey, hashSlot);

  }else if (tokens[2].equals("ftp")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[3],tokens[1],"Ftp Session",numBytes,2);
    hashCon.put(hashKey, hashSlot);

  }else if (tokens[4].equals("ftp")) {

    NetObj hashSlot = new NetObj(crntTime,tokens[1],tokens[3],"Ftp Session",numBytes,1);
    hashCon.put(hashKey, hashSlot);

  }
}

// HASH TABLE REFRESHING
if (crntTime >= (prvTime + REFRESH_RANGE)) {

  Enumeration hashKeys = hashCon.keys();

  for ( ;hashKeys.hasMoreElements() ;) {

    crntKey = (String)hashKeys.nextElement();
    hashEl = (NetObj)hashCon.get(crntKey);
    lastTime = hashEl.getEndTime();
```

```
          crntGap = crntTime - lastTime;

          if (crntGap >= RANGE) {
            hashEl = (NetObj)hashCon.remove(crntKey);
            ntokens[0] = new Timestamp(hashEl.getStartTime()).toString();
            ntokens[1] = new Timestamp(hashEl.getEndTime()).toString();
            ntokens[2] = hashEl.getOrHost();
            ntokens[3] = hashEl.getType();
            ntokens[4] = hashEl.getDstHost();
            ntokens[5] = new Long(hashEl.getVolume()).toString();
            ntokens[6] = new Integer(hashEl.getPackets()).toString();
            ntokens[7] = new Long(hashEl.getMin()).toString();
            ntokens[8] = new Long(hashEl.getMax()).toString();

            avg = (hashEl.getAvgCounter() / Math.max((hashEl.getPackets() - 1),1));
            ntokens[9] = new Double(avg).toString();
            stdDev = Math.sqrt((hashEl.getStdDevCounter()
                     / Math.max((hashEl.getPackets() - 1),1)) - (avg * avg));
            ntokens[10] = new Double(stdDev).toString();

            if (hashEl.getDir() == 3) {

              ntokens[11] = "Two";

            } else {

              ntokens[11] = "One";

            }
            processTokens(ntokens);

          }
        }

        prvTime = prvTime + REFRESH_RANGE;

      }


    } catch (Exception ex) {

      System.out.println("Error: Exception occurred");
      ex.printStackTrace();

    }
  }
}


*******************************************************************************


/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'NetObj' class is used to store Net sessions info
 *
 *
 * File     : NetObj.java
 * Created  : 25/05/2001
 *
```

```
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/


public class NetObj {

  long     startTime = 0;
  long     endTime = 0;
  String   orHost = "";
  String   dstHost = "";
  String   typeCon = "";
  long     volume = 0;
  int      packets = 0;
  long     min = Integer.MAX_VALUE;
  long     max = 0;
  long     avgCounter = 0;
  long     stdDevCounter = 0;
  int      direction = 0;

  public NetObj(long begin, String origin,
                String dest, String type, long bytes, int dir) {
    super();
    startTime = begin;
    endTime = begin;
    typeCon = type;
    orHost = origin;
    dstHost = dest;
    volume = bytes;
    packets = 1;
    min = Integer.MAX_VALUE;
    max = 0;
    avgCounter = 0;
    stdDevCounter = 0;
    direction = dir;
  }

  public long getStartTime() {
    return(startTime);
  }

  public long getEndTime() {
    return(endTime);
  }

  public String getOrHost() {
    return(orHost);
  }

  public String getDstHost() {
    return(dstHost);
  }

  public String getType() {
    return(typeCon);
  }

  public long getVolume() {
    return(volume);
  }
```

```
public int getPackets() {
  return(packets);
}

public long getMin() {
  return(min);
}

public long getMax() {
  return(max);
}

public long getAvgCounter() {
  return(avgCounter);
}

public int getDir() {
  return(direction);
}

public long getStdDevCounter() {
  return(stdDevCounter);
}

public void setStartTime(long start) {
  startTime = start;
}

public void setEndTime(long end) {
  endTime = end;
}

public void setOrHost(String origin) {
  orHost = origin;
}

public void setDstHost(String dest) {
  dstHost = dest;
}

public void setType(String type) {
  typeCon = type;
}

public void setVolume(long bytes) {
  volume = bytes;
}

public void setPackets(int pcks) {
  packets = pcks;
}

public void setMin(long newValue) {
  min = newValue;
}

public void setMax(long newValue) {
  max = newValue;
}

public void setAvgCounter(long newValue) {
  avgCounter = newValue;
}
```

```
  public void setDir(int dir) {
    direction = dir;
  }

  public void setStdDevCounter(long newValue) {
    stdDevCounter = newValue;
  }
}
```

A.2 The WEB Log Aggregator

```
/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'NetLogParser' class generates usage records aggregating
 * Web log entries
 *
 * File     : WebLogParser.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

import java.io.*;
import java.util.*;
import java.sql.*;
import com.ibm.idd.db.*;
import gnu.getopt.Getopt;

import com.ibm.idd.lib.conv.Regex2DB;
import com.ibm.idd.lib.AppFileProperties;

/* Sample Apache log line:
*
* 131.114.4.239 - - [25/Aug/2001:12:05:50 +0200] "GET /foto/portal.jpg HTTP/1.0" 200
*
* 4884 "http://www.di.unipi.it/" "Mozilla/4.73 [en] (X11; U; Linux 2.2.14-5.0smp i686)"
*
*
*/

class WebLogParser{

  static File webLog = null;

  public static void main(String args[]) {


    Connection        dbCon = null;
    BufferedReader    logFile = null;
    String            user = "", host, application, schema, targetHost = "";
    String            server = "", source, recorder, sqlStatement;
    long              time1 = 0;
```

```
try {

  // have to do this to init log4j:
  AppFileProperties.loadProperties();

  // DEFAULTS
  host = java.net.InetAddress.getLocalHost().getHostName();
  application = "Webserver";
  schema = "CADLI";

  // COMMAND LINE PARSING
  Getopt g = new Getopt("CommandLineParser", args, "u:h:a:s:l:");

  int c;

  while ((c = g.getopt()) != -1) {
    switch(c) {
    case 'u':
      user = g.getOptarg();
      break;
    case 'h':
      host = g.getOptarg();
      break;
    case 'a':
      application = g.getOptarg();
      break;
    case 's':
      schema = g.getOptarg();
      break;
    case 'l':
      server = g.getOptarg();
      break;
    case '?':
      break;
    default:
      System.out.println("getopt() returned " + c + "\n");
    }
  }

  recorder = user + " on " + host;
  if (!(server.equals(""))) {
    java.net.InetAddress iAdd = java.net.InetAddress.getByName(server);
    targetHost = iAdd.getHostAddress();
  }
  source = args[g.getOptind()];
  webLog = new File(source);

  // TABLES AND CONNECTIONS CREATION
  TableAccessProperty tap = new TableAccessProperty (DatabaseAccessProperty.getDefault(),
                                                     schema, "USAGE");
  SimpleTableLocalAccess.assertTableCreated(tap);
  dbCon = DatabaseLocalAccess.getConnection();
  dbCon.setAutoCommit(true);

  // LOG PARSING
  if (webLog.exists() == true) {
    FileReader fr = new FileReader(webLog);
    logFile = new BufferedReader(fr);

    // define SQL template
    sqlStatement = "INSERT INTO " + schema + ".USAGE " +
      "VALUES (DEFAULT,'Http Log','" + server + "','" + source +
      "','Http Session','{0}','{1}','{2}','{3}','','" + targetHost + "','{7}','{8}","+
```

```
          "'{4}',{5},{6},{9},{10},{11},{12},'" + recorder + "',DEFAULT)";

    //131.114.4.239 - - [25/Aug/2001:12:05:50 +0200] "GET /foto/portal.jpg HTTP/1.0" 200
    //4884 "http://www.di.unipi.it/" "Mozilla/4.73 [en] (X11; U; Linux 2.2.14-5.0smp i686)"
    // tokens[0] = 131.114.4.239
    // tokens[1] = -
    // tokens[2] = -
    // tokens[3] = 25/Aug/2001:12:05:50 +0200
    // tokens[4] = /foto/portal.jpg
    // tokens[5] = HTTP/1.0
    // tokens[6] = 200
    // tokens[7] = 4884
    // tokens[8] = http://www.di.unipi.it/
    // tokens[9] = Mozilla/4.73 [en] (X11; U; Linux 2.2.14-5.0smp i686)


      String regex = "([.\\w]*) ([-\\w]*) ([-\\w]*) \\[([:/\\w]* \\+\\d{4})\\] \"[\\w]* "+
                     "([~-/_.:\\w]*)(HTTP/[.\\d]*)\" ([-\\d]*) ([-\\d]*) \"(.*)\" \"(.*)\"";

        Regex2DB conv = new Regex2DB(logFile,regex,dbCon,sqlStatement);
        conv.setParsingHandler(new WebParsingHandler());

        // ... and then convert:
        conv.convert();


        // that's it
        System.exit(0);

      }
      else System.out.println("Error: File " + args[0] + " doesn't exist");

    } catch (Exception e) {
      System.out.println("Error: Exception occurred");
      e.printStackTrace();
    } finally {
      if (dbCon != null) {
        try {
          dbCon.close();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
      if (logFile != null) {
        try {
          logFile.close();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
    }
  }
}


*******************************************************************************


/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
```

```
 *
 * The 'WebParsingHandler' class handles Web log entries
 *
 *
 * File     : WebParsingHandler.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/


import java.sql.*;
import java.util.*;
import java.text.SimpleDateFormat;

import com.ibm.idd.lib.parser.BasicParsingHandler;
import com.ibm.idd.lib.parser.GeneratorException;


public class WebParsingHandler extends BasicParsingHandler {

  final  int RANGE = 60000;
  final  int REFRESH_RANGE = RANGE * 1;
  static Hashtable hashCon = new Hashtable();
  static long prvTime = 0;


  public WebParsingHandler() {
    super();
  }

  /**
  * This method is called when a line of text has been
  * successfully parsed. The parsed tokens are passed as
  * an array of String. The line which was
  * parsed is given for information purposes.
  */
  public void processTokens (String line, String[] tokens) throws GeneratorException {

    long      msLong, msTemp, secLong;
    String    msString, secString, hashKey, crntKey;
    long      crntTime, numBytes;
    String[]  ntokens = new String[13];
    WebObj    hashEl;
    long      firstTime, lastTime, crntGap;
    double    avg = 0, stdDev = 0;

    try {
      if ((tokens == null) || (tokens.length == 0)) {
        return;
      }

      SimpleDateFormat format = new SimpleDateFormat("dd/MMM/yyyy:HH:mm:ss zzzz");
      java.util.Date rec = format.parse(tokens[3]);
      if (prvTime == 0) {
        prvTime = rec.getTime();
      }
      crntTime = rec.getTime();

      // the hashKey is: originIP + agent
```

```
hashKey = tokens[0] + tokens[9];
if (tokens[7].equals("-")) tokens[7]="0";
numBytes = new Integer(tokens[7]).intValue();

// if this connection is already present into the hashtable
if (hashCon.containsKey(hashKey)) {
  hashEl = (WebObj)hashCon.remove(hashKey);
  lastTime = hashEl.getEndTime();
  crntGap = crntTime - lastTime;
  // if the time since last packet of this connection is greater than RANGE
  if (crntGap >= RANGE) {
    // I store the correspondant bucket into the DB
    ntokens[0] = new Timestamp(hashEl.getStartTime()).toString();
    ntokens[1] = new Timestamp(lastTime).toString();
    ntokens[2] = hashEl.getOrUser();
    ntokens[3] = hashEl.getOrHost();
    ntokens[4] = hashEl.getDescr();
    ntokens[5] = new Integer(hashEl.getPackets()).toString();
    ntokens[6] = new Long(hashEl.getVolume()).toString();
    ntokens[7] = hashEl.getProtVersion();
    ntokens[8] = hashEl.getFirstUrl();
    if (hashEl.getMin() != RANGE) {
      ntokens[9]=new Long(hashEl.getMin()).toString();
    }
    else {
      ntokens[9]="0";
    }
    ntokens[10]=new Long(hashEl.getMax()).toString();
    avg = (hashEl.getAvgCounter() / Math.max((hashEl.getPackets() - 1),1));
    ntokens[11]=new Double(avg).toString();
    stdDev = Math.sqrt((hashEl.getStdDevCounter() /
                    Math.max((hashEl.getPackets() - 1),1)) - (avg * avg));
    ntokens[12]=new Double(stdDev).toString();
    processTokens(ntokens);

    // and now I store the new packet into the hash table
    hashEl.setStartTime(crntTime);
    hashEl.setEndTime(crntTime);
    hashEl.setDescr(tokens[9]);
    hashEl.setVolume(numBytes);
    hashEl.setPackets(1);
    hashEl.setProtVersion(tokens[5]);
    hashEl.setFirstUrl(tokens[4]);
    hashEl.setMin(RANGE);
    hashEl.setMax(0);
    hashEl.setAvgCounter(0);
    hashEl.setStdDevCounter(0);
    hashCon.put(hashKey, hashEl);
  }
  else {
    //otherwise I just update the current object
    hashEl.setEndTime(crntTime);
    hashEl.setVolume(hashEl.getVolume() + numBytes);
    hashEl.setPackets(hashEl.getPackets() + 1);
    if ((hashEl.getProtVersion()).indexOf(tokens[5]) == -1) {
      hashEl.setProtVersion(hashEl.getProtVersion() + " " + tokens[5]);
    }
    if ((crntGap) < hashEl.getMin()) {
      hashEl.setMin(crntGap);
    }
    if ((crntGap) > hashEl.getMax()) {
      hashEl.setMax(crntGap);
    }
    hashEl.setAvgCounter(hashEl.getAvgCounter() + crntGap);
```

```
          hashEl.setStdDevCounter(hashEl.getStdDevCounter() + (crntGap * crntGap));
          hashCon.put(hashKey, hashEl);
        }
      }
      else { // the connection doesn't exist
        // I insert a new object into the hash table
        WebObj hashSlot =
          new WebObj(crntTime,tokens[0],tokens[1],tokens[5],tokens[4],numBytes,tokens[9]);
        hashCon.put(hashKey, hashSlot);
      }


      if (crntTime >= (prvTime + REFRESH_RANGE)) {
        // refresh hashtable
        Enumeration hashKeys = hashCon.keys();
        for ( ;hashKeys.hasMoreElements() ;) {

          crntKey = (String)hashKeys.nextElement();
          hashEl = (WebObj)hashCon.get(crntKey);
          lastTime = hashEl.getEndTime();
          if ((crntTime - lastTime) >= RANGE) {

            hashEl = (WebObj)hashCon.remove(crntKey);
            ntokens[0] = new Timestamp(hashEl.getStartTime()).toString();
            ntokens[1] = new Timestamp(lastTime).toString();
            ntokens[2] = hashEl.getOrUser();
            ntokens[3] = hashEl.getOrHost();
            ntokens[4] = hashEl.getDescr();
            ntokens[5] = new Integer(hashEl.getPackets()).toString();
            ntokens[6] = new Long(hashEl.getVolume()).toString();
            ntokens[7] = hashEl.getProtVersion();
            ntokens[8] = hashEl.getFirstUrl();
            if (hashEl.getMin() != RANGE) {
              ntokens[9]=new Long(hashEl.getMin()).toString();
            }
            else {
              ntokens[9]="0";
            }
            ntokens[10]=new Long(hashEl.getMax()).toString();
            avg = (hashEl.getAvgCounter() / Math.max((hashEl.getPackets() - 1),1));
            ntokens[11]=new Double(avg).toString();
            stdDev = Math.sqrt((hashEl.getStdDevCounter() /
                        Math.max((hashEl.getPackets() - 1),1)) - (avg * avg));
            ntokens[12]=new Double(stdDev).toString();
            processTokens(ntokens);
          }
        }
        prvTime = prvTime + REFRESH_RANGE;

      }


    } catch (Exception ex) {
      System.out.println("Error: Exception occurred");
      ex.printStackTrace();
    }
  }
}
```

****

```
/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'WebObj' class is used to store Web usage info
 *
 *
 * File     : WebObj.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

public class WebObj {

  long         startTime = 0;
  long         endTime = 0;
  String       orUser = "";
  String       orHost = "";
  String       descr = "";
  int          packets = 0;
  long         volume = 0;
  String       protVersion = "";
  String       firstUrl = "";
  long         min = 60000;
  long         max = 0;
  long         avgCounter = 0;
  long         stdDevCounter = 0;

  public WebObj(long begin, String origin, String user,
                String version, String first, long bytes, String agent) {
    super();
    startTime = begin;
    endTime = begin;
    orUser = user;
    orHost = origin;
    descr = agent;
    packets = 1;
    volume = bytes;
    protVersion = version;
    firstUrl = first;
    min = 60000;
    max = 0;
    avgCounter = 0;
    stdDevCounter = 0;
  }

  public long getStartTime() {
    return(startTime);
  }

  public long getEndTime() {
    return(endTime);
  }

  public String getOrUser() {
    return(orUser);
  }
```

```
public String getOrHost() {
  return(orHost);
}

public String getDescr() {
  return(descr);
}

public long getVolume() {
  return(volume);
}

public int getPackets() {
  return(packets);
}

public String getProtVersion() {
  return(protVersion);
}

public String getFirstUrl() {
  return(firstUrl);
}

public long getMin() {
  return(min);
}

public long getMax() {
  return(max);
}

public long getAvgCounter() {
  return(avgCounter);
}

public long getStdDevCounter() {
  return(stdDevCounter);
}

public void setStartTime(long start) {
  startTime = start;
}

public void setEndTime(long end) {
  endTime = end;
}

public void setOrHost(String origin) {
  orHost = origin;
}

public void setDescr(String agent) {
  descr = agent;
}

public void setVolume(long bytes) {
  volume = bytes;
}

public void setPackets(int pcks) {
  packets = pcks;
}
```

```
  public void setProtVersion(String version) {
    protVersion = version;
  }

  public void setFirstUrl(String first) {
    firstUrl = first;
  }

  public void setMin(long newValue) {
    min = newValue;
  }

  public void setMax(long newValue) {
    max = newValue;
  }

  public void setAvgCounter(long newValue) {
    avgCounter = newValue;
  }

  public void setStdDevCounter(long newValue) {
    stdDevCounter = newValue;
  }
}
```

A.3 The SMTP Log Aggregator

```
/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'SmtpLogParser' class generates usage records aggregating
 * Smtp log entries
 *
 * File     : SmtpLogParser.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

import java.io.*;
import java.util.*;
import java.sql.*;
import com.ibm.idd.db.*;
import gnu.getopt.Getopt;

import com.ibm.idd.lib.conv.Regex2DB;
import com.ibm.idd.lib.AppFileProperties;

/* Sample Sendmail log line:
*
*Aug 25 12:06:31 apis sendmail[19674]: MAA19674: from=<grimaldi@di.unipi.it>,
*size=1891, class=0, pri=31891, nrcpts=1, msgid=<01082512074901.13293@cleopatra.di.unipi.it>,
*bodytype=8BITMIME, proto=SMTP, relay=IDENT:grimaldi@cleopatra [131.114.4.204]
*
```

```
*
*Aug 25 12:06:33 apis sendmail[19676]: MAA19674: to=<mariaciocchetti@yahoo.com>,
*ctladdr=<grimaldi@di.unipi.it> (15774/180), delay=00:00:03, xdelay=00:00:02, mailer=esmtp,
*relay=mx2.mail.yahoo.com. [64.157.4.84], stat=Sent (ok dirdel)*
*
*/

class SmtpLogParser{

  static File logFile = null;

  public static void main(String args[]) {

    Connection        dbCon = null;
    BufferedReader    logReader = null;
    String            user = "", host, application, schema;
    String            server = "",source, recorder,sqlStatement;
    long              time1 = 0;

    try {
      // have to do this to init log4j:
      AppFileProperties.loadProperties();

      // DEFAULTS
      host = java.net.InetAddress.getLocalHost().getHostName();
      application = "Smtp server";
      schema = "CADLI";

      // COMMAND LINE PARSING
      Getopt g = new Getopt("CommandLineParser", args, "u:h:a:s:l:");

      int c;

      while ((c = g.getopt()) != -1) {
        switch(c) {
        case 'u':
          user = g.getOptarg();
          break;
        case 'h':
          host = g.getOptarg();
          break;
        case 'a':
          application = g.getOptarg();
          break;
        case 's':
          schema = g.getOptarg();
          break;
        case 'l':
          server = g.getOptarg();
          break;
        case '?':
          break; // getopt() already printed an error
        default:
          System.out.println("getopt() returned " + c + "\n");
        }
      }

      recorder = user + " on " + host;
      source = args[g.getOptind()];
      logFile = new File(source);

      // TABLES AND CONNECTIONS CREATION
      TableAccessProperty tap = new TableAccessProperty (DatabaseAccessProperty.getDefault(),
                                    schema, "USAGE");
```

```
      SimpleTableLocalAccess.assertTableCreated(tap);
      dbCon = DatabaseLocalAccess.getConnection();
      dbCon.setAutoCommit(true);

      // LOG PARSING
      if (logFile.exists() == true) {
        FileReader fr = new FileReader(logFile);
        logReader = new BufferedReader(fr);

        // define SQL template
        sqlStatement = "INSERT INTO " + schema + ".USAGE " +
          "VALUES (DEFAULT,'Smtp Log','" + server +"','" + source + "','{0}'," +
          "'{1}','{1}','{2}','{3}','{4}','{5}','{6}','{7}',"+
          "'',{8},{9},0,0,0,0,'" + recorder + "',DEFAULT)";

          String regex = "([.\\w]*)[\\s]*([\\d]*) ([:\\d]*) [\\w]* [\\[\\]\\w]*: ([\\w]*): " +
          "(?:from=([\\S]*), size=([\\d]*), [-=\\w]*, [=\\w]*, " +
          "nrcpts=([\\d]*), msgid=<([\\S]*)>, (?:[\\s\\S\\^,]*, )?[=\\w]*, relay=(?:[\\S]* " +
          ")?\\[([.\\d]*)\\].*)?" +
          "(?:to=([\\S]*), (?:[\\s\\S\\^,]*, )?delay=([:\\+\\d]*), xdelay=([:\\d]*), " +
          "[=\\w]*, (?:relay=(?:[\\S]* )?\\[([.\\d]*)\\], )?stat=Sent.*)?";



        Regex2DB conv = new Regex2DB(logReader,regex,dbCon,sqlStatement);
        conv.setParsingHandler(new SmtpParsingHandler());

        // ... and then convert:
        conv.convert();


        // that's it
        System.exit(0);

      }
      else System.out.println("Error: File " + args[0] + " doesn't exist");

    } catch (Exception e) {
      System.out.println("Error: Exception occurred");
      e.printStackTrace();
    } finally {
      if (dbCon != null) {
        try {
          dbCon.close();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
      if (logReader != null) {
        try {
          logReader.close();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
    }
  }
}
```

****

```
/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'SmtpParsingHandler' class handles Smtp log entries
 *
 *
 * File     : SmtpParsingHandler.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/


import java.sql.*;
import java.util.*;
import java.text.SimpleDateFormat;

import com.ibm.idd.lib.parser.BasicParsingHandler;
import com.ibm.idd.lib.parser.GeneratorException;


public class SmtpParsingHandler extends BasicParsingHandler {


  static int numParsed = 0;  //for profiling


  public SmtpParsingHandler() {
    super();
  }

  /**
  * This method is called when a line of text has been
  * successfully parsed. The parsed tokens are passed as
  * an array of String. The line which was
  * parsed is given for information purposes.
  *
  * Aug 25 12:06:31 apis sendmail[19674]: MAA19674: from=<grimaldi@di.unipi.it>, size=1891,
  *
  * class=0, pri=31891, nrcpts=1, msgid=<01082512074901.13293@cleopatra.di.unipi.it>,
  *
  * bodytype=8BITMIME, proto=SMTP, relay=IDENT:grimaldi@cleopatra [131.114.4.204]
  *
  * tokens[0] = Aug                                          -> Month
  * tokens[1] = 25                                           -> Day
  * tokens[2] = 12:06:31                                     -> Time
  * tokens[3] = MAA19674                                     -> localRef
  * tokens[4] = <grimaldi@di.unipi.it                        -> sender
  * tokens[5] = 1891                                         -> size
  * tokens[6] = 1                                            -> nrcpts
  * tokens[7] = 01082512074901.13293@cleopatra.di.unipi.it   -> msgID
  * tokens[8] = 131.114.4.204                                -> userHost
  * tokens[9] = null                                         -> recipient
  * tokens[10] = null                                        -> delay
  * tokens[11] = null                                        -> xdelay
  * tokens[12] = null                                        -> relay
```

```
*
*
* Aug 25 12:06:33 apis sendmail[19676]: MAA19674: to=<mariaciocchetti@yahoo.com>,
*
* ctladdr=<grimaldi@di.unipi.it> (15774/180), delay=00:00:03, xdelay=00:00:02,
*
* mailer=esmtp, relay=mx2.mail.yahoo.com. [64.157.4.84], stat=Sent (ok dirdel)*
*
* tokens[0] = Aug                                          -> Month
* tokens[1] = 25                                           -> Day
* tokens[2] = 12:06:33                                     -> Time
* tokens[3] = MAA19674                                     -> localRef
* tokens[4] = null                                         -> sender
* tokens[5] = null                                         -> size
* tokens[6] = null                                         -> nrcpts
* tokens[7] = null                                         -> msgID
* tokens[8] = null                                         -> userHost
* tokens[9] = mariaciocchetti@yahoo.com                    -> recipient
* tokens[10] = 00:00:03                                     -> delay
* tokens[11] = 00:00:02                                     -> xdelay
* tokens[12] = 64.157.4.8                                   -> relay
*
*
*/

public void processTokens (String line, String[] tokens) throws GeneratorException {

  String[]      ntokens = new String[10];
  String        yearString = "2001";                      // To be generalized
  StringBuffer  tmstBuf = new StringBuffer(yearString);

  try {
    if ((tokens == null) || (tokens.length == 0)) {
      return;
    }

    // I start creating an sql timestamp for this entry
    if (tokens[1].length() == 1) {
      //day must have two digits
      tokens[1] = "0" + tokens[1];
    }
    for (int i=0; i<=2; i++) {
      tmstBuf.append(tokens[i]);
    }
    SimpleDateFormat format = new SimpleDateFormat("yyyyMMMddHH:mm:ss");
    java.util.Date rec = format.parse(tmstBuf.toString());
    ntokens[1] = new Timestamp(rec.getTime()).toString();

    if (tokens[4] != null) {
      // It's a "from=...." line
      if (tokens[4].startsWith("<")) {
        ntokens[2] = tokens[4].substring(1,tokens[4].length()-1);
      }
      else {
        ntokens[2] = tokens[4];
      }
      if (tokens[8] != null) {
        ntokens[3] = tokens[8];
      }
      else {
        ntokens[3] = "";
      }
      ntokens[4] = "";
      ntokens[5] = "";
```

```
    ntokens[6] = tokens[7];
    ntokens[7] = tokens[3];
    ntokens[8] = tokens[6];
    ntokens[9] = tokens[5];

    if ((ntokens[2].toLowerCase()).endsWith("@di.unipi.it")) {
      ntokens[0] = "Message Sending";
      ntokens[2] =
        ntokens[2].substring(0,(ntokens[2].toLowerCase()).indexOf("@di.unipi.it"));
    }
    else {
      ntokens[0] = "Message Receiving";
    }
    processTokens(ntokens);

  }
  else if (tokens[9] != null) {
    // tokens[8] general form is <a@z>,<b@z>, ...,<k@z>,
    while(tokens[9].indexOf(">,<") != -1) {

      ntokens[0] = "Remote Forwarding";
      ntokens[2] = "";
      ntokens[3] = "";
      ntokens[4] = tokens[9].substring(1,tokens[9].indexOf(">,<"));
      if (tokens[12] != null) {
        ntokens[5] = tokens[12];
      }
      else {
        ntokens[5] = "";
      }
      ntokens[6] = "";
      ntokens[7] = tokens[3];
      ntokens[8] = "0";
      ntokens[9] = "0";
      processTokens(ntokens);
      tokens[9] = tokens[9].substring(tokens[9].indexOf(">,<") + 2, tokens[9].length());
    }

    if (tokens[9].startsWith("<")) {
      ntokens[4] = tokens[9].substring(1,tokens[9].length()-1);
    }
    else {
      ntokens[4] = tokens[9];
    }
    if ((ntokens[4].toLowerCase()).endsWith("@di.unipi.it")) {
      ntokens[0] = "Local Forwarding";
      ntokens[4] =
         ntokens[4].substring(0,(ntokens[4].toLowerCase()).indexOf("@di.unipi.it"));
    }
    else {
      ntokens[0] = "Remote Forwarding";
    }
    ntokens[2] = "";
    ntokens[3] = "";
    if (tokens[12] != null) {
      ntokens[5] = tokens[12];
    }
    else {
      ntokens[5] = "";
    }
    ntokens[6] = "";
    ntokens[7] = tokens[3];
    ntokens[8] = "0";
    ntokens[9] = "0";
```

```
        processTokens(ntokens);

      }
      else return;


    } catch (Exception ex) {
      System.out.println("Error: Exception occurred");
      ex.printStackTrace();
    }
  }
}
```

A.4 The POP Log Aggregator

```
/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'PopLogParser' class generates usage records aggregating
 * Pop log entries
 *
 * File     : PopLogParser.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

import java.io.*;
import java.util.*;
import java.sql.*;
import com.ibm.idd.db.*;
import gnu.getopt.Getopt;

import com.ibm.idd.lib.conv.Regex2DB;
import com.ibm.idd.lib.AppFileProperties;

/* Sample Popd log line:
*
*Aug 25 12:01:05 apis ipop3d[19490]: pop3 service init from 131.114.2.9x
*Aug 25 12:01:05 apis ipop3d[19490]: Login user=pippo host=capraia [131.114.2.9x] nmsgs=0/0
*Aug 25 12:01:05 apis ipop3d[19490]: Logout user=pippo host=capraia [131.114.2.9x] nmsgs=0
*ndele=0
*
*
*/

class PopLogParser {

  static File logFile = null;

  public static void main(String args[]) {
```

```
Connection        dbCon = null;
BufferedReader    logReader = null;
String            user = "", host, application, schema;
String            server = "",source,recorder,sqlStatement;


try {

  // have to do this to init log4j:
  AppFileProperties.loadProperties();

  // DEFAULTS
  host = java.net.InetAddress.getLocalHost().getHostName();
  application = "Smtp server";
  schema = "CADLI";

  // COMMAND LINE PARSING
  Getopt g = new Getopt("CommandLineParser", args, "u:h:a:s:l:");

  int c;

  while ((c = g.getopt()) != -1) {
    switch(c) {
    case 'u':
      user = g.getOptarg();
      break;
    case 'h':
      host = g.getOptarg();
      break;
    case 'a':
      application = g.getOptarg();
      break;
    case 's':
      schema = g.getOptarg();
      break;
    case 'l':
      server = g.getOptarg();
      break;
    case '?':
      break;
    default:
      System.out.println("getopt() returned " + c + "\n");
    }
  }

  recorder = user + " on " + host;
  source = args[g.getOptind()];
  logFile = new File(source);

  // TABLES AND CONNECTIONS CREATION
  TableAccessProperty tap = new TableAccessProperty (DatabaseAccessProperty.getDefault(),
                                                     schema, "USAGE");
  SimpleTableLocalAccess.assertTableCreated(tap);
  dbCon = DatabaseLocalAccess.getConnection();
  dbCon.setAutoCommit(true);

  // LOG PARSING
  if (logFile.exists() == true) {
    FileReader fr = new FileReader(logFile);
    logReader = new BufferedReader(fr);

    // define SQL template
    sqlStatement = "INSERT INTO " + schema + ".USAGE " +
```

```
          "VALUES (DEFAULT,'Pop Log','" + server + "','" + source +
          "','Pop Session','{0}','{1}','{2}','{3}','','','','','" +
          "',{4},{5},{6},{7},{8},{9},'" + recorder + "',DEFAULT)";


        String regex = "([.\\w]*)[\\s]*([\\d]*) ([:\\d]*) [\\w]* [\\S]*\\[([\\d]*)\\]: " +
          "(?:pop3 service init from ([.\\d]*))?" +
          "(?:Login user=([\\S]*) host=([\\S]*) \\[([.\\d]*)\\] nmsgs=([\\d]*)/([\\d]*))?" +
          "(?:Auth user=([\\S]*) host=([\\S]*) \\[([.\\d]*)\\] nmsgs=([\\d]*)/([\\d]*))?" +
          "(?:Logout user=([\\S]*) host=([\\S]*) \\[([.\\d]*)\\] nmsgs=([\\d]*) " +
          "ndele=([\\d]*))?";

        Regex2DB conv = new Regex2DB(logReader,regex,dbCon,sqlStatement);
        conv.setParsingHandler(new PopParsingHandler());

        // ... and then convert:
        conv.convert();


        // that's it
        System.exit(0);

      }
      else System.out.println("Error: File " + args[0] + " doesn't exist");

    } catch (Exception e) {
      System.out.println("Error: Exception occurred");
      e.printStackTrace();
    } finally {
      if (dbCon != null) {
        try {
          dbCon.close();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
      if (logReader != null) {
        try {
          logReader.close();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
    }
  }
}


*******************************************************************************


/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'PopParsingHandler' class handles Pop log entries
 *
 *
 * File     : PopParsingHandler.java
 * Created  : 25/05/2001
```

```
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

import java.sql.*;
import java.util.*;
import java.text.SimpleDateFormat;

import com.ibm.idd.lib.parser.BasicParsingHandler;
import com.ibm.idd.lib.parser.GeneratorException;


public class PopParsingHandler extends BasicParsingHandler {

  final  int           REFRESH_RANGE = 300000;
  static int           numParsed = 0;  //for profiling
  static int           numHandled = 0; //for profiling
  static Hashtable     hashCon = new Hashtable();
  static long          prvTime = 0;

  public PopParsingHandler() {
    super();
  }

  /**
  * This method is called when a line of text has been
  * successfully parsed. The parsed tokens are passed as
  * an array of String. The line which was
  * parsed is given for information purposes.
  */

  public void processTokens (String line, String[] tokens) throws GeneratorException {

    String[]      ntokens = new String[10];
    String        yearString = "2001";                      // To be generalized
    StringBuffer  tmstBuf = new StringBuffer(yearString);
    long          crntTime = 0;
    String        hashKey = "";
    PopObj        hashEl = null;
    long          gap,min,max;
    double        avg,stdDev;

    try {
      if ((tokens == null) || (tokens.length == 0)) {
        return;
      }


      // I start creating an sql timestamp for this entry
      if (tokens[1].length() == 1) {
        //day must have two digits
        tokens[1] = "0" + tokens[1];
      }
      for (int i=0; i<=2; i++) {
        tmstBuf.append(tokens[i]);
      }
      SimpleDateFormat format = new SimpleDateFormat("yyyyMMMddHH:mm:ss");
      java.util.Date rec = format.parse(tmstBuf.toString());
```

```
if (prvTime == 0) {
  prvTime = rec.getTime();
}
crntTime = rec.getTime();

hashKey = tokens[3];
if (!(tokens[4] == null)) {
  // It's an Init entry
  if (hashCon.containsKey(hashKey)) {
    hashEl = (PopObj)hashCon.remove(hashKey);
    hashEl.setStartTime(crntTime);
    hashEl.setEndTime(crntTime);
    hashEl.setHostIp(tokens[4]);
    hashCon.put(hashKey, hashEl);
  }
  else {
    hashEl = new PopObj(crntTime,tokens[4]);
    hashCon.put(hashKey, hashEl);
  }
}
else if (!(tokens[5] == null)) {
  //It's a Login entry
  if (hashCon.containsKey(hashKey)) {
    hashEl = (PopObj)hashCon.remove(hashKey);
    gap = crntTime - hashEl.getEndTime();
    hashEl.setEndTime(crntTime);
    hashEl.setHostIp(tokens[7]);
    hashEl.setUser(tokens[5]);
    hashEl.setNmsgs(tokens[8]);
    hashEl.setMin(gap);
    hashEl.setMax(gap);
    hashEl.setAvgCounter(gap);
    hashEl.setStdDevCounter(gap * gap);
    hashCon.put(hashKey, hashEl);
  }
  else {
    hashEl = new PopObj(crntTime,tokens[5],tokens[7],tokens[8]);
    hashCon.put(hashKey, hashEl);
  }
}
else if (!(tokens[10] == null)) {
  //it's an Auth log entry
  if (hashCon.containsKey(hashKey)) {
    hashEl = (PopObj)hashCon.remove(hashKey);
    gap = crntTime - hashEl.getEndTime();
    hashEl.setEndTime(crntTime);
    hashEl.setHostIp(tokens[12]);
    hashEl.setUser(tokens[10]);
    hashEl.setNmsgs(tokens[13]);
    hashEl.setMin(gap);
    hashEl.setMax(gap);
    hashEl.setAvgCounter(gap);
    hashEl.setStdDevCounter(gap * gap);
    hashCon.put(hashKey, hashEl);
  }
  else {
    hashEl = new PopObj(crntTime,tokens[10],tokens[12],tokens[13]);
    hashCon.put(hashKey, hashEl);
  }
}
else if (!(tokens[15] == null)) {
  //It's a logout entry
  if (hashCon.containsKey(hashKey)) {
    hashEl = (PopObj)hashCon.remove(hashKey);;
```

```
            ntokens[0] = (new Timestamp(hashEl.getStartTime())).toString();
            ntokens[1] = (new Timestamp(crntTime)).toString();
            ntokens[2] = tokens[15];
            ntokens[3] = tokens[17];
            ntokens[4] = tokens[18];
            ntokens[5] = tokens[19];
            gap = crntTime - hashEl.getEndTime();
            min = Math.min(gap,hashEl.getMin());
            ntokens[6] = (new Long(min)).toString();
            max = Math.max(gap,hashEl.getMax());
            ntokens[7] = (new Long(max)).toString();
            avg = (hashEl.getAvgCounter() + gap) / 2;
            ntokens[8] = (new Double(avg)).toString();
            stdDev = Math.sqrt(((hashEl.getStdDevCounter() + (gap * gap))/2) - (avg * avg));
            ntokens[9] = (new Double(stdDev)).toString();
            processTokens(ntokens);
          }
          else {
            ntokens[0] = (new Timestamp(crntTime)).toString();
            ntokens[1] = (new Timestamp(crntTime)).toString();
            ntokens[2] = tokens[15];
            ntokens[3] = tokens[17];
            ntokens[4] = tokens[18];
            ntokens[5] = tokens[19];
            ntokens[6] = "-1";
            ntokens[7] = "-1";
            ntokens[8] = "-1";
            ntokens[9] = "-1";
            processTokens(ntokens);
          }
        }
        else return;

        if (crntTime >= (prvTime + REFRESH_RANGE)) {
          // refresh hashtable
          System.out.println("***** HASH TABLE REFRESHING *****");
          Enumeration hashKeys = hashCon.keys();
          for ( ;hashKeys.hasMoreElements() ;) {
            hashKey = (String)hashKeys.nextElement();
            hashEl = (PopObj)hashCon.get(hashKey);
            if (!(hashEl.getNdel().equals(""))) {
              hashEl = (PopObj)hashCon.remove(hashKey);
              ntokens[0] = (new Timestamp(hashEl.getStartTime())).toString();
              ntokens[1] = (new Timestamp(hashEl.getEndTime())).toString();
              ntokens[2] = hashEl.getUser();
              ntokens[3] = hashEl.getHostIp();
              ntokens[4] = hashEl.getNmsgs();
              ntokens[5] = hashEl.getNdel();
              ntokens[6] = (new Long(hashEl.getMin())).toString();
              ntokens[7] = (new Long(hashEl.getMax())).toString();
              ntokens[8] = (new Double(hashEl.getAvgCounter())).toString();
              ntokens[9] = "0";
              processTokens(ntokens);
            }
          }
          prvTime = prvTime + REFRESH_RANGE;
        }

      } catch (Exception ex) {
        System.out.println("Error: Exception occurred");
        ex.printStackTrace();
      }
    }
  }
```

```
*******************************************************************************

/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'PopObj' class is used to store Pop usage info
 *
 *
 * File     : PopObj.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

public class PopObj {

  long          startTime = 0;
  long          endTime = 0;
  String        userName = "";
  String        hostIp = "";
  String        nmsgs = "";
  String        deleted = "";
  long          min = Long.MAX_VALUE;
  long          max = 0;
  long          avgCounter = 0;
  long          stdDevCounter = 0;

  public PopObj(long begin, String fromHost) {
    super();
    startTime = begin;
    endTime = begin;
    hostIp = fromHost;
    min = Long.MAX_VALUE;
    max = 0;
    avgCounter = 0;
    stdDevCounter = 0;
  }

  public PopObj(long begin, String fromUser, String fromHost, String numMsgs) {
    super();
    startTime = begin;
    endTime = begin;
    userName = fromUser;
    hostIp = fromHost;
    nmsgs = numMsgs;
    min = Long.MAX_VALUE;
    max = 0;
    avgCounter = 0;
    stdDevCounter = 0;
  }

  public PopObj(long begin, String fromUser, String fromHost,
                            String numMsgs, String numDeleted) {
    super();
```

```
  startTime = begin;
  endTime = begin;
  userName = fromUser;
  hostIp = fromHost;
  nmsgs = numMsgs;
  deleted = numDeleted;
  min = Long.MAX_VALUE;
  max = 0;
  avgCounter = 0;
  stdDevCounter = 0;
}

public long getStartTime() {
  return(startTime);
}

public long getEndTime() {
  return(endTime);
}

public String getUser() {
  return(userName);
}

public String getHostIp() {
  return(hostIp);
}

public String getNmsgs() {
  return(nmsgs);
}

public String getNdel() {
  return(deleted);
}

public long getMin() {
  return(min);
}

public long getMax() {
  return(max);
}

public long getAvgCounter() {
  return(avgCounter);
}

public long getStdDevCounter() {
  return(stdDevCounter);
}

public void setStartTime(long start) {
  startTime = start;
}

public void setEndTime(long end) {
  endTime = end;
}

public void setUser(String user) {
  userName = user;
}
```

```
  public void setHostIp(String host) {
    hostIp = host;
  }

  public void setNmsgs(String numMsgs) {
    nmsgs = numMsgs;
  }

  public void setNdel(String ndel) {
    deleted = ndel;
  }

  public void setMin(long newValue) {
    min = newValue;
  }

  public void setMax(long newValue) {
    max = newValue;
  }

  public void setAvgCounter(long newValue) {
    avgCounter = newValue;
  }

  public void setStdDevCounter(long newValue) {
    stdDevCounter = newValue;
  }
}
```

A.5 The IMAP Log Aggregator

```
/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'ImapLogParser' class generates usage records aggregating
 * Imap log entries
 *
 * File     : ImapLogParser.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

import java.io.*;
import java.util.*;
import java.sql.*;
import com.ibm.idd.db.*;
import gnu.getopt.Getopt;

import com.ibm.idd.lib.conv.Regex2DB;
import com.ibm.idd.lib.AppFileProperties;

/* Sample Popd log line:
```

```
*
*Aug 25 12:01:05 apis imapd[19490]: imap service init from 131.114.2.9x
*Aug 25 12:01:05 apis imapd[19490]: Login user=verdi host=pc-verdi [131.114.2.9x]
*Aug 25 12:01:05 apis imapd[19490]: Logout user=verdi host=pc-verdi [131.114.2.9x]
*
*/

class ImapLogParser{

  static File logFile = null;

  public static void main(String args[]){

    Connection        dbCon = null;
    BufferedReader    logReader = null;
    String            source, application = "Imap Server";
    String            server = "", sqlStatement, schema = "CADLI";
    String            recorder, localHostName, user = "";

  try {

    // DEFAULTS
    localHostName = java.net.InetAddress.getLocalHost().getHostName();

    // COMMAND LINE PARSING
    Getopt g = new Getopt("CommandLineParser", args, "u:h:a:s:l:");

    int c;

    while ((c = g.getopt()) != -1) {
      switch(c) {
        case 'u':
          user = g.getOptarg();
          break;
        case 'h':
          localHostName = g.getOptarg();
          break;
        case 'a':
          application = g.getOptarg();
          break;
        case 's':
          schema = g.getOptarg();
          break;
        case 'l':
          server = g.getOptarg();
          break;
        case '?':
          break;
        default:
          System.out.println("getopt() returned " + c + "\n");
      }
    }

    if (user.equals(""))  {
      recorder = localHostName;
    } else  {
      recorder = user + " on " + localHostName;
    }
    source = args[g.getOptind()];
    logFile = new File(source);

    // TABLES AND CONNECTIONS CREATION
    TableAccessProperty tap = new TableAccessProperty
             (DatabaseAccessProperty.getDefault(), schema, "USAGE");
```

```
    SimpleTableLocalAccess.assertTableCreated(tap);
    dbCon = DatabaseLocalAccess.getConnection();
    dbCon.setAutoCommit(true);

    // LOG PARSING
    if (logFile.exists() == true){
      FileReader fr = new FileReader(logFile);
      logReader = new BufferedReader(fr);

      // define SQL template
      sqlStatement =
           "INSERT INTO " + schema + ".USAGE " +
           "VALUES  (DEFAULT,'Imap Log','" + server +"','" + source +
           "','Imap Session','{0}','{1}','{2}','{3}','','','','',"+
           "'',-1,-1,{4},{5},{6},{7},'" + recorder + "',DEFAULT)";


      String regex =
           "([.\\w]*)[\\s]*([\\d]*) ([:\\d]*) [\\w]* [\\S]*\\[([\\d]*)\\]: " +
           "(?:imap service init from ([.\\d]*))?" +
           "(?:Login user=([\\S]*) host=([\\S]*) \\[([.\\d]*)\\])?" +
           "(?:Authenticated user=([\\S]*) host=([\\S]*) \\[([.\\d]*)\\])?" +
           "(?:Logout user=([\\S]*) host=([\\S]*) \\[([.\\d]*)\\])?";

      Regex2DB conv = new Regex2DB(logReader,regex,dbCon,sqlStatement);
      conv.setParsingHandler(new ImapParsingHandler());

      // ...and then convert:
      conv.convert();


      // that's it
      System.exit(0);
    }

    else System.out.println("Error: File " + args[0] + " doesn't exist");

  }catch (Exception e){

    System.out.println("Error: Exception occurred");
    e.printStackTrace();

  }finally {
    if (dbCon != null) {
      try {
        dbCon.close();
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }
    if (logReader != null) {
      try {
        logReader.close();
      } catch (Exception ex) {
        ex.printStackTrace();
      }
    }
  }
```

****

```
/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'ImapParsingHandler' class handles Imap log entries
 *
 *
 * File     : ImapParsingHandler.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/

import java.sql.*;
import java.util.*;
import java.text.SimpleDateFormat;
import com.ibm.idd.lib.parser.BasicParsingHandler;
import com.ibm.idd.lib.parser.GeneratorException;


public class ImapParsingHandler extends BasicParsingHandler {

  static Hashtable     hashCon = new Hashtable();
  static long          prvTime = 0;

  public ImapParsingHandler() {
    super();
  }

  /**
  * This method is called when a line of text has been
  * successfully parsed. The parsed tokens are passed as
  * an array of String. The line which was
  * parsed is given for information purposes.
  */

public void processTokens (String line, String[] tokens) throws GeneratorException {

    String[]      ntokens = new String[8];
    String        yearString = "2001";         // To be generalized
    StringBuffer  tmstBuf = new StringBuffer(yearString);
    long          crntTime = 0;
    String        hashKey = "";
    ImapObj       hashEl = null;
    long          gap,min,max;
    double        avg,stdDev;

    try {


      if ((tokens == null) || (tokens.length == 0)) {
        return;
      };

      // I start creating an sql timestamp for this entry
      if (tokens[1].length() == 1) {
        //day must have two digits
        tokens[1] = "0" + tokens[1];
      }
```

```
for (int i=0; i<=2; i++) {
  tmstBuf.append(tokens[i]);
}
SimpleDateFormat format = new SimpleDateFormat("yyyyMMMddHH:mm:ss");
java.util.Date rec = format.parse(tmstBuf.toString());
if (prvTime == 0) {
  prvTime = rec.getTime();
}
crntTime = rec.getTime();

hashKey = tokens[3];
if (!(tokens[4] == null)) {
  // It's an Init entry
  if (hashCon.containsKey(hashKey)) {
    hashEl = (ImapObj)hashCon.remove(hashKey);
    hashEl.setEndTime(crntTime);
    hashEl.setHostIp(tokens[4]);
    hashCon.put(hashKey, hashEl);
  }
  else {
    hashEl = new ImapObj(crntTime,tokens[4]);
    hashCon.put(hashKey, hashEl);
  }
}
else if (!(tokens[5] == null)) {
  //It's a Login entry
  if (hashCon.containsKey(hashKey)) {
    hashEl = (ImapObj)hashCon.remove(hashKey);
    gap = crntTime - hashEl.getEndTime();
    hashEl.setEndTime(crntTime);
    hashEl.setHostIp(tokens[7]);
    hashEl.setUser(tokens[5]);
    hashEl.setMin(gap);
    hashEl.setMax(gap);
    hashEl.setAvgCounter(gap);
    hashEl.setStdDevCounter(gap * gap);
    hashCon.put(hashKey, hashEl);
  }
  else {
    hashEl = new ImapObj(crntTime,tokens[5],tokens[7]);
    hashCon.put(hashKey, hashEl);
  }
}
else if (!(tokens[8] == null)) {
  //it's an Auth log entry
  if (hashCon.containsKey(hashKey)) {
    hashEl = (ImapObj)hashCon.remove(hashKey);
    gap = crntTime - hashEl.getEndTime();
    hashEl.setEndTime(crntTime);
    hashEl.setHostIp(tokens[10]);
    hashEl.setUser(tokens[8]);
    hashEl.setMin(gap);
    hashEl.setMax(gap);
    hashEl.setAvgCounter(gap);
    hashEl.setStdDevCounter(gap * gap);
    hashCon.put(hashKey, hashEl);
  }
  else {
    hashEl = new ImapObj(crntTime,tokens[8],tokens[10]);
    hashCon.put(hashKey, hashEl);
  }
}
else if (!(tokens[11] == null)) {
  //It's a logout entry
```

```
        if (hashCon.containsKey(hashKey)) {
          hashEl = (ImapObj)hashCon.remove(hashKey);
          ntokens[0] = (new Timestamp(hashEl.getStartTime())).toString();
          ntokens[1] = (new Timestamp(crntTime)).toString();
          ntokens[2] = tokens[11];
          ntokens[3] = tokens[13];
          gap = crntTime - hashEl.getEndTime();
          min = Math.min(gap,hashEl.getMin());
          ntokens[4] = (new Long(min)).toString();
          max = Math.max(gap,hashEl.getMax());
          ntokens[5] = (new Long(max)).toString();
          avg = (hashEl.getAvgCounter() + gap) / 2;
          ntokens[6] = (new Double(avg)).toString();
          stdDev = Math.sqrt(((hashEl.getStdDevCounter() + (gap * gap))/2) - (avg * avg));
          ntokens[7] = (new Double(stdDev)).toString();
          processTokens(ntokens);
        }
        else {
          ntokens[0] = (new Timestamp(crntTime)).toString();
          ntokens[1] = (new Timestamp(crntTime)).toString();
          ntokens[2] = tokens[11];
          ntokens[3] = tokens[13];
          ntokens[4] = "-1";
          ntokens[5] = "-1";
          ntokens[6] = "-1";
          ntokens[7] = "-1";
          processTokens(ntokens);
        }
      }
      else return;

    } catch (Exception ex) {
      System.out.println("Error: Exception occurred");
      ex.printStackTrace();
    }
  }
}



*******************************************************************************


/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'ImapObj' class is used to store Imap usage info
 *
 *
 * File     : ImapObj.java
 * Created  : 25/05/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/
```

```
public class ImapObj {

  long          startTime = 0;
  long          endTime = 0;
  String        userName = "";
  String        hostIp = "";
  long          min = Long.MAX_VALUE;
  long          max = 0;
  long          avgCounter = 0;
  long          stdDevCounter = 0;


  public ImapObj(long begin, String fromHost) {
    super();
    startTime = begin;
    endTime = begin;
    hostIp = fromHost;
    long min = Long.MAX_VALUE;
    long max = 0;
    long avgCounter = 0;
    long stdDevCounter = 0;
  }

  public ImapObj(long begin, String fromUser, String fromHost) {
    super();
    startTime = begin;
    endTime = begin;
    userName = fromUser;
    hostIp = fromHost;
    long min = Long.MAX_VALUE;
    long max = 0;
    long avgCounter = 0;
    long stdDevCounter = 0;
  }


  public long getStartTime() {
    return(startTime);
  }

  public long getEndTime() {
    return(endTime);
  }

  public String getUser() {
    return(userName);
  }

  public String getHostIp() {
    return(hostIp);
  }

  public long getMin() {
    return(min);
  }

  public long getMax() {
    return(max);
  }

  public long getAvgCounter() {
    return(avgCounter);
  }
```

```
  public long getStdDevCounter() {
    return(stdDevCounter);
  }

  public void setStartTime(long start) {
    startTime = start;
  }

  public void setEndTime(long end) {
    endTime = end;
  }

  public void setUser(String user) {
    userName = user;
  }

  public void setHostIp(String host) {
    hostIp = host;
  }

  public void setMin(long newValue) {
    min = newValue;
  }

  public void setMax(long newValue) {
    max = newValue;
  }

  public void setAvgCounter(long newValue) {
    avgCounter = newValue;
  }

  public void setStdDevCounter(long newValue) {
    stdDevCounter = newValue;
  }

}
```

Appendix B

THE ACTIVITIES BUILDER

```
/**
 *
 * © Copyright International Business Machines Corporation 2000, 2001.
 * All rights reserved.
 *
 * The 'ActivitiesBuilder' class generates activities starting from
 * the Usage Table
 *
 * File     : ActivitiesBuilder.java
 * Created  : 01/07/2001
 *
 * @author    Milco Filoni  (mfi@zurich.ibm.com)
 * @version   1.00, 25/11/2001
 * @since     JDK 1.3
 *
 *
 **/


import java.io.*;
import java.util.*;
import java.sql.*;
import com.ibm.idd.db.*;
import gnu.getopt.Getopt;

import com.ibm.idd.lib.AppFileProperties;

public class ActivitiesBuilder {

  static Connection        dbCon = null;
  static String            schema = "CADLI";
  static String            recorder = "";

  public static void BuildLocalWebActivities() throws java.sql.SQLException {

    Timestamp   startTime = null, endTime = null;
    String      sqlStatement0 = null, sqlStatement1 = null, sqlStatement2 = null;
    String      crntHost,agent,targetHost = null,descr = null;
    int         activityGap = 3600000; // 1 hour
    int         packets = 0,volume = 0;
    int         min = Integer.MAX_VALUE,max = -1,numUsages = 0;
    long        crntTime = 0,prvTime = 0,crntGap = 0;
    long        avgCounter = 0,stdDevCounter = 0;
    double      avg = 0, stdDev = 0;
    boolean     flag = false;

    System.out.println("******* LocalWebActivities **********");

    sqlStatement0 = "INSERT INTO " + schema + ".activities " +
      "(RecordType,StartTime,EndTime," +
```

```
  " InitiatingUser,InitiatingHost,Target," +
  " Description,NumUsages,Data1,Data2,Data3," +
  " Min,Max,Average,StdDev,Recorder)" +
  " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
PreparedStatement pstmt = dbCon.prepareStatement(sqlStatement0);

Statement stmt1 = dbCon.createStatement();
Statement stmt2 = dbCon.createStatement();

// I first select all the local hosts that have
// been seen at least in one usage record
sqlStatement1 = "SELECT DISTINCT InitiatingHost " +
  "FROM " + schema + ".usage " +
  "WHERE (InitiatingHost LIKE '131.114.2.%' " +
  "OR InitiatingHost LIKE '131.114.4.%') " +
  "AND RecordType='Http Session'";


ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
while (rs1.next() ) {

  crntHost = rs1.getString(1);
  System.out.println("crntHost=" + crntHost);

  // then for each of such hosts I select all usage records
  // that refer to valid ('Two' directions and with DataVolume>0)
  // web activities (from both net and web log)
  sqlStatement2 =
    "SELECT StartTime,EndTime,Description,DataPackets,DataVolume,TargetHost,Category "+
    "FROM " + schema + ".usage " +
    "WHERE InitiatingHost='" + crntHost + "' " +
    "AND RecordType='Http Session' " +
    "ORDER BY StartTime";

  ResultSet rs2 = stmt2.executeQuery(sqlStatement2);
  crntTime = 0;
  flag = false;
  while (rs2.next()) {
    flag = true;
    prvTime = crntTime;
    crntTime = (rs2.getTimestamp(1)).getTime();
    if (prvTime == 0) {
      startTime = rs2.getTimestamp(1);
      endTime = rs2.getTimestamp(2);
      targetHost = rs2.getString(6);
      if (rs2.getString(7).equals("Http Log")) {
        descr = rs2.getString(3);
      }
      else descr = "";
      packets = rs2.getInt(4);
      volume = rs2.getInt(5);
      min = Integer.MAX_VALUE;
      max = -1;
      avgCounter = 0;
      stdDevCounter = 0;
      numUsages = 1;
    }
    else if ((crntGap = (crntTime - prvTime)) <= activityGap) {
      endTime = rs2.getTimestamp(2);
      if (targetHost.indexOf(rs2.getString(6)) == -1) {
        targetHost = targetHost + " || " + rs2.getString(6);
                  }
      if ((rs2.getString(7).equals("Http Log")) &&
        (descr.indexOf(rs2.getString(3)) == -1)) {
```

```
      descr = descr + " || " + rs2.getString(3);
    }
    packets = packets + rs2.getInt(4);
    volume = volume + rs2.getInt(5);
    min = (int)Math.min(min, crntGap);
    max = (int)Math.max(max, crntGap);
    avgCounter = avgCounter + crntGap;
    stdDevCounter = stdDevCounter + (crntGap * crntGap);
    numUsages++;

  }
  else if (volume > 0) {
    // close and store current activity
    pstmt.clearParameters();
    pstmt.setString(1, "Web Surfing");
    pstmt.setTimestamp(2, startTime);
    pstmt.setTimestamp(3, endTime);
    pstmt.setString(4, "");
    pstmt.setString(5, crntHost);
    pstmt.setString(6, targetHost.substring(0,Math.min(511,targetHost.length())));
    pstmt.setString(7, descr.substring(0,Math.min(511,descr.length())));
    pstmt.setInt(8, numUsages);
    pstmt.setInt(9, packets);
    pstmt.setInt(10, volume);
    pstmt.setInt(11, -1);
    if (min == Integer.MAX_VALUE) {
      min = -1;
    }
    pstmt.setInt(12, min);
    pstmt.setInt(13, max);
    avg = avgCounter / Math.max(numUsages - 1, 1);
    pstmt.setDouble(14, avg);
    stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
    pstmt.setDouble(15, stdDev);
    pstmt.setString(16, recorder);
    pstmt.executeUpdate();

    // and start a new one
    startTime = rs2.getTimestamp(1);
    endTime = rs2.getTimestamp(2);
    targetHost = rs2.getString(6);
    if (rs2.getString(7).equals("Http Log")) {
      descr = rs2.getString(3);
    }
    else descr = "";
    packets = rs2.getInt(4);
    volume = rs2.getInt(5);
    min = Integer.MAX_VALUE;
    max = -1;
    avgCounter = 0;
    stdDevCounter = 0;
    numUsages = 1;
  }
  else {
    // just start a new one
    startTime = rs2.getTimestamp(1);
    endTime = rs2.getTimestamp(2);
    targetHost = rs2.getString(6);
    if (rs2.getString(7).equals("Http Log")) {
      descr = rs2.getString(3);
    }
    else descr = "";
    packets = rs2.getInt(4);
    volume = rs2.getInt(5);
```

```
        min = Integer.MAX_VALUE;
        max = -1;
        avgCounter = 0;
        stdDevCounter = 0;
        numUsages = 1;
      }

    }

    if (flag && (volume > 0)) {
      // close and store current activity
      pstmt.clearParameters();
      pstmt.setString(1, "Web Surfing");
      pstmt.setTimestamp(2, startTime);
      pstmt.setTimestamp(3, endTime);
      pstmt.setString(4, "");
      pstmt.setString(5, crntHost);
      pstmt.setString(6, targetHost.substring(0,Math.min(511,targetHost.length())));
      pstmt.setString(7, descr.substring(0,Math.min(511,descr.length())));
      pstmt.setInt(8, numUsages);
      pstmt.setInt(9, packets);
      pstmt.setInt(10, volume);
      pstmt.setInt(11, -1);
      if (min == Integer.MAX_VALUE) {
        min = -1;
      }
      pstmt.setInt(12, min);
      pstmt.setInt(13, max);
      avg = avgCounter / Math.max(numUsages - 1, 1);
      pstmt.setDouble(14, avg);
      stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
      pstmt.setDouble(15, stdDev);
      pstmt.setString(16, recorder);
      pstmt.executeUpdate();

    }
  }
}

public static void BuildRemoteWebActivities() throws java.sql.SQLException {

  int         activityGap = 1800000; // 1 hour
  String      sqlStatement1 = null, sqlStatement2 = null;
  String      sqlStatement0 = null;
  String      crntHost = null, crntServer = null;
  long        crntTime = 0,prvTime = 0,crntGap = 0;
  Timestamp   startTime = null, endTime = null;
  int         min = Integer.MAX_VALUE,max = -1,numUsages = 0;
  int         data1 = 0, data2 = 0;
  long        avgCounter = 0,stdDevCounter = 0;
  double      avg = 0, stdDev = 0;
  boolean     flag = false;

  System.out.println("******* RemoteWebActivities **********");

  sqlStatement0 = "INSERT INTO " + schema + ".activities " +
    "(RecordType,StartTime,EndTime," +
    " InitiatingUser,InitiatingHost,Target," +
    " Description,NumUsages,Data1,Data2,Data3," +
    " Min,Max,Average,StdDev,Recorder)" +
    " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
  PreparedStatement pstmt = dbCon.prepareStatement(sqlStatement0);

  Statement stmt1 = dbCon.createStatement();
```

```
Statement stmt2 = dbCon.createStatement();

sqlStatement1 = "SELECT DISTINCT TargetHost " +
  "FROM " + schema + ".usage " +
  "WHERE Category='Net Log' " +
  "AND RecordType='Http Session' " +
  "AND (TargetHost LIKE '131.114.2.%' " +
  "OR TargetHost LIKE '131.114.4.%') " +
  "AND TargetHost <> '131.114.4.11' " +
  "AND description = 'Two' " +
  "AND DataVolume > 0";

ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
while (rs1 = rs1.getString(1); .next()) {
  crntHost
  System.out.println("LocalHost:" + crntHost );
  sqlStatement2 = "SELECT StartTime,EndTime,DataPackets,DataVolume " +
    "FROM " + schema + ".usage " +
    "WHERE Category='Net Log' " +
    "AND RecordType='Http Session' " +
    "AND TargetHost='" + crntHost + "' " +
    "AND Description = 'Two' " +
    "AND DataVolume > 3000 " +
    "ORDER BY StartTime";
  ResultSet rs2 = stmt2.executeQuery(sqlStatement2);
  crntTime = 0;
  flag = false;
  while (rs2.next()) {
    flag = true;
    prvTime = crntTime;
    crntTime = (rs2.getTimestamp(1)).getTime();
    if (prvTime == 0) {
      startTime = rs2.getTimestamp(1);
      endTime = rs2.getTimestamp(2);
      data1 = rs2.getInt(3);
      data2 = rs2.getInt(4);
      min = Integer.MAX_VALUE;
      max = -1;
      avgCounter = 0;
      stdDevCounter = 0;
      numUsages = 1;
    }
    else if ((crntGap = (crntTime - prvTime)) <= activityGap) {
      endTime = rs2.getTimestamp(2);
      data1 = data1 + rs2.getInt(3);
      data2 = data2 + rs2.getInt(4);
      min = (int)Math.min(min, crntGap);
      max = (int)Math.max(max, crntGap);
      avgCounter = avgCounter + crntGap;
      stdDevCounter = stdDevCounter + (crntGap * crntGap);
      numUsages++;

    }
    else {
      // close and store current activity
      pstmt.clearParameters();
      pstmt.setString(1, "Remote Web Surfing");
      pstmt.setTimestamp(2, startTime);
      pstmt.setTimestamp(3, endTime);
      pstmt.setString(4, "");
      pstmt.setString(5, crntHost);
      pstmt.setString(6, crntServer);
      pstmt.setString(7, "");
      pstmt.setInt(8, numUsages);
```

```
        pstmt.setInt(9, data1);
        pstmt.setInt(10, data2);
        pstmt.setInt(11, -1);
        if (min == Integer.MAX_VALUE) {
          min = -1;
        }
        pstmt.setInt(12, min);
        pstmt.setInt(13, max);
        avg = avgCounter / Math.max(numUsages - 1, 1);
        pstmt.setDouble(14, avg);
        stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
        pstmt.setDouble(15, stdDev);
        pstmt.setString(16, recorder);
        pstmt.executeUpdate();

        // and start a new one
        startTime = rs2.getTimestamp(1);
        endTime = rs2.getTimestamp(2);
        data1 = rs2.getInt(3);
        data2 = rs2.getInt(4);
        min = Integer.MAX_VALUE;
        max = -1;
        avgCounter = 0;
        stdDevCounter = 0;
        numUsages = 1;
      }
    }

    if (flag) {
      // close and store current activity
      pstmt.clearParameters();
      pstmt.setString(1, "Remote Web Surfing");
      pstmt.setTimestamp(2, startTime);
      pstmt.setTimestamp(3, endTime);
      pstmt.setString(4, "");
      pstmt.setString(5, crntHost);
      pstmt.setString(6, crntServer);
      pstmt.setString(7, "");
      pstmt.setInt(8, numUsages);
      pstmt.setInt(9, data1);
      pstmt.setInt(10, data2);
      pstmt.setInt(11, -1);
      if (min == Integer.MAX_VALUE) {
        min = -1;
      }
      pstmt.setInt(12, min);
      pstmt.setInt(13, max);
      avg = avgCounter / Math.max(numUsages - 1, 1);
      pstmt.setDouble(14, avg);
      stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
      pstmt.setDouble(15, stdDev);
      pstmt.setString(16, recorder);
      pstmt.executeUpdate();

    }
  }
}

public static void BuildLocalSmtpActivities() throws java.sql.SQLException {

  String      sqlStatement1 = null, sqlStatement2 = null;
  String      senderStatement, recipientStatement;
  String      crntRef = null, crntUser = null;
  String      recList = "";
```

```
Timestamp   lastSending = null;

System.out.println("******* LocalSmtpActivities **********");

senderStatement = "INSERT INTO " + schema + ".activities " +
  "(RecordType,StartTime,EndTime," +
  " InitiatingUser,InitiatingHost,Target," +
  " Description,NumUsages,Data1,Data2,Data3," +
  " Min,Max,Average,StdDev,Recorder)" +
  " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
PreparedStatement pstmtSender = dbCon.prepareStatement(senderStatement);

recipientStatement = "INSERT INTO " + schema + ".recipients " +
  "(MsgId,StartTime,EndTime,Recipient,HostIp) " +
  "VALUES (?,?,?,?,?)";
PreparedStatement pstmtRecipient = dbCon.prepareStatement(recipientStatement);

Statement stmt1 = dbCon.createStatement();
Statement stmt2 = dbCon.createStatement();

sqlStatement1 = "SELECT DISTINCT LocalRef " +
  "FROM " + schema + ".usage " +
  "WHERE Category='Smtp Log'" +
  "AND RecordType='Message Sending'";

ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
while (rs1.next()) {
  crntRef = rs1.getString(1);
  System.out.println("CrntRef:" + crntRef);
  sqlStatement2 = "SELECT * " +
    "FROM " + schema + ".usage " +
    "WHERE Category='Smtp Log' " +
    "AND LocalRef='" + crntRef + "' " +
    "ORDER BY Recorded";
  ResultSet rs2 = stmt2.executeQuery(sqlStatement2);
  recList = "";
  while (rs2.next()) {

    if (!((rs2.getString(8)).equals(""))) {
      pstmtSender.clearParameters();
      pstmtSender.setString(1, "Local Email Sending");
      pstmtSender.setTimestamp(2, rs2.getTimestamp(6));
      lastSending = rs2.getTimestamp(7);
      pstmtSender.setString(4, rs2.getString(8));
      pstmtSender.setString(5, rs2.getString(9));
      pstmtSender.setString(7, rs2.getString(12));
      pstmtSender.setInt(8, rs2.getInt(15));
      pstmtSender.setInt(9, rs2.getInt(15));
      pstmtSender.setInt(10, rs2.getInt(16));
      pstmtSender.setInt(11, -1);
      pstmtSender.setInt(12, -1);
      pstmtSender.setInt(13, -1);
      pstmtSender.setInt(14, -1);
      pstmtSender.setInt(15, -1);
      pstmtSender.setString(16,recorder);

    }
    else {
      pstmtRecipient.clearParameters();
      pstmtRecipient.setInt(1, -1);
      pstmtRecipient.setTimestamp(2, rs2.getTimestamp(6));
      lastSending = rs2.getTimestamp(7);
      recList = recList + " || " + rs2.getString(10);
      pstmtRecipient.setTimestamp(3, lastSending);
```

```
        pstmtRecipient.setString(4, rs2.getString(10));
        pstmtRecipient.setString(5, rs2.getString(11));
        pstmtRecipient.executeUpdate();
      }

    }
    pstmtSender.setTimestamp(3, lastSending);
    recList = recList.substring(0, Math.min(511, recList.length()));
    pstmtSender.setString(6, recList);
    pstmtSender.executeUpdate();
  }

}

public static void BuildRemoteSmtpActivities() throws java.sql.SQLException {

  String      sqlStatement1 = null;
  String      sqlStatement0 = null;

  System.out.println("******* RemoteSmtpActivities **********");

  sqlStatement0 = "INSERT INTO " + schema + ".activities " +
    "(RecordType,StartTime,EndTime," +
    " InitiatingUser,InitiatingHost,Target," +
    " Description,NumUsages,Data1,Data2,Data3," +
    " Min,Max,Average,StdDev,Recorder)" +
    " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
  PreparedStatement pstmt = dbCon.prepareStatement(sqlStatement0);

  Statement stmt1 = dbCon.createStatement();
  sqlStatement1 = "SELECT StartTime,EndTime,InitiatingHost,TargetHost, " +
                        "DataPackets,DataVolume,Min,Max,Average,StdDev " +
    "FROM " + schema + ".usage " +
    "WHERE Category='Net Log' " +
    "AND RecordType='Smtp Session' " +
    "AND InitiatingHost <> '131.114.4.6' " +
    "AND TargetHost <> '131.114.4.6' " +
    "AND DataVolume > 0";

  ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
  while (rs1.next()) {
    // store current activity
    pstmt.clearParameters();
    if ((rs1.getString(3).indexOf("131.114.2.") == -1) &&
      (rs1.getString(3).indexOf("131.114.4.") == -1)) {
      pstmt.setString(1, "Remote Email Receiving");
    }
    else {
      pstmt.setString(1, "Remote Email Sending");
    }
    pstmt.setTimestamp(2, rs1.getTimestamp(1));
    pstmt.setTimestamp(3, rs1.getTimestamp(2));
    pstmt.setString(4, "");
    pstmt.setString(5, rs1.getString(3));
    pstmt.setString(6, rs1.getString(4));
    pstmt.setString(7, "");
    pstmt.setInt(8, 1);
    pstmt.setInt(9, rs1.getInt(5));
    pstmt.setInt(10, rs1.getInt(6));
    pstmt.setInt(11, -1);
    pstmt.setInt(12, rs1.getInt(7));
    pstmt.setInt(13, rs1.getInt(8));
    pstmt.setDouble(14, rs1.getInt(9));
    pstmt.setDouble(15, rs1.getInt(10));
```

```
    pstmt.setString(16, recorder);
    pstmt.executeUpdate();
  }
}


public static void BuildLocalPopActivities() throws java.sql.SQLException {

  int         activityGap = 3600000; // 1 hour
  String      sqlStatement1 = null, sqlStatement2 = null;
  String      sqlStatement0 = null;
  String      crntUser = null,crntHost = null;
  long        crntTime = 0,prvTime = 0,crntGap = 0;
  Timestamp   startTime = null, endTime = null;
  int         newMsgs = 0,delMsgs = 0,leftMsgs = 0,foundMsgs = 0;
  int         min = Integer.MAX_VALUE,max = -1,numUsages = 0;
  long        avgCounter = 0,stdDevCounter = 0;
  double      avg = 0, stdDev = 0;
  boolean     flag = false;

  System.out.println("******* LocalPopActivities **********");

  sqlStatement0 = "INSERT INTO " + schema + ".activities " +
    "(RecordType,StartTime,EndTime," +
    " InitiatingUser,InitiatingHost,Target," +
    " Description,NumUsages,Data1,Data2,Data3," +
    " Min,Max,Average,StdDev,Recorder)" +
    " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
  PreparedStatement pstmt = dbCon.prepareStatement(sqlStatement0);

  Statement stmt1 = dbCon.createStatement();
  Statement stmt2 = dbCon.createStatement();

  sqlStatement1 = "SELECT DISTINCT InitiatingUser,InitiatingHost " +
    "FROM " + schema + ".usage " +
    "WHERE Category='Pop Log' ";

  ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
  while (rs1.next()) {
    crntUser = rs1.getString(1);
    crntHost = rs1.getString(2);
    System.out.println("CrntUser:" + crntUser + "    " + "CrntHost:" + crntHost);
    sqlStatement2 = "SELECT StartTime,EndTime,DataPackets,DataVolume " +
      "FROM " + schema + ".usage " +
      "WHERE Category='Pop Log' " +
      "AND InitiatingUser='" + crntUser + "' " +
      "AND InitiatingHost='" + crntHost + "' " +
      "ORDER BY StartTime";
    ResultSet rs2 = stmt2.executeQuery(sqlStatement2);
    crntTime = 0;
    flag = false;
    while (rs2.next()) {
      flag = true;
      prvTime = crntTime;
      crntTime = (rs2.getTimestamp(1)).getTime();
      if (prvTime == 0) {

        startTime = rs2.getTimestamp(1);
        endTime = rs2.getTimestamp(2);
        foundMsgs = rs2.getInt(3) + rs2.getInt(4);
        leftMsgs = rs2.getInt(3);
        delMsgs = rs2.getInt(4);
        min = Integer.MAX_VALUE;
        max = -1;
```

```
      avgCounter = 0;
      stdDevCounter = 0;
      numUsages = 1;

    }
    else if ((crntGap = (crntTime - prvTime)) <= activityGap) {
      endTime = rs2.getTimestamp(2);
      delMsgs = delMsgs + rs2.getInt(4);
      leftMsgs = rs2.getInt(4);
      min = (int)Math.min(min, crntGap);
      max = (int)Math.max(max, crntGap);
      avgCounter = avgCounter + crntGap;
      stdDevCounter = stdDevCounter + (crntGap * crntGap);
      numUsages++;

    }
    else {
      // close and store current activity
      pstmt.clearParameters();
      pstmt.setString(1, "Local Email Downloading");
      pstmt.setTimestamp(2, startTime);
      pstmt.setTimestamp(3, endTime);
      pstmt.setString(4, crntUser);
      pstmt.setString(5, crntHost);
      pstmt.setString(6, "");
      pstmt.setString(7, "");
      pstmt.setInt(8, numUsages);
      pstmt.setInt(9, foundMsgs);
      pstmt.setInt(10, delMsgs);
      pstmt.setInt(11, leftMsgs);
      if (min == Integer.MAX_VALUE) {
        min = -1;
      }
      pstmt.setInt(12, min);
      pstmt.setInt(13, max);
      avg = avgCounter / Math.max(numUsages - 1, 1);
      pstmt.setDouble(14, avg);
      stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
      pstmt.setDouble(15, stdDev);
      pstmt.setString(16, recorder);
      pstmt.executeUpdate();

      // and start a new one
      startTime = rs2.getTimestamp(1);
      endTime = rs2.getTimestamp(2);
      foundMsgs = rs2.getInt(3) + rs2.getInt(4);
      leftMsgs = rs2.getInt(3);
      delMsgs = rs2.getInt(4);
      min = Integer.MAX_VALUE;
      max = -1;
      avgCounter = 0;
      stdDevCounter = 0;
      numUsages = 1;
    }
  }

  if (flag) {
    // close and store current activity
    pstmt.clearParameters();
    pstmt.setString(1, "Local Email Downloading");
    pstmt.setTimestamp(2, startTime);
    pstmt.setTimestamp(3, endTime);
    pstmt.setString(4, crntUser);
    pstmt.setString(5, crntHost);
```

```
      pstmt.setString(6, "");
      pstmt.setString(7, "");
      pstmt.setInt(8, numUsages);
      pstmt.setInt(9, foundMsgs);
      pstmt.setInt(10, delMsgs);
      pstmt.setInt(11, leftMsgs);
      if (min == Integer.MAX_VALUE) {
        min = -1;
      }
      pstmt.setInt(12, min);
      pstmt.setInt(13, max);
      avg = avgCounter / Math.max(numUsages - 1, 1);
      pstmt.setDouble(14, avg);
      stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
      pstmt.setDouble(15, stdDev);
      pstmt.setString(16, recorder);
      pstmt.executeUpdate();

    }
  }
}

public static void BuildRemoteOutPopActivities() throws java.sql.SQLException {

  int         activityGap = 3600000; // 1 hour
  String      sqlStatement1 = null, sqlStatement2 = null;
  String      sqlStatement0 = null;
  String      crntHost = null, crntServer = null;
  long        crntTime = 0,prvTime = 0,crntGap = 0;
  Timestamp   startTime = null, endTime = null;
  int         min = Integer.MAX_VALUE,max = -1,numUsages = 0;
  int         data1 = 0, data2 = 0;
  long        avgCounter = 0,stdDevCounter = 0;
  double      avg = 0, stdDev = 0;
  boolean     flag = false;

  System.out.println("******* RemoteOutPopActivities **********");

  sqlStatement0 = "INSERT INTO " + schema + ".activities " +
    "(RecordType,StartTime,EndTime," +
    " InitiatingUser,InitiatingHost,Target," +
    " Description,NumUsages,Data1,Data2,Data3," +
    " Min,Max,Average,StdDev,Recorder)" +
    " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
  PreparedStatement pstmt = dbCon.prepareStatement(sqlStatement0);

  Statement stmt1 = dbCon.createStatement();
  Statement stmt2 = dbCon.createStatement();

  sqlStatement1 = "SELECT DISTINCT InitiatingHost,TargetHost " +
    "FROM " + schema + ".usage " +
    "WHERE Category='Net Log' " +
    "AND RecordType='Pop Session' " +
    "AND (initiatinghost LIKE '131.114.2.%' " +
    "OR initiatinghost LIKE '131.114.4.%')";

  ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
  while (rs1.next()) {
    crntHost = rs1.getString(1);
    crntServer = rs1.getString(2);
    System.out.println("LocalHost:" + crntHost + "    " + "RemoteHost:" + crntServer);

    sqlStatement2 = "SELECT StartTime,EndTime,DataPackets,DataVolume " +
      "FROM " + schema + ".usage " +
```

```
  "WHERE Category='Net Log' " +
  "AND RecordType='Pop Session' " +
  "AND InitiatingHost='" + crntHost + "' " +
  "AND TargetHost='" + crntServer + "' " +
  "AND DataVolume <> 0 " +
  "ORDER BY StartTime";
ResultSet rs2 = stmt2.executeQuery(sqlStatement2);

crntTime = 0;
flag = false;
while (rs2.next()) {

  flag = true;
  prvTime = crntTime;
  crntTime = (rs2.getTimestamp(1)).getTime();
  if (prvTime == 0) {
    startTime = rs2.getTimestamp(1);
    endTime = rs2.getTimestamp(2);
    data1 = rs2.getInt(3);
    data2 = rs2.getInt(4);
    min = Integer.MAX_VALUE;
    max = -1;
    avgCounter = 0;
    stdDevCounter = 0;
    numUsages = 1;
  }
  else if ((crntGap = (crntTime - prvTime)) <= activityGap) {
    endTime = rs2.getTimestamp(2);
    data1 = data1 + rs2.getInt(3);
    data2 = data2 + rs2.getInt(4);
    min = (int)Math.min(min, crntGap);
    max = (int)Math.max(max, crntGap);
    avgCounter = avgCounter + crntGap;
    stdDevCounter = stdDevCounter + (crntGap * crntGap);
    numUsages++;

  }
  else {
    // close and store current activity
    pstmt.clearParameters();
    pstmt.setString(1, "Remote Email Downloading");
    pstmt.setTimestamp(2, startTime);
    pstmt.setTimestamp(3, endTime);
    pstmt.setString(4, "");
    pstmt.setString(5, crntHost);
    pstmt.setString(6, crntServer);
    pstmt.setString(7, "");
    pstmt.setInt(8, numUsages);
    pstmt.setInt(9, data1);
    pstmt.setInt(10, data2);
    pstmt.setInt(11, -1);
    if (min == Integer.MAX_VALUE) {
      min = -1;
    }
    pstmt.setInt(12, min);
    pstmt.setInt(13, max);
    avg = avgCounter / Math.max(numUsages - 1, 1);
    pstmt.setDouble(14, avg);
    stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
    pstmt.setDouble(15, stdDev);
    pstmt.setString(16, recorder);
    pstmt.executeUpdate();

    // and start a new one
```

```
        startTime = rs2.getTimestamp(1);
        endTime = rs2.getTimestamp(2);
        data1 = rs2.getInt(3);
        data2 = rs2.getInt(4);
        min = Integer.MAX_VALUE;
        max = -1;
        avgCounter = 0;
        stdDevCounter = 0;
        numUsages = 1;
      }
    }

    if (flag) {
      // close and store current activity
      pstmt.clearParameters();
      pstmt.setString(1, "Remote Email Downloading");
      pstmt.setTimestamp(2, startTime);
      pstmt.setTimestamp(3, endTime);
      pstmt.setString(4, "");
      pstmt.setString(5, crntHost);
      pstmt.setString(6, crntServer);
      pstmt.setString(7, "");
      pstmt.setInt(8, numUsages);
      pstmt.setInt(9, data1);
      pstmt.setInt(10, data2);
      pstmt.setInt(11, -1);
      if (min == Integer.MAX_VALUE) {
        min = -1;
      }
      pstmt.setInt(12, min);
      pstmt.setInt(13, max);
      avg = avgCounter / Math.max(numUsages - 1, 1);
      pstmt.setDouble(14, avg);
      stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
      pstmt.setDouble(15, stdDev);
      pstmt.setString(16, recorder);
      pstmt.executeUpdate();

    }
  }
}

public static void BuildRemoteInPopActivities() throws java.sql.SQLException {

  int         activityGap = 3600000; // 1 hour
  String      sqlStatement1 = null, sqlStatement2 = null;
  String      sqlStatement0 = null;
  String      crntHost = null, crntServer = null;
  long        crntTime = 0,prvTime = 0,crntGap = 0;
  Timestamp   startTime = null, endTime = null;
  int         min = Integer.MAX_VALUE,max = -1,numUsages = 0;
  int         data1 = 0, data2 = 0;
  long        avgCounter = 0,stdDevCounter = 0;
  double      avg = 0, stdDev = 0;
  boolean     flag = false;

  System.out.println("******* RemoteInPopActivities **********");

  sqlStatement0 = "INSERT INTO " + schema + ".activities " +
    "(RecordType,StartTime,EndTime," +
    " InitiatingUser,InitiatingHost,Target," +
    " Description,NumUsages,Data1,Data2,Data3," +
    " Min,Max,Average,StdDev,Recorder)" +
    " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
```

```
PreparedStatement pstmt = dbCon.prepareStatement(sqlStatement0);

Statement stmt1 = dbCon.createStatement();
Statement stmt2 = dbCon.createStatement();

sqlStatement1 = "SELECT DISTINCT InitiatingHost,TargetHost " +
  "FROM " + schema + ".usage " +
  "WHERE Category='Net Log' " +
  "AND RecordType='Pop Session' " +
  "AND (TargetHost LIKE '131.114.2.%' " +
  "OR TargetHost LIKE '131.114.4.%') " +
  "AND TargetHost <> '131.114.4.6'";

ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
while (rs1.next()) {
  crntHost = rs1.getString(1);
  crntServer = rs1.getString(2);
  System.out.println("LocalHost:" + crntHost + "    " + "RemoteHost:" + crntServer);
  sqlStatement2 = "SELECT StartTime,EndTime,DataPackets,DataVolume " +
    "FROM " + schema + ".usage " +
    "WHERE Category='Net Log' " +
    "AND RecordType='Pop Session' " +
    "AND InitiatingHost='" + crntHost + "' " +
    "AND TargetHost='" + crntServer + "' " +
    "AND DataVolume > 0 " +
    "ORDER BY StartTime";
  ResultSet rs2 = stmt2.executeQuery(sqlStatement2);

  crntTime = 0;
  flag = false;
  while (rs2.next()) {

    flag = true;
    prvTime = crntTime;
    crntTime = (rs2.getTimestamp(1)).getTime();
    if (prvTime == 0) {
      startTime = rs2.getTimestamp(1);
      endTime = rs2.getTimestamp(2);
      data1 = rs2.getInt(3);
      data2 = rs2.getInt(4);
      min = Integer.MAX_VALUE;
      max = -1;
      avgCounter = 0;
      stdDevCounter = 0;
      numUsages = 1;
    }
    else if ((crntGap = (crntTime - prvTime)) <= activityGap) {
      endTime = rs2.getTimestamp(2);
      data1 = data1 + rs2.getInt(3);
      data2 = data2 + rs2.getInt(4);
      min = (int)Math.min(min, crntGap);
      max = (int)Math.max(max, crntGap);
      avgCounter = avgCounter + crntGap;
      stdDevCounter = stdDevCounter + (crntGap * crntGap);
      numUsages++;

    }
    else {
      // close and store current activity
      pstmt.clearParameters();
      pstmt.setString(1, "Remote Email Downloading");
      pstmt.setTimestamp(2, startTime);
      pstmt.setTimestamp(3, endTime);
      pstmt.setString(4, "");
```

```
        pstmt.setString(5, crntHost);
        pstmt.setString(6, crntServer);
        pstmt.setString(7, "");
        pstmt.setInt(8, numUsages);
        pstmt.setInt(9, data1);
        pstmt.setInt(10, data2);
        pstmt.setInt(11, -1);
        if (min == Integer.MAX_VALUE) {
          min = -1;
        }
        pstmt.setInt(12, min);
        pstmt.setInt(13, max);
        avg = avgCounter / Math.max(numUsages - 1, 1);
        pstmt.setDouble(14, avg);
        stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
        pstmt.setDouble(15, stdDev);
        pstmt.setString(16, recorder);
        pstmt.executeUpdate();

        // and start a new one
        startTime = rs2.getTimestamp(1);
        endTime = rs2.getTimestamp(2);
        data1 = rs2.getInt(3);
        data2 = rs2.getInt(4);
        min = Integer.MAX_VALUE;
        max = -1;
        avgCounter = 0;
        stdDevCounter = 0;
        numUsages = 1;
      }
    }

    if (flag) {
      // close and store current activity
      pstmt.clearParameters();
      pstmt.setString(1, "Remote Email Downloading");
      pstmt.setTimestamp(2, startTime);
      pstmt.setTimestamp(3, endTime);
      pstmt.setString(4, "");
      pstmt.setString(5, crntHost);
      pstmt.setString(6, crntServer);
      pstmt.setString(7, "");
      pstmt.setInt(8, numUsages);
      pstmt.setInt(9, data1);
      pstmt.setInt(10, data2);
      pstmt.setInt(11, -1);
      if (min == Integer.MAX_VALUE) {
        min = -1;
      }
      pstmt.setInt(12, min);
      pstmt.setInt(13, max);
      avg = avgCounter / Math.max(numUsages - 1, 1);
      pstmt.setDouble(14, avg);
      stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
      pstmt.setDouble(15, stdDev);
      pstmt.setString(16, recorder);
      pstmt.executeUpdate();

    }
  }
}

public static void BuildLocalImapActivities() throws java.sql.SQLException {
```

```
int         activityGap = 3600000; // 1 hour
String      sqlStatement1 = null, sqlStatement2 = null;
String      sqlStatement0 = null;
String      crntUser = null,crntHost = null;
long        crntTime = 0,prvTime = 0,crntGap = 0;
Timestamp   startTime = null, endTime = null;
int         min = Integer.MAX_VALUE,max = -1,numUsages = 0;
long        avgCounter = 0,stdDevCounter = 0;
double      avg = 0, stdDev = 0;
boolean     flag = false;

System.out.println("******* LocalImapActivities **********");

sqlStatement0 = "INSERT INTO " + schema + ".activities " +
  "(RecordType,StartTime,EndTime," +
  " InitiatingUser,InitiatingHost,Target," +
  " Description,NumUsages,Data1,Data2,Data3," +
  " Min,Max,Average,StdDev,Recorder)" +
  " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
PreparedStatement pstmt = dbCon.prepareStatement(sqlStatement0);

Statement stmt1 = dbCon.createStatement();
Statement stmt2 = dbCon.createStatement();

sqlStatement1 = "SELECT DISTINCT InitiatingUser,InitiatingHost " +
  "FROM " + schema + ".usage " +
  "WHERE Category='Imap Log' ";

ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
while (rs1.next()) {
  crntUser = rs1.getString(1);
  crntHost = rs1.getString(2);
  System.out.println("CrntUser:" + crntUser + "    " + "CrntHost:" + crntHost);
  sqlStatement2 = "SELECT StartTime,EndTime,InitiatingHost,DataPackets,DataVolume " +
    "FROM " + schema + ".usage " +
    "WHERE Category='Imap Log' " +
    "AND InitiatingUser='" + crntUser + "' " +
    "AND InitiatingHost='" + crntHost + "' " +
    "ORDER BY StartTime";
  ResultSet rs2 = stmt2.executeQuery(sqlStatement2);
  crntTime = 0;
  flag = false;
  while (rs2.next()) {
    flag = true;
    prvTime = crntTime;
    crntTime = (rs2.getTimestamp(1)).getTime();
    if (prvTime == 0) {
      startTime = rs2.getTimestamp(1);
      endTime = rs2.getTimestamp(2);
      min = Integer.MAX_VALUE;
      max = -1;
      avgCounter = 0;
      stdDevCounter = 0;
      numUsages = 1;
    }
    else if ((crntGap = (crntTime - prvTime)) <= activityGap) {
      endTime = rs2.getTimestamp(2);
      min = (int)Math.min(min, crntGap);
      max = (int)Math.max(max, crntGap);
      avgCounter = avgCounter + crntGap;
      stdDevCounter = stdDevCounter + (crntGap * crntGap);
      numUsages++;

    }
```

```
  else {
    // close and store current activity
    pstmt.clearParameters();
    pstmt.setString(1, "Local Email Downloading");
    pstmt.setTimestamp(2, startTime);
    pstmt.setTimestamp(3, endTime);
    pstmt.setString(4, crntUser);
    pstmt.setString(5, crntHost);
    pstmt.setString(6, "");
    pstmt.setString(7, "");
    pstmt.setInt(8, numUsages);
    pstmt.setInt(9, -1);
    pstmt.setInt(10, -1);
    pstmt.setInt(11, -1);
    if (min == Integer.MAX_VALUE) {
      min = -1;
    }
    pstmt.setInt(12, min);
    pstmt.setInt(13, max);
    avg = avgCounter / Math.max(numUsages - 1, 1);
    pstmt.setDouble(14, avg);
    stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
    pstmt.setDouble(15, stdDev);
    pstmt.setString(16, recorder);
    pstmt.executeUpdate();

    // and start a new one
    startTime = rs2.getTimestamp(1);
    endTime = rs2.getTimestamp(2);
    min = Integer.MAX_VALUE;
    max = -1;
    avgCounter = 0;
    stdDevCounter = 0;
    numUsages = 1;
  }
}

if (flag) {
  // close and store current activity
  pstmt.clearParameters();
  pstmt.setString(1, "Local Email Downloading");
  pstmt.setTimestamp(2, startTime);
  pstmt.setTimestamp(3, endTime);
  pstmt.setString(4, crntUser);
  pstmt.setString(5, crntHost);
  pstmt.setString(6, "");
  pstmt.setString(7, "");
  pstmt.setInt(8, numUsages);
  pstmt.setInt(9, -1);
  pstmt.setInt(10, -1);
  pstmt.setInt(11, -1);
  if (min == Integer.MAX_VALUE) {
    min = -1;
  }
  pstmt.setInt(12, min);
  pstmt.setInt(13, max);
  avg = avgCounter / Math.max(numUsages - 1, 1);
  pstmt.setDouble(14, avg);
  stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
  pstmt.setDouble(15, stdDev);
  pstmt.setString(16, recorder);
  pstmt.executeUpdate();

}
```

```
    }
  }

  public static void BuildRemoteOutImapActivities() throws java.sql.SQLException {

    int         activityGap = 3600000; // 1 hour
    String      sqlStatement1 = null, sqlStatement2 = null;
    String      sqlStatement0 = null;
    String      crntHost = null, crntServer = null;
    long        crntTime = 0,prvTime = 0,crntGap = 0;
    Timestamp   startTime = null, endTime = null;
    int         min = Integer.MAX_VALUE,max = -1,numUsages = 0;
    int         data1 = 0, data2 = 0;
    long        avgCounter = 0,stdDevCounter = 0;
    double      avg = 0, stdDev = 0;
    boolean     flag = false;

    System.out.println("******* RemoteOutImapActivities **********");


sqlStatement0 = "INSERT INTO " + schema + ".activities " +
      "(RecordType,StartTime,EndTime," +
      " InitiatingUser,InitiatingHost,Target," +
      " Description,NumUsages,Data1,Data2,Data3," +
      " Min,Max,Average,StdDev,Recorder)" +
      " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
    PreparedStatement pstmt = dbCon.prepareStatement(sqlStatement0);

    Statement stmt1 = dbCon.createStatement();
    Statement stmt2 = dbCon.createStatement();

    sqlStatement1 = "SELECT DISTINCT InitiatingHost,TargetHost " +
      "FROM " + schema + ".usage " +
      "WHERE Category='Net Log' " +
      "AND RecordType='Imap Session' " +
      "AND (initiatinghost LIKE '131.114.2.%' " +
      "OR initiatinghost LIKE '131.114.4.%')";

    ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
    while (rs1.next()) {
      crntHost = rs1.getString(1);
      crntServer = rs1.getString(2);
      System.out.println("LocalHost:" + crntHost + "    " + "RemoteHost:" + crntServer);
      sqlStatement2 = "SELECT StartTime,EndTime,DataPackets,DataVolume " +
        "FROM " + schema + ".usage " +
        "WHERE Category='Net Log' " +
        "AND RecordType='Imap Session' " +
        "AND InitiatingHost='" + crntHost + "' " +
        "AND TargetHost='" + crntServer + "' " +
        "AND DataVolume > 0 " +
        "ORDER BY StartTime";
      ResultSet rs2 = stmt2.executeQuery(sqlStatement2);
      crntTime = 0;
      flag = false;
      while (rs2.next()) {
        flag = true;
        prvTime = crntTime;
        crntTime = (rs2.getTimestamp(1)).getTime();
        if (prvTime == 0) {
          startTime = rs2.getTimestamp(1);
          endTime = rs2.getTimestamp(2);
          data1 = rs2.getInt(3);
          data2 = rs2.getInt(4);
          min = Integer.MAX_VALUE;
```

```
      max = -1;
      avgCounter = 0;
      stdDevCounter = 0;
      numUsages = 1;
    }
    else if ((crntGap = (crntTime - prvTime)) <= activityGap) {
      endTime = rs2.getTimestamp(2);
      data1 = data1 + rs2.getInt(3);
      data2 = data2 + rs2.getInt(4);
      min = (int)Math.min(min, crntGap);
      max = (int)Math.max(max, crntGap);
      avgCounter = avgCounter + crntGap;
      stdDevCounter = stdDevCounter + (crntGap * crntGap);
      numUsages++;

    }
    else {
      // close and store current activity
      pstmt.clearParameters();
      pstmt.setString(1, "Remote Email Downloading");
      pstmt.setTimestamp(2, startTime);
      pstmt.setTimestamp(3, endTime);
      pstmt.setString(4, "");
      pstmt.setString(5, crntHost);
      pstmt.setString(6, crntServer);
      pstmt.setString(7, "");
      pstmt.setInt(8, numUsages);
      pstmt.setInt(9, data1);
      pstmt.setInt(10, data2);
      pstmt.setInt(11, -1);
      if (min == Integer.MAX_VALUE) {
        min = -1;
      }
      pstmt.setInt(12, min);
      pstmt.setInt(13, max);
      avg = avgCounter / Math.max(numUsages - 1, 1);
      pstmt.setDouble(14, avg);
      stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
      pstmt.setDouble(15, stdDev);
      pstmt.setString(16, recorder);
      pstmt.executeUpdate();

      // and start a new one
      startTime = rs2.getTimestamp(1);
      endTime = rs2.getTimestamp(2);
      data1 = rs2.getInt(3);
      data2 = rs2.getInt(4);
      min = Integer.MAX_VALUE;
      max = -1;
      avgCounter = 0;
      stdDevCounter = 0;
      numUsages = 1;
    }
  }

  if (flag) {
    // close and store current activity
    pstmt.clearParameters();
    pstmt.setString(1, "Remote Email Downloading");
    pstmt.setTimestamp(2, startTime);
    pstmt.setTimestamp(3, endTime);
    pstmt.setString(4, "");
    pstmt.setString(5, crntHost);
    pstmt.setString(6, crntServer);
```

```
      pstmt.setString(7, "");
      pstmt.setInt(8, numUsages);
      pstmt.setInt(9, data1);
      pstmt.setInt(10, data2);
      pstmt.setInt(11, -1);
      if (min == Integer.MAX_VALUE) {
        min = -1;
      }
      pstmt.setInt(12, min);
      pstmt.setInt(13, max);
      avg = avgCounter / Math.max(numUsages - 1, 1);
      pstmt.setDouble(14, avg);
      stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
      pstmt.setDouble(15, stdDev);
      pstmt.setString(16, recorder);
      pstmt.executeUpdate();

    }
  }
}

public static void BuildRemoteInImapActivities() throws java.sql.SQLException {

  int         activityGap = 3600000; // 1 hour
  String      sqlStatement1 = null, sqlStatement2 = null;
  String      sqlStatement0 = null;
  String      crntHost = null, crntServer = null;
  long        crntTime = 0,prvTime = 0,crntGap = 0;
  Timestamp   startTime = null, endTime = null;
  int         min = Integer.MAX_VALUE,max = -1,numUsages = 0;
  int         data1 = 0, data2 = 0;
  long        avgCounter = 0,stdDevCounter = 0;
  double      avg = 0, stdDev = 0;
  boolean     flag = false;

  System.out.println("******* RemoteInImapActivities **********");

  sqlStatement0 = "INSERT INTO " + schema + ".activities " +
    "(RecordType,StartTime,EndTime," +
    " InitiatingUser,InitiatingHost,Target," +
    " Description,NumUsages,Data1,Data2,Data3," +
    " Min,Max,Average,StdDev,Recorder)" +
    " VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
  PreparedStatement pstmt = dbCon.prepareStatement(sqlStatement0);

  Statement stmt1 = dbCon.createStatement();
  Statement stmt2 = dbCon.createStatement();

  sqlStatement1 = "SELECT DISTINCT InitiatingHost,TargetHost " +
    "FROM " + schema + ".usage " +
    "WHERE Category='Net Log' " +
    "AND RecordType='Imap Session' " +
    "AND (TargetHost LIKE '131.114.2.%' " +
    "OR TargetHost LIKE '131.114.4.%') " +
    "AND TargetHost <> '131.114.4.6'";

  ResultSet rs1 = stmt1.executeQuery(sqlStatement1);
  while (rs1.next()) {
    crntHost = rs1.getString(1);
    crntServer = rs1.getString(2);
    System.out.println("LocalHost:" + crntHost + "    " + "RemoteHost:" + crntServer);
    sqlStatement2 = "SELECT StartTime,EndTime,DataPackets,DataVolume " +
      "FROM " + schema + ".usage " +
      "WHERE Category='Net Log' " +
```

```
  "AND RecordType='Imap Session' " +
  "AND InitiatingHost='" + crntHost + "' " +
  "AND TargetHost='" + crntServer + "' " +
  "AND DataVolume > 0 " +
  "ORDER BY StartTime";
ResultSet rs2 = stmt2.executeQuery(sqlStatement2);
crntTime = 0;
flag = false;
while (rs2.next()) {
  flag = true;
  prvTime = crntTime;
  crntTime = (rs2.getTimestamp(1)).getTime();
  if (prvTime == 0) {
    startTime = rs2.getTimestamp(1);
    endTime = rs2.getTimestamp(2);
    data1 = rs2.getInt(3);
    data2 = rs2.getInt(4);
    min = Integer.MAX_VALUE;
    max = -1;
    avgCounter = 0;
    stdDevCounter = 0;
    numUsages = 1;
  }
  else if ((crntGap = (crntTime - prvTime)) <= activityGap) {
    endTime = rs2.getTimestamp(2);
    data1 = data1 + rs2.getInt(3);
    data2 = data2 + rs2.getInt(4);
    min = (int)Math.min(min, crntGap);
    max = (int)Math.max(max, crntGap);
    avgCounter = avgCounter + crntGap;
    stdDevCounter = stdDevCounter + (crntGap * crntGap);
    numUsages++;

  }
  else {
    // close and store current activity
    pstmt.clearParameters();
    pstmt.setString(1, "Remote Email Downloading");
    pstmt.setTimestamp(2, startTime);
    pstmt.setTimestamp(3, endTime);
    pstmt.setString(4, "");
    pstmt.setString(5, crntHost);
    pstmt.setString(6, crntServer);
    pstmt.setString(7, "");
    pstmt.setInt(8, numUsages);
    pstmt.setInt(9, data1);
    pstmt.setInt(10, data2);
    pstmt.setInt(11, -1);
    if (min == Integer.MAX_VALUE) {
      min = -1;
    }
    pstmt.setInt(12, min);
    pstmt.setInt(13, max);
    avg = avgCounter / Math.max(numUsages - 1, 1);
    pstmt.setDouble(14, avg);
    stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
    pstmt.setDouble(15, stdDev);
    pstmt.setString(16, recorder);
    pstmt.executeUpdate();

    // and start a new one
    startTime = rs2.getTimestamp(1);
    endTime = rs2.getTimestamp(2);
    data1 = rs2.getInt(3);
```

```
        data2 = rs2.getInt(4);
        min = Integer.MAX_VALUE;
        max = -1;
        avgCounter = 0;
        stdDevCounter = 0;
        numUsages = 1;
      }
    }

    if (flag) {
      // close and store current activity
      pstmt.clearParameters();
      pstmt.setString(1, "Remote Email Downloading");
      pstmt.setTimestamp(2, startTime);
      pstmt.setTimestamp(3, endTime);
      pstmt.setString(4, "");
      pstmt.setString(5, crntHost);
      pstmt.setString(6, crntServer);
      pstmt.setString(7, "");
      pstmt.setInt(8, numUsages);
      pstmt.setInt(9, data1);
      pstmt.setInt(10, data2);
      pstmt.setInt(11, -1);
      if (min == Integer.MAX_VALUE) {
        min = -1;
      }
      pstmt.setInt(12, min);
      pstmt.setInt(13, max);
      avg = avgCounter / Math.max(numUsages - 1, 1);
      pstmt.setDouble(14, avg);
      stdDev = Math.sqrt((stdDevCounter / Math.max(numUsages - 1, 1)) - (avg * avg));
      pstmt.setDouble(15, stdDev);
      pstmt.setString(16, recorder);
      pstmt.executeUpdate();

    }
  }
}

public static void main(String args[]) {

  String            user = "", host, application;
  String            server = "", source;
  long              time1 = 0;

  try {

    // have to do this to init log4j:
    AppFileProperties.loadProperties();

    // DEFAULTS
    host = java.net.InetAddress.getLocalHost().getHostName();
    application = "ActivitiesBuilder";

    // COMMAND LINE PARSING
    Getopt g = new Getopt("CommandLineParser", args, "u:h:a:s:l:");

    int c;

    while ((c = g.getopt()) != -1) {
      switch(c) {
      case 'u':
        user = g.getOptarg();
        break;
```

```
        case 'h':
          host = g.getOptarg();
          break;
        case 'a':
          application = g.getOptarg();
          break;
        case 's':
          schema = g.getOptarg();
          break;
        case 'l':
          server = g.getOptarg();
          break;
        case '?':
          break; // getopt() already printed an error
        default:
          System.out.println("getopt() returned " + c + "\n");
        }
      }


      recorder = user + " on " + host;

      // TABLES CREATION
      TableAccessProperty tap1 = new TableAccessProperty (DatabaseAccessProperty.getDefault(),
                                                          schema, "ACTIVITIES");
      SimpleTableLocalAccess.assertTableCreated(tap1);
      TableAccessProperty tap2 = new TableAccessProperty (DatabaseAccessProperty.getDefault(),
                                                          schema, "RECIPIENTS");
      SimpleTableLocalAccess.assertTableCreated(tap2);
      dbCon = DatabaseLocalAccess.getConnection();
      dbCon.setAutoCommit(true);

      BuildLocalWebActivities();
      BuildLocalPopActivities();
      BuildRemoteOutPopActivities();
      BuildRemoteInPopActivities();
      BuildLocalImapActivities();
      BuildRemoteOutImapActivities();
      BuildRemoteInImapActivities();
      BuildRemoteSmtpActivities();
      BuildRemoteWebActivities();
      BuildLocalSmtpActivities();


    } catch (Exception e) {
      System.out.println("Error: Exception occurred");
      e.printStackTrace();
    } finally {
      if (dbCon != null) {
        try {
          dbCon.close();
        } catch (Exception ex) {
          ex.printStackTrace();
        }
      }
    }
  }
}
```