XLattice

Table of contents

1 Overview....2
2 Recent News/Changes....2
3 Model Applications....6
4 Nodes....7
5 Overlays....7
6 Protocols....7
7 Transport....8
8 Encryption....8
9 Implementation Languages....9
9.1 Java....9
9.2 C/C++....9
10 Platforms....9
11 Project Status....9

1. Overview

XLattice is a communications infrastructure for peer-to-peer (p2p) networks. These networks may be server-assisted. XLattice is a modular system consisting of components intended to be usable for a wide variety of purposes, ranging from p2p chat to distributed software development environments.

The software will also be useful in applications requiring anonymity, privacy, and filtering, such as messaging systems for business.

2. Recent News/Changes

2007-07-10	**corexml-0.3.5 release** We add NameGenerator to corexml.bind. This is a facility for generating class, method, and variable names in a standardized way. These names are to be used in various types of automatically generated code, as for example in other bind methods; in XLattice's ProjMgr component; and in the forthcoming JAppGen J2EE MVC application generator project, jappgen.sourceforge.net.
2007-07-10	**util-0.3.9 release** This release adds ucFirst and lcFirst methods to StringLib. These are used fairly heavily by upcoming corexml and projmgr component releases, which are in turn used by the JAppGen J2EE MVC application generator, about to become available in alpha at jappgen.sourceforge.net.
2006-06-21	**corexml-0.3.4** This is a minor release. context/ has been moved to the util component, and necessary changes in code have been made. We switch to using junit-4.1 for testing, in line with the earlier switch to Java 1.5. And the release is now packaged as a ZIP file instead of a UNIX tarball. As usual, we include source code for this component and jars for all dependencies.
2006-06-21	**projmgr-0.4.0** ProjMgr's first update in a couple of years sees util/context/ replacing corexml/context/, JUnit updated to 4.1, and the use of Java 1.5. We also relax restrictions on version numbers; any JNLP version number is now acceptable. Finally, a number of

	minor bugs have been fixed. This is a clean-up preparatory to using ProjMgr for project management (meaning largely automatic generation of configuration files) in the soon-to-be released JAppGen project. As usual, the .ZIP file includes source code for this component (ProjMgr) and jars for all dependencies.
2006-06-19	**util-0.3.8** v0.3.8 makes minor changes to fundamental XLattice classes, including Transport, Peer, and NodeID. We switch to using JUnit 4.1 for unit testing. The release is now packaged as a ZIP file instead of a UNIX tarball.
2006-06-15	**crypto-0.1.2 release** This release adds TLS contexts and sessions which considerably reduce the effort required to implement peer-to-peer TLS connections between XLattice Nodes as well as SSL connections between XLattice-based HTTP clients and servers.
2006-06-04	**crypto-0.1.1 release, switch to Java 1.5** Today's crypto-0.1.1 release improves the implementation of SignedLists and adds the TLSEngine which will be used as the basis for most node-to-node communications in XLattice networks. This is our first release using features of Java 1.5. Excepting only the CryptoServer, all code in CVS has now been tested using Java 1.5. All future releases will assume Java 1.5 or better.
2006-05-24	**node-0.1.0** This is the first release of XLattice's node component. A node is the key building block in our approach to peer-to-peer networking. A node has a cryptographic identity (an RSA key and a 160-bit SHA1-derived node ID) and can communicate with other nodes through one or more overlays. An overlay is a combination of a transport, a protocol, and an address space. Nodes from this first release communicate using TLS/SSL and UDP. This release includes source code for the node component and jars for all earlier components used, as well as unit tests used during the build process. The unit tests are also working examples of how to use the XLattice code.

2006-05-21	**transport-0.1.1** We add both blocking and non-blocking UDP to the transport component.
2006-05-05	**Whoops!** There was a serious bug in 0.1.5 making it difficult to run the server. v0.1.6 of the protocol/stunplus componet went out today. The 0.1.6 server is running on stun.xlattice.org.
2006-05-04	**v0.1.5 protocol, STUN+ GUI client releases** This is version 0.1.5 of the STUN+/protocol package. It corrects bugs reported and makes most enhancements requested through the Sourceforge tracker. It also adds the first elements of the XLattice messaging protocol. The GUI client now supports encrypted authentication using TLS/TCP. It also allows users to select most of the standard STUN servers through a pulldown menu. We would appreciate any further suggestions for GUI client enhancements after user experience with the package. As usual, there are three files in the release: protocol-0.1.5.zip, stunplus-0.1.5.zip, and stunplus-current.zip.
2006-04-25	**protocol, STUN+ with GUI client releases** The 0.1.4 protocol release adds a GUI client and corrects all of the more serious limitations reported through the Sourceforge tracker. We add a facility for testing binding lifetime, but this has not been thoroughly tested. The GUI client should run on any host with a reasonably recent version of Java. We would appreciate feedback from users. Server discovery is supported in this release but limited to UDP servers in the GUI client. The command line client supports encrypted authentication using TLS. The GUI client as yet doesn't. As usual, there are three files in the release: protocol-0.1.4.zip, stunplus-0.1.4.zip, and stunplus-current.zip.
2006-04-16	**STUN+, protocol releases** These are bug-fix releases of the protocol component.

	As usual source code is packaged as protocol-0.1.3.zip and the binary release is stunplus-0.1.3.zip AKA stunplus-current.zip.
2006-04-15	**component releases** Because Sourceforge CVS is currently running a couple of weeks behind, we put interim releases of several components on Sourceforge today. These are • util-0.3.7.tar.gz • corexml-0.3.3.tar.gz • crypto-0.1.0.zip • transport-0.1.0.zip All of these been tested on both Linux (Debian) and Windows (XP). The **util** and **corexml** releases are packaged as tarballs solely due to lack of time. In the next release they will be .zip files.
2006-04-05	**STUN+ and protocol third releases** This is the third set of XLattice protocol releases in this series. It incorporates what is as far as we know the first open source version of STUN with TLS-based authentication. As usual, there are three release: source code release as protocol-0.1.2.zip, binary as stunplus-0.1.2.zip, and stunplus-current now mirrors the binary release. The binary package should run on any machine with a recent Java run-time (1.4 or better JRE). This code has been developed and tested on Debian Linux hosts. There are several known limitations which will be documented on the Sourceforge project tracker. These will have been corrected and the code tested on Windows XP prior to the next release.
2006-03-10	**STUN+ and protocol second releases** These are the second releases of the XLattice STUN client and server. The source code release is protocol-0.1.1.zip. This is accompanied by a new binary release, stunplus-0.1.1.zip which, as before, is mirrored by an update to stunplus-current.zip. That is, stunplus-current is now identical to stunplus-0.1.1.
2006-03-07	**STUN+ and protocol releases** We published a preliminary release of our STUN client and server on Sourceforge today. The source code was released as protocol-0.1.0.zip. The STUN-specific scripts and documentation plus the jars necessary to run the client and server were packaged as two separate but identical releases. The first is stunplus-0.1.0.zip. The second is

	stunplus-current.zip. This will be kept in sync with the current numbered release (0.1.0 at this time).
2006-02-08	**STUN and Kademlia** Implementation of the STUN protocol is well underway; code can be found in CVS. An overview of STUN is now available here and design specifications for Kademlia here.
2006-01-30	**Web site changes** The Web site is now being generated using Apache Forrest, which has resulted in some glitches. The process should be completed in a week or so. The older version of the Web site continues to be available at www.xlattice.org and will remain unchanged for some time.
2005-09-22	**CryptoServer 0.0.11 release** This is a minor release to fix a bug that has shown up in production use of the server.
2005-03-11	**CryptoServer 0.0.10 release** This is a major upgrade which is now being used to support www.xlattice.org. That is, xlattice.sourceforge.net runs on Apache, but www.xlattice.org runs on CryptoServer 0.0.10. This is a caching server and a secure server: pages are loaded by content key, site build lists are digitally signed, and the master key is on a different machine several thousand miles away.
2005-02-15	**CryptoServer 0.0.9 release** A substantially improved version of the CryptoServer. It now supports HTTP/1.1 GET and HEAD methods and has been tested with a number of domain names. Most HTTP headers are recognized.

3. Model Applications

XLattice is to some degree still in the planning stage; it is not fully specified. To advance this process, we have sketched out a number of model applications. These are meant to be typical of the applications that can be built on top of XLattice. They are not necessarily part of the XLattice project. They should be understood as *use cases*, aids in specifying and developing XLattice's external interface, its API.

4. Nodes

An XLattice node is a network component that can send and receive messages and has some (potentially shared) local store. The node has at least one logical address; such addresses may contain some sort of physical address, such as an Internet Protocol (IP) address and port number.

There are several types of XLattice nodes. An XLattice network will typically connects mix of nodes of various types. The client node, used to connect users (human beings), is the most basic type. Most networks will also include servers of one type or another.

5. Overlays

As used here, an **Overlay** consists of

- a protocol, as described below; essentially this is a set of message types
- an address space, a set of addresses
- a transport mechanism such as TCP/IP or UDP
- and optionally a set of rules for routing the messages

As an example, a set of nodes might be organized as an N-dimensional hypercube or torus. In this case each would have an address specifying its position on the hypercube. Messages could be routed through the neighbor nearest to the destination node.

Overlays may be nested. The hypercube, for example, might be overlaid on an IP network, the Internet.

Where an overlay allows messages to be routed to specific nodes, then it can be used as a transport.

The expectation is that most nodes will belong to a number of overlays and that many overlays will be relatively short-lived, with lifetimes ranging from tens of minutes to a few hours.

Most node-node communications are expected to be direct. Where broadcast is involved, messages might be sent to peers in distinct sub-hypercubes, so that the load is distributed evenly.

6. Protocols

Note:

What follows is a preliminary description of our use of the term **protocol**. We have been implementing a number of protocols, including STUN, to get a better understanding of what the term should mean; click here for the current state of play.

As we use the term, a **protocol** is a set of message definitions and rules governing the sequencing of the messages.

Message definitions specify how messages are constructed and serialized into and deserialized from wire format. In other words, a message definition tells you how to interpret the binary data sent over the network.

Individual messages normally form part of a sequence. The protocol will specify which message types start such a sequence and which may follow. Generally such sequences branch. The type of reply sent as well as its content will vary according to the state of the recipient and the content of the message received. Almost always, for example, the recipient will examine the incoming message to see that it is well-formed. If not, either the message will be ignored or an error reply will be sent. Otherwise some other action will be taken, which may include a reply in an appropriate format.

A protocol is distinct from the underlying transport. The HTTP protocol used to carry Web traffic, for example, is usually implemented over TCP/IP. However, HTTP could also be used to manage information transfers over SMTP (the email transport protocol) or any other transport, so long as both parties to the communication agreed.

On the other hand, a transport always has an underlying protocol which is in general not visible through the transport's API.

7. Transport

A **transport** is a block of software that enables a node to send and receive messages. Transports are either block or stream oriented. In the first case the transport sends and receives fixed-length blocks of data, arrays of bytes. UDP follows this model. In the second case the application opens a connection which enables it to reliably send or receive an indefinite number of bytes.

The initial implementation will emphasize TCP/IP and sockets. Support for UDP, HTTP, and SMTP (email) will follow.

The assumption is that UDP will provide better performance for many applications. However, some company firewalls will block UDP and some will only permit HTTP.

In the longer term, XLattice will provide a discovery facility that probes node-to-node connections and settles on the best transport protocol available. Some method of expressing a preferred transport will be provided.

See the section on transport for further discussion of the ideas involved; from there you can get to the API and source code from the menu.

8. Encryption

It is expected that most communications will be encrypted, either using standard XLattice components or application-specific modules.

XLattice will provide similar mechanisms for encrypting data in local file systems and support application-specific encryption methods.

However, the initial implementation may only support encryption of messages.

9. Implementation Languages

9.1. Java

The initial reference implementation of XLattice is in Java. The software is being developed incrementally using test-driven development techniques, so that it is thoroughly tested before deployment. The emphasis in Java development will be on correctness rather than performance.

Until very recently XLattice has been being implemented using Java 1.4.2, for greater compatibility. However we have now begun using Java 1.5, principally to make use of non-blocking TLS.

9.2. C/C++

C/C++ development is following the Java implementation. Insofar as possible the same interface and the same tests will be used for both. Interoperability with the Java version of the code is a primary objective.

Further information on XLattice's C++ implementation can be found here.

10. Platforms

For our purposes here, a **platform** is a machine of one type or another running a specific operating system. So a platform might be a Sun Sparc machine running Linux or a box with an x86 processor running Windows.

The intention is that XLattice should run on all common platforms, specifically including mobile (cellular) telephones and J2ME.

Initial development in Java is on x86 hosts running several flavors of Linux (RedHat, Debian, and Fedora).

11. Project Status

XLattice is still fairly early in development, although we have released a couple of applications, including a secure Web server (the CryptoServer) and STUN+. The latter is an implementation of STUN, the IETF's protocol for determining the characteristics of NATs, network address translation boxes that commonly sit between users and the global Internet, especially on broadband networks.

Most components are now available in CVS. Several are also available as separate tarballs containg both Java source code and the jars necessary to run the software, including all external (non-XLattice) dependencies. All software is open source.

At the time of writing, separate component tarballs are largely out of date. However, component jars released with applications such as the CryptoServer or STUN+ are those current at the time of release.

See components for more specific information.