Licensing Technology

Anurag Aggarwal
Department of Computer Science and Engineering,
Indian Institute of Technology,
Kanpur 208016
anuragag@cse.iitk.ac.in

Diwaker Gupta
Department of Computer Science and Engineering,
Indian Institute of Technology,
Kanpur 208016
gdiwaker@cse.iitk.ac.in

***Abstract*— Software piracy and licensing are becoming increasingly important in the Software industry. With the advent of computer networks and the explosion in the number of computer users, new concepts in licensing have come forth. In this report, we study the various licensing solutions possible in a networked environment under different requirements and constraints. We then take a look at some existing solutions. Finally, we present the design and implementation of two solutions – a peer-to-peer based mechanism , and a hybrid central server based mechanism.**

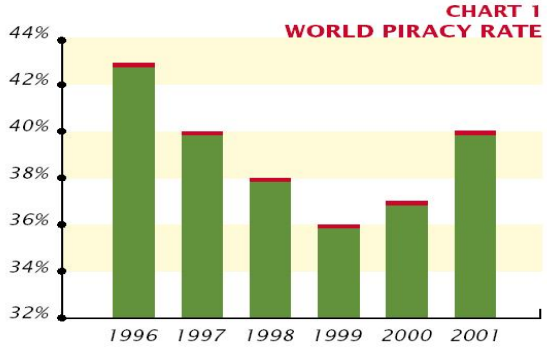
***Index Terms*— Software Piracy, Software Licensing, Copy Protection**

I. Introduction

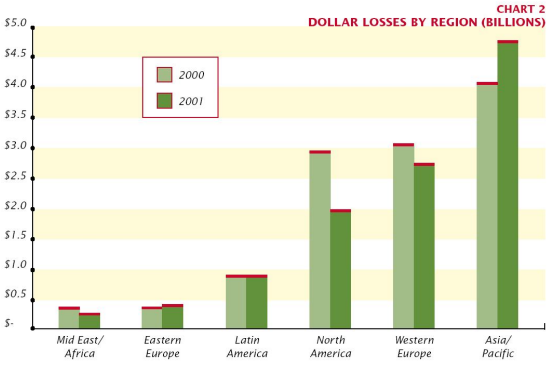
WORLDWIDE, the unauthorized copying, use and distribution of software is a continuing source of concern. Despite increased attention worldwide from government, industry and legal agencies alike, software piracy still has a severe impact on states and local communities.

Reports published by the BSA (Business Software Alliance) indicate that annual losses to the industry are $10.9 billion [2] (see Figure 1) . Another study conducted by International Planning and Research (IPR) and Microsoft [7] revealed the following in US alone:

- The increased software piracy rates experienced by 21 states caused a loss in excess of $2 million in wages and salaries, over 56,000 jobs and over $500,000 in tax revenue
- Total losses for the 10 most impacted states equaled approximately $3.5 billion, or 57 percent of the nation's losses
- In 1998, tax losses due to software piracy increased by approximately $40 million over 1997.
- Communities nationwide lost over a billion dollars in 1998 tax revenues.



(a) World Piracy Rates



(b) Dollar Losses by Region

Fig. 1. BSA Piracy Study [2]

From another perspective, most corporate users these days buy licenses for a certain number of copies. The software vendors need some mechanisms to ensure that the licenses are honored, and at the same time, the user is not too inconvenienced – that is, it might be okay for the license to be invalidated over a few applications, if the user is willing to buy more licenses keeping in mind the increased need.

These, and many other such considerations demands flexible and robust licensing mechanisms. In this report, we take a look at the different kind of solutions possible in a networked environment, under varying requirements and constraints.

The report is organized as follows: section II presents the licensing problem in a networked environment in detail, section III discusses the various problems posed in such an environment which make licensing difficult to implement, section IV talks about the various approaches of implementing a solution. We take a brief look at some existing solutions in section V and then present a peer-to-peer protocol in VI. This

is followed by a discussion of issues in central server based approaches in section VII and VIII. Some future extensions and suggestions are presented in section IX.

II. The Licensing Problem

A. Motivation

Throughout this report, we will assume that our users are not malicious – that is, users are not hell-bent upon cracking the licensing mechanism. Instead, our focus is simply to make sure the licensing agreements are enforced – and if they are violated by the user unintentionally, then we should be able to take some actions. As we shall see, no licensing model is completely secure, and more often than not, it is not difficult to break them.

Most licensing models are counter-intuitive, and hence not very widely accepted. One of the popular licensing schemes, for instance, allows a piece of software to be installed on a single machine. A more natural – and thus more acceptable – licensing model might enable the owner of the license to use the corresponding piece of software like a book: anywhere, i.e. on any of his/her computers, but only at one location at the same time.

An obvious generalization for multi-user licenses follows in context of networked environments – the software may be installed on many machines, and users may use any machine to run it. But at any instant, the total number of users actually running the application should be less than the allowed maximum. This motivates the network-license problem.

Thus, we now have new concepts of software licensing:

1) software is a network resource – licenses ”float” on the network,
2) software costs are a function of how many users simultaneously run the software – pricing is based on users, not CPU performance,
3) value exists in the use of software, not in the number of copies on disk or tape.

Clearly, this requires dynamic tracking of users and software licenses. Note that license management differs from copy protection because it controls execution instead of copying.

B. Definitions

- **LAN**:A Local Area Network is a group of computers and associated devices that share a common communications line within a small geographical area.
- **Application instance** An application program in execution is called an instance of the application.
- **License** An entity which encapsulates the license agreement.

C. The problem

With the above definitions, we can now define the problem of networked licensing – given a LAN and a N-user license, we want to ensure that at any time, no more than N application instances are executing. However, there may not be any restriction on the number of machines the software is actually installed on.

D. Requirements and design goals

In addition to the above basic requirement, we would also like the licensing scheme to fulfill the following design goals:

1) Legal use of software should not be hampered.
2) The licensing scheme must be transparent to the user – in terms of application performance, user involvement etc.
3) The licensing mechanism should not cause any unnecessary inconvenience to a user.
4) Robustness and fault tolerance: when servers go down or networks fail, the license manager must automatically reconfigure itself.
5) The scheme should be flexible enough so that it can be easily configured for different environments.

III. Challenges

The very nature of the licensing problem leads to some inherent challenges.

For instance, most software vendors keep their licensing policies secret in the hope of making their schemes more secure. This means that there are no standard protocols or interfaces in place – which essentially forces licensing schemes to be highly individualized, and not applicable in all situations/all applications.

Most of the existing models are proprietary, so the development of licensing schemes becomes slow, since all vendors begin at ground zero on the learning curve.

Almost all licensing schemes are vulnerable in some aspect or the other – simply because the security requirements are too high: in a multi-user network, the number of trustworthy entities is quite low, and there are additional hazards due to sniffing, replay, Denial of Service attacks etc. Hence, it is almost impossible to come up with a totally secure scheme.

The more secure and fault tolerant the licensing scheme, the more overhead it incurs – in terms of number of messages required, synchronization issues, cryptography issues etc. But we want to keep overheads minimum so that the user does not feel any performance degradation.

IV. Approaches

In this section, we will take a look at the contemporary licensing solutions and their classifications.

A. A General Classification

Very broadly, licensing schemes can be classified into two categories on the basis of how the licenses are expended by the users – these are the consumptive schemes and the allocative schemes (c.f [3])

1) Consumptive Schemes: In the consumptive schemes, either some form of electronic money is used for pre-payment or a trusted logging mechanism is used for billing the customer after wards. This type of scheme is relatively straightforward to implement, once the problem of how to enforce it on the user’s side is solved.

However, since such schemes always involve some kind of Prue-payment or accumulated billing, they are not really

good for real-time usage control. Further, it forces software vendors to enter into a continuously involved relationship with the clients since every usage incurs extra cost.

2) Allocative Schemes: Allocative schemes restrict the usage in real-time to a certain upper limit of concurrent users. In this report, we will be focusing on allocative schemes. Allocative licenses may be unlimited in time, or limited in time and users both.

3) Intermediate: An intermediate scheme works with "use-once" licenses. Such licenses may be used to install a trusted licensing system, which then transmits extra information to the vendor – and then some kind of "late-binding" mechanism may be used to extend the license.

B. Licensing Policies

Yet another classification is on the basis of the licensing policy used to enforce the licensing agreement.

- **Node-locking**: Once the most popular form of licensing, parallels the view that software is licensed to a particular computer. Typically, users may remotely login to the licensed computer. This licensing model is typically found in computationally intense applications and with software used on workstations dedicated to a particular application.
- **User-based**: This policy assigns licenses to a specific user-id. This is useful for products that are user dependent, e.g., an e-mail product, or business transaction applications – applications in which "lending" someone a user-id conflicts with the very nature of the application.
- **Site licensing**: software to a geographical site is another approach, one that generally has a high entry price for the customer. Both the software vendor and customer frequently spend a great deal of time negotiating over price, as site licenses generally do not match pricing to actual usage very well. Defining a "site" can also be difficult. For example, a company with a large corporate campus may be so large that multiple sites are defined. Site licenses are most appropriate when companies standardize on a specific product.
- **"Network licensing" and "floating licenses"** are almost synonymous with license management. Floating licenses fit very well with the concepts of networked computing. Floating licenses became the prevailing workstation licensing policy due to its efficient matching of usage to the number of licenses sold. Many customers at a site can have access to a software product, but the cost of this access may be the price of one license. As the software becomes widely used, additional licenses are purchased.

Other licensing policies include specifying an "expiry date" or a "start date" for licenses, used in software evaluations or leasing programs.

There is no "right" licensing policy for all products. Developers decide policies based upon the way the software product is designed and used, and on competitive factors. Sometimes the same software may be licensed with two very different licensing models.

For example, an electronic publishing package may be available as either a floating license or a node-locked license, with the node locked license having a lower price.

License management helps users choose the right software, when software vendors implement extensive customer evaluation programs. This allows users to evaluate a software product for a few weeks before it is purchased.

Users at a site tend to "standardize" on a particular software product that is license managed software because the product is a networked resource that can be used by all, not just those who have a license for their workstation.

System administrators also have tools to control the use of licenses. For example, administrators may make licenses available only to specific groups, or specific users.

C. Classification on Control

Another perspective for classification is based on control flow – which are the entities who process licensing information, perform validation and take appropriate actions.

As in all communication/networking systems, there are two major approaches here – a centralized approach, and a distributed peer-to-peer approach.

1) Central Server based: In the centralized scheme, there is a designated server machine which is responsible for enforcing the license agreement. A single point of control means a single point of failure, which makes the centralized approach less fault tolerant. However, since there is a dedicated machine for control, it is easier to fix problems, upgrade technologies with minimal overhead and change the licensing schemes. Further, since all clients are required to "register" with the central server, it becomes easy to check the authenticity of the clients (using IP addresses, passwords etc)

Again, we can have multiple variants here depending on the location of the central server – whether the central server is on the local Intranet, or it is somewhere on the Internet.

Having a local Intranet server has several implications: on one hand communication is fast and more reliable, and administration becomes easier; and on the other hand, local machines are more vulnerable to attacks and failures.

An Internet based server can be quite slow and hence a performance bottleneck. Further, it requires that all machines running the application have access to the Internet. However, it is much more secure since in most cases this server will be administered by the software vendor only.

All central server based schemes assume that the central server never goes down. This has the advantage that the application now has means to find out whether it is connected to the network, but lonely (that is, its the only application) OR whether it is not connected at all (in which case it will not be able to connect to the central server). As we shall see, this is not possible in the peer-to-peer models, and poses some serious hurdles.

Typically, central server based mechanisms do the following:

- All application clients at start up time try and connect to the central server. If it cannot connect, it reports failure

of the license validation.

- If it can connect, it sends its identification (license number etc) to the central server. Since all clients need to connect to the server for validation, the server has information about all clients currently executing application instances. So it can easily verify both the authenticity and the validity of the client's license. Accordingly it sends back a response to the client.

2) Peer to Peer: In the peer-to-peer (P2P) schemes, there is no central coordinator, and in-effect all participating nodes are equivalent. This increases fault tolerance in the system, but now one has to deal with other issues like synchronization, mutual exclusion and consistency.

Since there is no single point of failure now, the fault tolerance increases but the system becomes more vulnerable and prone to failure. Further, the communication overhead is higher since all machines need to talk to each other to enforce licensing.Also since communication message are visible to everyone, the security threats due to sniffing increase.

Communication can be either by broadcast or by multicast. While broadcast is easier to do, it may increase load on the network when messages are passed across routers. Multicast messages save bandwidth but one has to incur the additional overhead of managing multicast groups. Note that within a subnetwork, multicast and broadcast are more or less equivalent.

A typical P2P scheme will work as follows:

- Any time an application wants to start up or validate its license, it sends out a message to all other applications on the network (via broadcast or multicast)
- Already running applications on receiving such a message determine whether they are using the same license as this application and send an appropriate response
- The application that originated the "discovery" process computes whether it confirms to the licensing agreement depending upon the responses it gets.

D. Classification on Transport

A third classification is based upon the transport used for communication. Depending upon the requirements, designers of licensing scheme may use different transport – for instance, if we want reliable state oriented communication, it makes sense to use TCP; whereas if we want fast, unreliable communication, UDP is the ideal choice.

There may also be systems which communication using high level protocols like HTTP or SMTP (for emails)

E. Other

An interesting way to look at the licensing problem is to model it as a k-mutual exclusion problem. In the k-mutual exclusion problem, no more than k processes are allowed to be executing their critical sections simultaneously. If we say that a process is in its critical section if it is starting up and requires license validation, then the licensing problem reduces to the k-mutex problem.

There are several algorithms in literature for the k-mutex problem [1], [4], [5]. It might be interesting to look at some of these – though the main problem with all these algorithms is that they are designed in theoretical frameworks and may not work well in practice.

V. Existing Systems

In this section, we take a look at two existing solutions – one is a P2P solution based on the Name Binding Protocol (from Apple) and the other is a central server based scheme.

A. NBP

The Name Binding Protocol from Apple is part of the AppleTalk suite of protocols. NBP is used by AppleTalk advertise resources, such as printers and file servers, to the network. Any resource that other users can access will have NBP information that must be communicated to other nodes.

The software licensing problem requires us to prevent multiple copies of an application running on a network.We need some mechanism to declare to the network the applications that we are running.A simple method to ensure this is to register a fictitious device on the network using NBP with the name being single serial number of the license. Other attempts to register the same device and serial number give an error that the program acts on to deny the use of the program.

The main advantage of a NBP based solution is that communication overhead is minimized since there is no dedicated message passing happening. Since we are using OS based services for communication, the cost of broadcasting or announcing is not much.Also the solution works for a peer-to-peer setting requiring no dedicated servers as are needed in other solutions making this solution is lightweight and user friendly

This approach also has some very obvious problems.One can easily write a simple program to unregister an application and continue to use the same license number.A denial of service attack can also be easily be mounted.

Hence there is a trade off between user-friendliness and security and clearly this solution is applicable in settings where we are more concerned about of ease of use.

B. FullyLicensed ™

Fully Licensed technology ([6]) is a centralized server based scheme. The fundamental idea is to separate the distribution of a piece of software from the distribution of the corresponding licenses. End-users acquire copies of software products by download, whereas the licenses remain on the license server.

Each time a piece of software is invoked by an end-user, the license enforcement mechanism contacts the centralized server via the Internet. It is then verified that the end-user is in possession of a valid license for the software to be run.

The license server provides a web based user interface that enables a subject participating in the system to trigger license transactions such as transferring one of its licenses to another subject.

Software licenses are stored in a license database integrated into the license server. For each license, its entry in the database contains the characteristics of the license, e.g. the expiration date or the licensed piece of software. In addition, the entry identifies the subject currently in possession of the license. Thus, transferring a license from one subject to another subject, for example, is easily realized by substituting the former by the latter as the current possessor in the matching database entry.

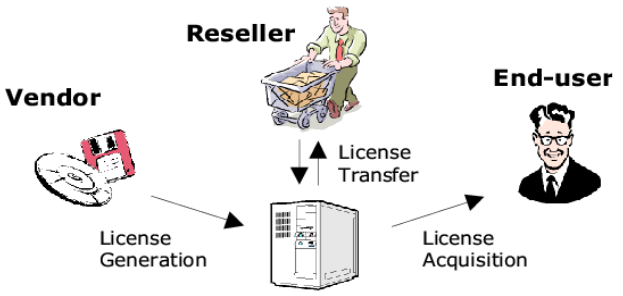


Fig. 2. FullyLicensed technology [6]

To reflect a license distribution policy, each database entry further contains a set of permissions. The permissions specify which transactions the subject possessing a given license is, according to its role, entitled to perform. In this way, arbitrary distribution policies can be modeled and enforced.

The license enforcement mechanism creates a link between the software downloaded by an end-user and his or her licenses stored in the license database. Any program automatically contacts the license server at each of its invocations and requests permission to run. The license server then searches the database for a license possessed by the end-user that authorizes his or her use of the program. If a valid license is found, the server grants permission to run and otherwise denies permission. In the latter case, the invoked program refuses to run.

VI. A Peer-to-Peer solution

A. Assumptions

- All systems are interconnected through a LAN.
- There may be more than one different type of licenses being used by clients.
- Maximum number of allowable hosts and secret key is known to all applicants.

B. Definitions

- **License** The license agreement to be enforced.
- **License Key** A secret key encoded in the license number itself. All clients using the same license number share the same license key.
- **License Number** A unique, public number which the application for each different license (or, equivalently, each application may have just one unique license number) The license number contains some information encoded in its structure which makes it possible to verify the validity of a license number. Among other things, the license number stores the license key and the maximum number of allowable users.
- **Aspirant application** An application which is just starting up and wishes to find out whether it is allowed to start or not.
- **Mature application** An application which has successfully started in conformance with the license agreement.

Algorithm 1 Peer to Peer protocol

$Peer$:
proc *synchCheck* $\equiv$
  Broadcast DISCOVERY
  Wait for responses
  Take appropriate action
□
proc *asyncCheck*$(timePeriod, callBackFunction)$ $\equiv$
  spawn a separate thread which performs checks after every timePeriod
  if *max users exceed*
    then call *callBackFunction*
  fi
□
proc *startListener* $\equiv$
  start the listener thread
□
proc *stopListener* $\equiv$
  if $($*already running*$)$
    then *suspend listener thread*
  fi
□

C. The Protocol

The basic idea is very simple:

An aspirant application broadcasts a `DISCOVERY` packet on the network. The structure of the discovery packet is as follows – [License Number, Time stamp, Checksum] – where the checksum is calculated using the secret key. This makes sure that if the packet is tampered with en route, then the recipient mature application can detect it. Also, it will help mature applications authenticate the identity of an aspirant application.

A mature application on receiving a discovery packet, sends back a `DISCOVERY RESPONSE`. The structure of the response is as follows: [IP, Received Timestamp, Cryptographic checksum] Here again, the cryptographic checksum is computed using the license key. The time stamp used here is the time stamp received in the discovery packet. This will help in detecting replay.

The aspirant applications simply counts the number of discovery responses it gets and using its knowledge of the

maximum number of allowable users, it decides whether it should start up or not.

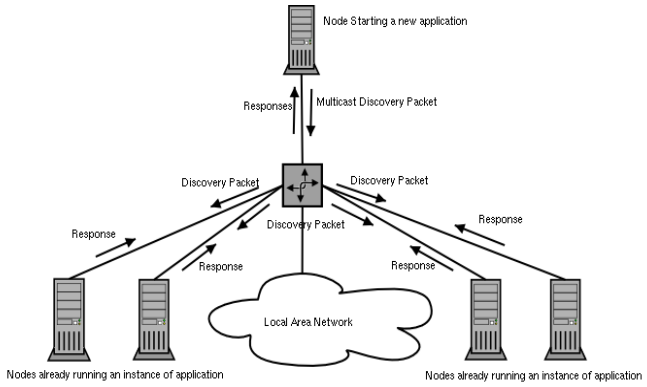


Fig. 3. A Peer to Peer licensing protocol

D. Issues

When the very first aspirant application comes up, it will not receive any response whatsoever. Hence, we will have a timeout on the discovery packets. If no response is received within a particular time, then the aspirant application will assume that it is the only application and hence will startup.

Note that this situation is indistinguishable from the case in which the aspirant application is blocked behind the firewall – in that case, even if there are mature applications on the network, the aspirant application will not receive any response. If the number of mature applications is already maximum, then this will violate the license agreement. Clearly, unless there is some mechanism to detect whether we have been blocked behind a firewall at the application level, it is very difficult, if not impossible to solve this problem.

A denial of service attack is prevented since it is impossible to forge a discovery response (the license key is secret). A sniffer doesn't gain anything by replaying the packets because the aspirant application on receipt of a discovery response can easily check the "freshness" of response using the time stamp contained in it.

Also note that in this protocol, there is no persistent information being stored about mature applications currently executing. In some sense, this makes the protocol more secure.

There may be a special case where the number of mature applications is already equal to the maximum allowable number. Now, suppose an aspirant application comes up and sends out discovery packets. If one or more of the mature applications crash/die just after sending a discovery response, the aspirant application will still not be able to start up because as far as it knows, the maximum limit has already been reached.

Of course, in the converse scenario where the aspirant application suddenly crashes, there is no problem.

Now consider the case when there are applications using different licenses. Since the discovery packet includes the license number of the aspirant application, a mature application can use this license number to check if it is itself using the same license or not. If so, only then will it respond with a discovery response. Otherwise, it will just ignore the discovery packet.

E. Implementation

To test the protocol, we implemented a LicenseManager library which implements most of the functionality discussed above, and some sample applications to demonstrate the licensing capabilities. All development work was done using Visual Studio .NET on Windows XP.

The LicenseManager library provides a clean interface to the calling applications, which makes the life of application developers very easy – the entry point to the library is through a single function (or one of its variants) and thats it! Everything else (sending discovery packets, handling responses, timeouts etc) is handled by the underlying library making the application totally transparent.

The architecture of LicenseManager is as follows: The start function requires the license key (a string in this case) and the maximum number of allowable users. Of course, as mentioned in section VI-A we assume that each application has knowledge of the number of maximum users, or is able to compute this information from the license number.

At this point, the application developer has two options – he may wish that the application blocks (waits) until license has been validated, or he may want that the function returns immediately and the application proceeds normally, and when the licensing information has been processed, then somehow the application is informed about what happened.

The basic idea is that the user of the application should not get unnecessarily inconvenienced by the license validation procedure. Moreover, most software vendors would be more interested in simply logging the usage information so that they may monitor license confirmation patterns and ask the client to buy more licenses if maximum users is exceeding very frequently – only in extremely rare cases would one want to actually do something drastic like terminate the application or something like that.

Another functionality that comes extremely handy in making applications license aware is to be able to make periodic checks on the network for number of users. Clearly, such a periodic check has to be asynchronous. Similarly, synchronous methods have to use some timeout mechanisms to make sure that the application does not block forever.

To provide all this functionality, LicenseManager has a rich set of options:

- `startListener()`: starts the handler for discovery packets
- `stopListener()`: stops the handler for discovery packets
- `asyncCheck()`: asynchronous method to check number of users on the network. It takes a *periodicity* argument which specifies the interval at which the network should be monitored for users. After every interval, if the number of users exceed, then a user specified function is called (also passed as a parameter to the function). Examples of such a call back function could be:

1) `logMessage()`: Log the message in a file
2) `notifiy()`: Notify the user (pop up a message box, for instance)

- `syncCheck()`: This is a synchronous method which blocks until the timeout has expired or all responses have been received.
- `resetAsync()`: To reset/suspend the asynchronous check.

The actual communication between the participating machines may take place through broadcast or multicast as discussed in section IV-C.2. In the multicast approach, one has to make sure that different applications using the same license number use the same multicast address and port for communicating. For this, we use a simple hashing mechanism to hash the license number into a multicast address and a port.

A small problem might occur here if some other application has already bound with the same port as our application. However, the probability of this occurring is not very high. To be on a safe side, we use two ports simultaneously for sending and receiving. The chances that *both* these *hashed* ports are already in use is very very small – though the possibility of this happening can not be denied.

VII. Server Based Licensing

A server based licensing mechanism offers several advantages over a peer-to-peer mechanism. The most important advantage is that a server based solution can provide a lot more flexibility and security compared to a P2P solution. Moreover, a P2P scheme is not entirely reliable because it is easy to end up with inconsistent states where different machines behave differently. It is also easy to upgrade/change the software for license management at one place than on all machines in a LAN. And when the number of applications becomes very high, then the communication costs in a P2P scheme far exceed than that in a server based scheme, since the number of messages in a P2P scheme are much larger. So a server based solution is more scalable.

Most server based licensing solutions typically work as follows: The client sends a request to the licensing server, which includes the license number of the client. Since all clients need to contact the license server for license validation, the license server has a complete knowledge of the licenses in use over the network at any particular time. Therefore, it can decide whether or not a new application is allowed to start up or not. So the server sends back an appropriate response to the client. Depending on this response, clients may take necessary actions. Alternatively, the server may simply report the number of concurrent users using that same license number and the decision making may be left to the client application.

There are many issues to be considered in this simplistic scheme.

A. Location of the License Server

The license server may reside on the local LAN (Intranet) or on the Internet. Both the schemes have their pros and cons. With an Intranet server, the responsibility of administration and license management falls upon the users. So the risk of tampering or violation of the license agreement increases. However, management becomes easier because the hardware is under the control of the user. Moreover, the communication is faster and more reliable. An Intranet solution is also more convenient to the user compared to an Internet version because the problems can be addressed in real-time, whereas with an Internet server it may take days and weeks for a problem to get resolved.

On the other hand, an Internet based server is more reliable (in terms of the enforcement of the license) since it is under the direct operation of the software vendor. However, the communication is often slow and broken. Further, an Internet license server may cater to multiple subnets over distributed geographical locations simultaneously. Hence, it is more suited for large scale wide-spread applications. Another issue that is attached to the location is that of discovery of the license server. For a local server, it is possible to discover the server by a broadcast mechanism. However, this cannot be done across networks, or over the Internet. So in this case, either the license server has to be hard-coded inside the client, or there is some other mechanism to inform the client about the server's location, like a configuration file etc.

Now consider a situation in which more than one license servers are available - so how will licensing proceed in this case? Will the clients be distributed between the two servers, or will one server just act as a backup for the other one, replicating all its states so that it can take over in case of a failure. In the latter case, how is the consensus reached as to which server should act as the primary server? A trivial implementation may follow a first-come-first policy where the first server to declare itself primary becomes the primary server.

B. Authentication

A central server based scheme necessarily needs to have an authentication mechanism for establishing both the identity of the server as well as client. A client must ensure that the server is authentic because a fake server can always tell the client not to start up even when the licensing constraints are satisfied. On the other hand, server has to make sure that it registers only genuine clients so that the authentic clients are not denied the service. There are many approaches to implement authentication. Most of them are based on a challenge-response protocol or a public/private key encryption mechanism, where the challenge/keys are usually derived from the license numbers.

C. Fault Tolerance

Since the License server needs to have an up-to-date status of licenses on the network, all mature applications need to send periodic heart beat messages to the server saying that "I am alive". Note that in such a scheme, applications are completely dependent on the License server for their operation. Hence if the License server goes down or crashes, then the applications may not be able to start up. So there has

to be some backup mechanism. Further, after the License server crashes and comes back up again it must have some mechanism of knowing the state of the system. If possible, it can save its state in some persistent storage so that if it comes back up again in a short time , it can restore itself to the last consistent state.

Algorithm 2 Server based Licensing: Server Protocol

Note that in algorithm below we can also use the PING messages in other direction i.e. from client to server.

$Server$:
var
  comment: Hashtable of Vectors for storing clients
  $LicenseStatus$
  comment: Hashmap of active clients
  $LicenseInfo$
end
procedures
comment: Listen and handle messages
$MainThread$
comment: Periodically updates the status of clients
$UpdateStatusThread$
endprocedures
proc *MainThread* $\equiv$
  On receive of client request (VALIDATE_LICENSE) $\equiv$
    authenticate client
    validate license and add entry in LicenseInfo
    send response depending on policy
  □
  On startup $\equiv$
    Broadcast SERVER_START
    Wait for responses
    if $(REGISTER)$
      then *update LicenseStatus*
    elsif $(ALREADY\_RUNNING)$
        $shutdown$
    fi
  □
  On receive of SERVER_START $\equiv$
    send ALREADY_RUNNING
  □
□
proc *UpdateStatusThread* $\equiv$
  foreach *entry in LicenseStatus* **do**
    if *(entry is stale)*
      then *delete the entry*
    fi
  od
□

Another idea would be two have both server based and peer-to-peer licensing mechanism co-existing in the network. So when the server goes down, the applications might switch to a peer-to-peer mode. And when the server comes back up, it is informed of the new state and the network switches back to the server based scheme. Note that for this to be possible the applications which have already started will either have to keep track of the state of the server or just be ready to do a peer-to-peer exchange anytime.

Of course, this is easier said than done. To implement such a system, one would have to take care that the licensing scheme is enforced at all times, or at least violated only within certain constraints. There would also be synchronization issues, since all applications must use the same mechanism - it should not be that some applications are running peer-to-peer based licensing while others are still waiting for the server to come up. Also, when the server comes up, it has to be informed of the new state of the network.

A possible method of implementing co-existent licensing mechanisms is as follows: All systems will run a peer-to-peer component at all times. This component will be effectively defunct so long as the application is in contact with the server. Now, when the server goes down (or equivalently, when the applications thinks the server has gone down), it will start using the peer-to-peer component. Consider the case when some machines in a network think that the server is down, while others can still communicate with the server. Then, the machines which can communicate with the server can act as proxies for any communication between the server and the other machines. So if any new application broadcasts discovery requests, these may be forwarded to the central server.

At the same time, each machine can maintain the current state of the network as known to itself. Also, each machine periodically checks if it can reach the server. So when the server comes back up, any application which connects to it first can inform the server of the most recent state of the network. In this way, the server will know how many connections should it expect to regain consistency. A small catch here is that not all applications might be able to connect even when the server comes back up. And if during this time, a new application contacts the server, then it may lead to an inconsistent state.

To circumvent this problem, the server will withhold any requests it receives after starting up, till a certain timeout value - this timeout value will be at least as large as the time period of checks to be made by already running applications. So after this timeout, the server will assume that whatever state of the network is has perceived is the actual state. If an application which was running previously tries to connect to the server after this timeout, then it will be treated as a new application. Clearly, this is not an ideal protocol, since it may be unfair at times. However, it is almost impossible to devise any scheme in which the server can restore the most recent consistent state in presence of message losses or application crashes.

D. Policy Server

Till now, we have either been assuming that the license decisions are taken by the server, or that the license server simply report the number of applications using the same license number and the actual decision is made by the client. Both of these situations reduce the flexibility of the licensing

mechanism. Ideally, we would want that the various licensing policies reside in a convenient format on a policy server, and the applications may request a particular policy from the server. The format for storing/transmitting the policies must be such that not much overhead is incurred in actually implementing that policy. This means that parsing the policy file should be fast and efficient. Also, the language used to write the policies must be flexible enough to support the majority of licensing policies.

A possible representation of the policy files may be on the lines of the format used by the Java Runtime Environment. Another possibility is to use an XML based encoding for the policies. However, both these representations would incur additional overhead in parsing and interpreting the data. If the policy decisions are simple enough, then we can simply use a few bits to encode the type of the policy to be enforced within the license number itself.

E. License Generation and Maintenance

Suppose a company already has some licenses, and acquires some more. How much effort would it entail to configure the license server to handle these new licenses? Can the process be made completely automatic, or some kind of manual assistance is needed. Another key issue is that of user involvement - does the user have to do some active registration on the license server, or the whole procedure is completely transparent to the user?

VIII. Implementaion Server based Licensing Protocol

Most server based licensing solutions typically work as follows:

The client sends a request to the licensing server, which includes the license number of the client. Since all clients need to contact the license server for license validation, the license server has a complete knowledge of the licenses in use over the network at any particular time. Therefore, it can decide whether or not a new application is allowed to start up or not. So the server sends back an appropriate response to the client. Depending on this response, clients may take necessary actions. Alternatively, the server may simply report the number of concurrent users using that same license number and the decision making may be left to the client application.

In this section, we propose the details of a server based licensing solution. The main issues to be considered are those of security and fault tolerance. The assumptions that we make are as follows:

1) License Server is Intranet (LAN) based, not Internet.
2) Network partitions do not occur - i.e, it does not happen that a few machines see that the server is alive while others cannot see the server.
3) The policy decisions are quite simplistic - allow, disallow, log, email etc. Hence there is no need for a complex representation of the policy. It can be easily embedded into the license number.

Algorithm 2 Server based Licensing: Client Protocol

```
Client :
var
   serverIP      //comment: Needs to be discovered
   serverAlive   //comment: true if server is alive, false otherwise
end
On startup() ≡
   Broadcast VALIDATE_LICENSE
   Collect the response and check authenticity of the server
   if (authentic)
      then set serverIP
           Depending on response take action
   fi
   if (no valid messages received within certain timeout)
      then use exponential backoff and repeat above steps
   fi
   if (no server found still)
      then
           serverAlive = false
           Broadcast DISCOVERY to peers
           Wait for response and take action accordingly
   fi
□
On receive DISCOVERY ≡
   if (authentication on ∧ client authenticated)
      then
           setserverAlive = false
           Respond to the client if has same licenseNum
       else
           check(if server is dead)
           if found true set
              then
                   serverAlive = false
                   Respond to the client if has same licenseNum
           fi
   fi
□
On receive PING ≡
   send(PING_ACK);
□
On receive SERVER_START ≡
   setserveAlive = true
   send(REGISTER)
□
```

4) A symmetric key is hard coded in all the applications as well as the license server.

We will describe the protocol by detailing the behavior of the License server and the clients. For security, we will use a symmetric key mechanism; and for fault tolerance, we will have a fall back peer-to-peer based licensing protocol running on the clients.

A. Server Protocol

The server waits for client connections on a fixed port On an incoming request, it verifies the authenticity using the symmetric key Since the policy is now encoded within the license number, the server can decide what action to take depending on the number of applications currently using the same license number. So it can send an appropriate response to the application. When the server comes up (perhaps after a crash or even otherwise), it announces its presence by a broadcast message on the network. This is so that applications which might be running in P2P mode may switch back to server mode. Server periodically interrogates the clients through a broadcast to get a fresh state of the network.

B. Client Protocol

All clients first try to contact the license server. Possibly try a few times using an exponential back off. If able to connect to the server, client waits for server's response. Depending on the application semantics, the client may block for a response, or may continue with start up and call back a function when the response arrives. If a client is not able to connect to the server on startup, it assumes that the server is dead and broadcasts a P2P discovery message on the LAN. A mature client keeps the P2P discovery port always open. When it receives a DISCOVERY message, it believes that the server is dead (by our assumption of no network partition) and sends back a response to the aspirant applications The P2P back end proceeds exactly as outlined in our previous document with minor changes. Switching from P2P to Server mode simply means setting a bit - that is, the listener thread and other components of P2P are still running, only defunct.

IX. Future Directions

As we saw in section VI-D, we can't easily distinguish between an isolated machine, and a lone application on a network. These days more and more people are using personal firewalls on their machines. Suppose that such a firewall blocks the set of ports that we are using (intentionally or otherwise) – how can we handle such cases? Since we can not easily find out if we are lonely or blocked, it is not easy to answer this question.

One possible solution is to use some kind of port-hopping scheme where in the licensing library changes the port(s) it uses in every session, or even between sessions. However, the major problem with this approach is that we will have to somehow synchronize this hopping pattern across applications.

In the server based solution, the scheme needs to be made more robust and fault tolerance if it is to made actually *usable and deployable*. One way of increasing fault tolerance is to have a back up server. For this, we need to design an efficient and reliable replication mechanism by which the data on the two servers can be synchronized. We may also use some persistent storage in combination with message logging and checkpointing to make sure that when the server comes back up again, it starts in a consistent state.

A. Plan for next semester

1) Fault tolerance in server based mechanism using replication at back up server
2) Increase robustness by having persistent storage, message logging and check pointing.
3) Design of a Policy server and appropriate language for encoding complex policies

X. Conclusion

A good licensing scheme must be user-friendly, at the same time flexible and robust. However, contemporary licensing schemes are counter-intuitive and hence, difficult to use. Further, it is almost impossible to design a fool-proof licensing mechanism. So one should first prioritize the requirements and then focus on those.

The two popular models for licensing are central server based, and peer-to-peer distributed systems. Both the models have their pros and cons. While a peer-to-peer based model requires minimal set up, it is not as robust or flexible as the server based mechanism. There is a trade-off between security and performance overheads, and the final decision should be based on the requirements and the environmental constraints.

Acknowledgements

We are grateful to Dr. Manindra Agarwal, Dr. Deepak Gupta and Dr. Dheeraj Sanghi for their guidance throughout this work. We would also like to thank Adobe India for their support for this project. And special thanks to Debyajoti Bera and Arvind Jha for their cooperation and enthusiasm.

References

[1] D. Agrawal and A. E. Abbadi, "An efficient and fault-tolerant solution for distributed mutual exclusion," *ACM Transactions on Computer Systems (TOCS)*, vol. 9, no. 1, pp. 1–20, 1991.

[2] BSA, "Sixth annual bsa global software piracy study," May 2001. [Online]. Available: http://www.bsa.org/resources/2001-05-21.55.pdf

[3] R. C. Hauser, "Does licensing require new access control techniques?" in *Proceedings of the 1st ACM conference on Computer and communications security*. ACM Press, 1993, pp. 1–8.

[4] H. Kakugawa, "A study on distributed k-mutual exclusion algorithms," Ph.D. dissertation, Hiroshima University, Feb 1995. [Online]. Available: "citeseer.nj.nec.com/kakugawa95study.html"

[5] H. Kakugawa, S. Fujita, M. Yamashita, and T. Ae, "A distributed k-mutual exclusion algorithm using k-coterie," *Information Processing Letters*, vol. 49, no. 4, pp. 213–218, 1994. [Online]. Available: citeseer.nj.nec.com/kakugawa94distributed.html

[6] T. Lopatic, "Fully licensed technology whitepaper i," Fully Licensed GmbH, Tech. Rep., Aug 2001.

[7] Microsoft, "Software piracy continues to impact communities across the country," Microsoft Corp., 1999. [Online]. Available: http://www.microsoft.com/presspass/press/1999/sept99/impactpr.asp