ABSTRACT
This paper describes GIMnet, a software infrastructure designed for the distributed control of generic intelligent machines. GIMnet is an abstraction layer built over TCP/IP, and it combines the Client-Server model with the Peer-to-Peer model. The heart of the system is a software called tcpHub, which provides the connections between different applications in the network. The tcpHub is used to create a Virtual Private Network to allow communication through firewalls. Built on top of the VPN is a communication framework library called GIMI, which has been designed to be easy to use and, more importantly, easy to deploy in various usage environments. This paper describes the software and presents results to show its functionality.

KEYWORDS
Distributed control, teleoperation, remote process communication

1 Introduction
This paper describes a software infrastructure developed for a project called INTEGRATOR. The project aims to provide a mutual testbed for two laboratories which form the Finnish Centre of Excellence on Generic Intelligent Machines (GIM). The Automation Technology Laboratory (TKK/ATL) and the Institute of Hydraulics and Automation (TTY/IHA) are physically 200 km apart. The testbed is intended to serve the GIM for at least the next seven years. The purpose of the testbed is to provide a demonstrational platform for all the research made in the GIM.

Figure 1 illustrates the conceptual overview of the GIM demonstrator. The work-site consists of multiple work machines as well as local human operators. The work-site and its entities are connected to the system that is controlled and supervised from multiple control stations. The concept also includes the use of simulators and virtual environments as supporting tools.

The platform sets a number of requirements for software development. First of all, the infrastructure must allow communication between modules that are situated within the two laboratories. For this alone the infrastructure meets strict demands to allow bypassing the firewalls between the universities. Secondly, the infrastructure should be applicable for real-time control of mobile machines. Thirdly, the infrastructure cannot limit the implementations, as within seven years the researchers want to study various subjects. Finally, the infrastructure must be easy to use and implement if researchers with various backgrounds are to use it.

These requirements have resulted in a software architecture implementation called GIMnet. It consists of a network layer (tcpHub) and a session layer (GIMI) which together form a robust and easy-to-deploy interface for distributed robot control. The tcpHub is an abstraction over TCP/IP with some advanced features designed for this specific task. GIMnet provides a virtual ad-hoc network, which makes it possible to avoid the firewalls, but still allows peer-to-peer communications as well as server-client implementations.

2 Related Work
GIMnet is a complete infrastructure that defines the network architecture as well as a framework for communication between distributed modules. For such systems there exists multiple network architectures, the major types being Client-Server, Peer-to-Peer (P2P), and their derivatives. In client-server architectures clients connect to a service
provider (server) and all action requests are made by the client while the server is just passively waiting. The Client-Server architecture is efficient in one-way information sharing like WWW. The downside of this architecture is that it is not suitable in situations where clients need to communicate with each other.

In Peer-to-Peer architectures, participating nodes have client as well as server functionality. All nodes can share resources directly with each other without a centralized controller. [8] One problem of the P2P architecture is firewalls. The clients must be able to connect to each other, which is practically impossible between isolated networks.

To overcome the problem of firewalls the solution needs some Virtual Private Network (VPN) functionality. VPN is a virtual network topology created over an existing network infrastructure. The purpose of VPN is to hide the details of the underlying network and to allow VPN members to communicate with each other without problems caused by firewalls or such. There exist multiple VPN implementations, both commercial and free. The most popular free VPN implementation today is OpenVPN (http://openvpn.net) which is a full-featured VPN implementation offering both encrypted and unencrypted tunneling. [7] The downside of this solution is that the implementation is always required on both ends of the tunnel. Many embedded systems nowadays contain a TCP/IP stack and C++ libraries, but getting a full-featured VPN implementation to work may be hard or impossible.

An IRC (Internet Relay Chat) network creates a VPN by connecting multiple nodes to a single point and connecting these points together. [6] The network provides a way for clients to send information to each other and still avoids the problem of firewalls (to some extent). IRC is not intended for real-time control, but a similar hybrid architecture is adopted in GIMnet. It creates a virtual private network between the hubs and peers in the system. Each peer only needs to be able to connect to a hub in the system to be able to communicate with every other peer in the system.

Other commonly used frameworks for remote process communication are CORBA and Java RMI. Both of them enable clients to make requests to remote objects as if they were local objects.[1][2] In CORBA this is achieved by the use of object request brokers (ORBs) and in Java RMI with RMI remote object registry. The primary advantages of CORBA are platform independence and interoperability across different programming languages, achieved by defining interfaces to objects with CORBA’s own interface definition language. Both of these solutions require multiple ports open in firewalls. [3, 4] In addition with RMI we would limit ourselves to operating mainly with Java.

3 GIMnet

3.1 Overview

GIMnet is a communication infrastructure designed for robotics applications. Basically, the system is a remote process communication implementation, which additionally functions as a base architecture for the software system.

Figure 2 illustrates the basic idea of GIMnet. The hubs are the backbone of the network. A program called tcpHub performs the tasks of a hub. It only runs on Linux, and it is the only part of the infrastructure that is fully platform dependent. The design of GIMnet allows scaling from a single local hub up to a large network of interconnected hubs. This hub network forms a Virtual Private Network, and the only requirement is that one of the hubs has one TCP port open for connections. The other advantage of this type of “ad-hoc” topology is the scalability and extendibility of the system. As an example, the bottom right box in figure 2 represents a machine in the network. There is a separate tcpHub running in the machine, allowing the internal processes to communicate through it without being affected by the network. When the machine is connected to a larger network, accessing remote modules is programmatically no different from accessing local ones. Thus some of the machine’s functions can be moved outside the machine without the need for any change in the programming. This also facilitates extendability in the sense that more computers can be easily added to the system to share the computing load. For research and development the system offers the possibility to develop and test software on one’s own desktop while keeping it connected to the rest of the system.

The software modules in figure 2 can be considered clients for the hub. The modules are separate processes running anywhere in the network. When connecting to the
hub, the modules register their name and receive an ID, both of which can be later used to address the module. The Network Interface (NIF) encapsulates the low level communication protocols. The Generic Intelligent Machine Interface (GIMI) can be considered an application layer, which provides an easily accessible API for the module developers. GIMI encapsulates the network so that the developers do not need to know the underlying structure; they only use the simple function interface. It also implements some important functions which are required by almost all modules. Below is listed the main features of the GIMnet:

- Distributed name and ID service
- Unicast, multicast, broadcast
- Synchronized and unsynchronized data transmission
- Automatic hub-to-hub and client-to-hub reconnect
- Service registration, subscription and listing
- Application level ping

### 3.2 tcpHub

tcpHub is the “Network layer” of the GIMnet architecture. The Network layer is responsible for transferring packets from node to node over the GIMnet.

Nodes are connected to the GIMnet by connecting to one of the hubs in the system. New nodes can be connected and disconnected from the system at any time. Each time a new node connects to the system it is assigned a unique identifier which consists of the Hub Identifier and Node identifier inside the given hub. Multiple hubs can be connected together to form a single larger GIMnet. Hubs will attempt to maintain the routing topology by auto-reconnecting as existing connections fail. If a hub is completely disconnected from the GIMnet, it will continue to operate with its local nodes normally, and only transmissions to non-local nodes will fail.

Each node connected to a hub can directly communicate with any other node in the whole GIMnet, therefore GIMnet offers virtual peer-to-peer communication between nodes. The hub uses TCP/IP for all node and hub connections, and nodes create a TCP connection towards the hub. Since most firewalls permit outgoing traffic, we can have an interconnected system without any extra firewall configuration.

A single hub in the system can have any number of local nodes, and transfers between nodes are handled independently of each other. If the hub is located on the same machine as the nodes, the transfer rates can be considered to be near IPC (Inter Process Communication) rates.

For GIMnet, tcpHub has three different protocols:

- The HubNS protocol provides system wide unique naming which can be used to reach nodes without knowing their physical location on the system (directly comparable to DNS on the Internet). Each name corresponds to a number which is a unique identifier for a node and is used for routing packets between nodes. HubNS also provides facilities for a node to reconnect using the same identifier number each time (bypassing the dynamic identifier assignment). This helps to keep node to node connections intact and minimizes the need for name cache updates on nodes.

- FastProtocol is the real payload carrier protocol. Data is transferred in frames, providing a variable size packet format over a reliable stream protocol (TCP). FastProtocol’s routing is implemented using mostly look-up tables. FastProtocol also implements multicast efficiently reducing the sender’s bandwidth usage. If multiple recipients reside on another hub, the packet is sent only once over a hub-to-hub connection. The FastProtocol header is kept to a minimum (16 bytes) allowing small packets (like direct teleoperation data) to be transferred without excessive overhead.

- The HubCtrl protocol provides some special features for configuring and managing the tcpHub. This includes for example routing, node identifier assignment/reassignment, and instrumentation. This protocol is intended mostly for configuring the hub itself and has little use on the application layer.

### 3.3 NIF

NIF is a light-weight, low-level Network Interface library written in C. It wraps the OS-specific socket code (Unix/Linux and Windows) and the bit-level protocol used by tcpHub into an OS-independent API, presenting all the basic operations like connecting to the hub, sending and receiving data packets, and querying peer IDs by name. It is thread safe, and designed for high throughput and low latency. It is not used directly by GIMnet applications; GIMI provides higher-level functionality and is built on top of NIF.

### 3.4 GIMI

GIMI, or GIM Interface, is the application-layer communication API for GIMnet. It provides a simple C++ interface for basic operations like joining the network, sending and receiving data to and from other clients, as well as for advanced operations like subscribing data from other clients and searching for clients that provide data. GIMI encapsulates the low-level functionality of GIMnet into a simple function interface. As the users of GIMI have various programming abilities, it is also designed as user tolerant. For example, GIMI is fully thread safe. Another important aspect is that GIMI is not platform dependent.

GIMI features synchronized and unsynchronized data sending. Synchronized sending is used when the data must be delivered in real-time. The synchronization causes some
loss in maximum throughput, but it prevents the data from being buffered in various TCP stacks. The sent data is limited to a given number of unreplied data packets, which enables the server to automatically send data at different rates to clients with different bandwidth capabilities.

GIMI has a service-based messaging system. A GIMnet application specifies its public interface as one or several services. A service may provide a specific data type, or accept a certain data type as input (e.g., control commands). Data types are mostly application specific and are created to suit the application's needs. In the INTEGRATOR project the services are related to mobile machines and teleoperation, including position, tele-control, image streams, etc.

GIMI implements an automatic service discovery that allows global querying of available services. The service discovery reveals all the registered services in the GIMnet and the name and location of the module that provides the services. Any client in the GIMnet can subscribe to any provided data service. The data is then sent as a multicast packet in the network. The service system removes the need for polling, as the newest data is automatically sent to all subscribers.

The handling of incoming messages is based on the service id’s as well. Each received message is stored in a queue according to its type id and instance id until the application fetches it from the queue. The size of the queues can be freely set by the user. In case the queue is full it functions as a ring buffer. The queue system provides efficient fetching of specified types of messages. Moreover, for real-time purposes a ring buffer of size one always makes sure that the user has the most recent data available when requested.

GIMI uses the Bitestream library (Binary Transport Encoding) for encapsulating the data. Bitestream is a small C library that is used to collect labeled pieces of binary data into a single binary data packet that can be easily transmitted over a network. It is used to construct a FIFO stream of key-value pairs in memory prior to transmission. The keys, or labels, can be 24-bit integers, or text up to 255 bytes long, and associated with each label is binary data from 1 byte up to 1 gigabyte in length. Bitestream is designed to operate efficiently and to add very little overhead to the actual data.

4 Tests and Results

The tests presented here were made to prove the functionality of the overall system. Real-time control requires a consistently fast and predictable response. Latency tests were made to measure the round-trip time of a packet on the application level. The bandwidth tests were made to find the limits of the system.

Latency tests were conducted by sending small ping packets and reply packets using the GIMI API. Pings were sent at 100 ms intervals between programs running on the same computer as the tcpHub (figure 3), as well as over a 100 Mbit LAN (figure 4). As can be seen from the graphs, most local ping times remain well under 1 ms, and most remote ping times fall within the range 1-2 ms, with occasional greater latencies.

Bandwidth tests were similar to the latency tests, but with extra data as payload. All data packets were replied to with a small packet. In the "sync 1" mode in figure 5 the test program did not send another packet until the previous one was replied to, while in the "sync 5" mode it only stopped to wait whenever there were five unreplied packets. As the graphs show, the bandwidth in packets per second (figure 5) only drops when the packet size is very large, and the bandwidth in bytes per second (figure 6) can reach significantly high values.
4.1 Demonstrational tests

The first demonstration of the GIM testbed has been a multi-machine teleoperation scenario. The demonstration included three types of entities: a locally operated robot, teleoperated robots, and a teleoperated simulator. Figure 7 is a screen capture from the teleoperator's screen. The picture is from an overview camera in the test hall. The simulator (top right) is augmented to the real image. The simulator is driven inside a 3D model of the test hall.

The implementation of the demonstration is illustrated in figure 8. The setup consists of one tcpHub situated in each laboratory. The applications connect to their local hubs and the hubs connect together to form the network. In this demonstration the control packets are sent to the robots over radio modem. The simulator has the same interface as the real machines.

Three programs related to teleoperation are connected to the TTY tcpHub. The connection handlers connect the robots to the GIMnet. The camera server provides images from the testing hall. In the TKK remote control room, two teleoperation applications are running on different computers, displaying the image streams and the model of the worksite. Additionally, a robot in the remote control room is teleoperated from the worksite at TTY.

Figure 9 shows the remote control station with the supervisory robot. Figure 10 shows the robot control modules used in the teleoperated robot. The robot has an internal tcpHub for inter-process communication. The actual robot control module is called ASRobo. ASRobo is an application that allows controlling robots with diverse hardware through a coherent interface; in essence, a hardware abstraction layer. ASRobo has features similar to Player [5], but in addition it is specifically designed to work with GIMnet. Also, ASRobo can be simultaneously controlled by multiple clients. For example, one client controlling the position (moving around) and another one controlling the PTU (Pan/Tilt Unit).

The setup in figure 10 shows two additional modules. This shows the power of GIMnet in rapid prototyping. A stereo camera unit was added on top of the robot, but the camera software was not integrated to ASRobo. However,
the software modules were written to use GiMI. Thus the modules related to the camera unit are running as separate processes, but the outside user does not notice that.

The purpose of the demonstration was to demonstrate the functionality of the system. The system has different types of machines that are operated from various locations. The data was transferred both ways and the robots were controlled in real time.

5 Conclusion and Future work

This paper has described a software infrastructure for generic intelligent machine control. The major design criteria, detailed modules and principle of operation were introduced. The results show that the infrastructure provides reliable communication over the Internet. The infrastructure itself is fast and reliable. The only unpredictable part is the network delay.

The system has been initially demonstrated in a direct teleoperation task. The teleoperated machines were approximately 200 km away from the operating station. The suitability of the infrastructure for robot control was demonstrated.

In the future GiMNet will serve as an infrastructure for implementing research results made in GiM. The future research plan of the GiM is massive coverage of all the aspects of the future worksite. This includes the development of the mobile work machines to be digitally fully controllable, having low level adaptation and autonomy as well as higher level autonomy. The machines will also serve as a testbed for various novel sensors such as 3D laser ranging, Pseudolites, etc.

The control architecture for the mobile work machines is in special scope throughout the project. The future worksite concept requires the use of general tools for different types of machines. Thus, self-configuration of the systems is required. On the user interface side the challenges are how to define the work tasks for the multiple different types of machines, how to control the task execution, and how the state of the task is presented. The goal is to be able to model the state of tasks in real time using the work site sensors and on-board sensors.

Acknowledgment

This work has been supported by Finnish funding agency for technology and innovation Tekes. Authors are grateful for the support.

References


