



IP-FP6-015964

AEOLUS

Algorithmic Principles for Building Efficient Overlay Computers

Deliverable D6.0.5

Scientific Highlights and Recommendations

Responsible Partner: Radiolabs (I)
Report Preparation Date: February 2010
Contract Start Date: 01/09/05 Duration: 54 months
Project Co-ordinator: University of Patras (EL)

Table of contents

1. The AEOLUS testbed core.....	5
2. JXTACh: improving JXTA.....	7
3. P2P storage systems	8
4. Publish/subscribe functionalities.....	10
5. Arigatoni.....	10
6. The importance of performance estimation in overlay computers.....	11
7. Replication services for heterogeneity and dynamicity	13
8. Scheduling services for OCP applications	14
9. A toolset of security primitives	16
10. SybilGuard	18
11. A cryptographic protocol analyzer	19
12. TRUST-X and trust negotiation	20
13. Functionalities for extending the OCP to wireless users.....	21
14. AEOFORGE.....	24
References	26

This deliverable summarizes highlighted results from the implementation work during AEOLUS in the context of subproject SP6. This includes the core of the AEOLUS testbed itself, AEOFORGE, and functionalities provided from each thematic subproject: JXTACH, peer-to-peer storage systems, publish/subscribe functionalities, and Arigatoni from SP2, the microbenchmarking suite, the replication services, and the schedulers for OCP application from SP3, a toolset for basic security primitives as well as additional functionalities such as SybilGuard, Trust-X, and the Cryptographic protocol analyzer from SP4, and highlighted functionalities that facilitate the extension of the Overlay Computing Platform and the testbed to wireless users from SP5.

These are presented in the following. Each chapter is devoted to a functionality or a set of functionalities; it gives a brief description of them and presents the rationale behind them and provides (technical) recommendations as lesson learnt either during their development or during their deployment and testing.

1. The AEOLUS testbed core

The AEOLUS testbed is a distributed system of large scale that has supported the development, execution, and testing of functionalities developed within AEOLUS. The current testbed setup consists of more than 100 computers located at many different sites as well as almost 300 wireless devices.

The architecture of the testbed is depicted in the following figure.

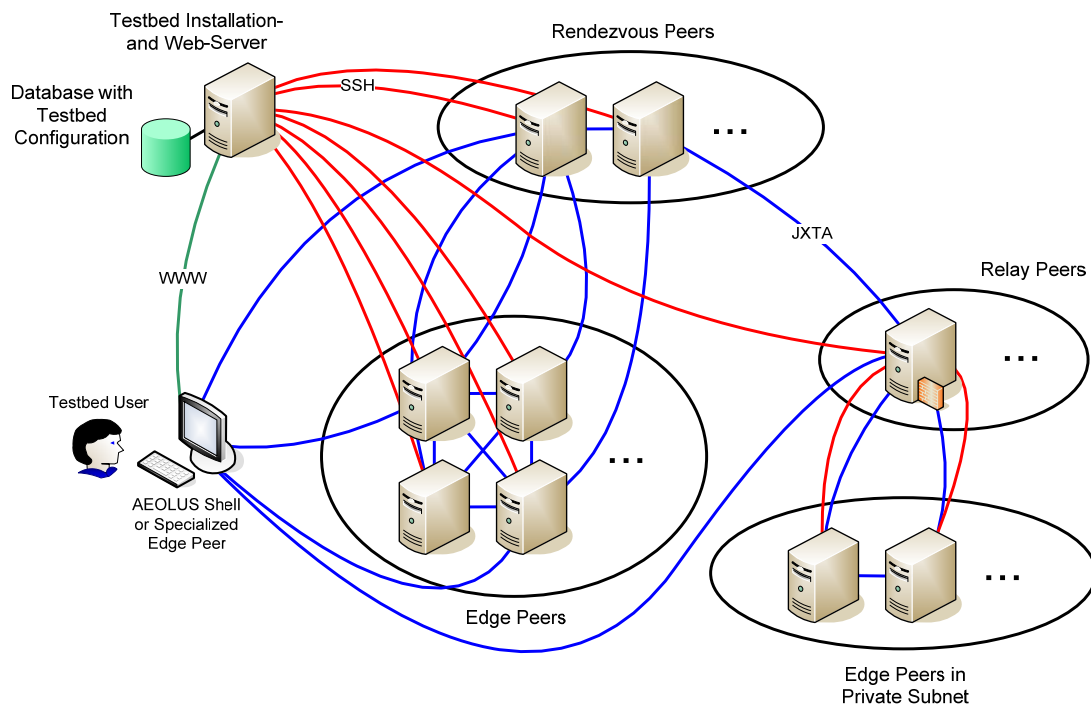


Figure 1: The architecture of the AEOLUS testbed

Its components are:

- the *testbed server* which includes the web server, the management software, and the database with the testbed configuration;

- the *edge peers*: these are essentially the computing nodes donated to the testbed by the several different sites;
- *rendezvous peers* and *relay peers*: these nodes support special functionalities of JXTA, which enable the edge peers to locate each other and communicate. Furthermore, they can assist the incorporation of wireless nodes to the testbed;
- *testbed user nodes*: these are nodes through which the users access the testbed. Besides using the testbed through the web interface, specialized edge peers can be employed as user front-ends or interfaces to other software outside the testbed.

A typical scenario from a testbed user's (i.e., software developer's) point of view would look like this: the user logs in to the web front-end and creates a partition, i.e., a virtual space to hold the data and configuration of her work. She then allocates a number of edge peers and uploads the software (i.e., the JXTA service) to test her testbed partition. A suitable rendezvous peer is allocated automatically by the testbed. The edge peers allocated can be used as physical instances, i.e., one computer to run one testbed node, or in a virtualized way, i.e., one computer simulates up to ten edge peers. In addition to the software, the user may upload configuration files and/or native libraries. All JXTA services to run on the allocated edge peers have to be registered through the web interface. Since not all edge peers in a partition have to fulfill the same task, the user can configure them through a web interface, i.e., she can choose which of the registered JXTA services to run at which nodes, which configuration files to upload to which nodes, and so on. To facilitate the handling of the configuration files, the testbed supports so called scripted constants, i.e., special keywords (e.g., for the IP address), which are replaced with their particular actual values on a per-edge-peer-basis by the testbed software. Once the user has configured everything, she can start, stop, or restart all the nodes of her partition with one click through the testbed's web front-end. All logfiles of all physical and virtual nodes are also accessible with one click through the web interface.

For system administrators the testbed is quite easy to handle and to maintain. To add a computer to the testbed, an administrator needs a testbed login (for security reasons her identity is verified using TrustX). The installation on Windows computers is effortless consisting only of a few steps and being guided by a graphical installer. Once the testbed software is installed, it configures itself automatically. In case of hardware failure (or temporal unavailability), the system administrator will be informed via mail if the problem persists for several hours.

Since the latest release, the testbed software also supports dynamic nodes, i.e., edge peers, which are not necessarily always switched on and may have a dynamic IP address. When joining the testbed for the first time, they are authenticated using TrustX. In case an edge peer is not operational for some time or when its IP address has changed, the peer is re-authenticated using TrustX once more.

Technical recommendations

Since its early deployment, the testbed ran reliably, showing good fault-tolerance and scalability. Since bugs in JXTA and programming errors (such as busy loops) can lead to a very high load in particular nodes, even in the rendezvous nodes, we recommend not to run any node of any partition on the testbed server itself; the latest version of the testbed software thus automatically spreads the rendezvous peers among the set of available computers (which was not the case in former versions in order to keep the rendezvous IP addresses constant).

A new feature worth implementing would be the ability to store the configuration of a partition based on a virtual edge peer layer, which would allow the user to exchange or reassign edge peers without having to reconfigure them. On the one hand, this would facilitate the replacement of failing nodes; on the other hand, this would motivate users to release nodes that are not required in the foreseeable future.

Our multi-level security model, which, in the first place, only accepts connections on the management port from the testbed server and, secondly, only accepts a fixed set of parameter-less commands that trigger an

installation or configuration action, appeared to be a good architectural choice, since we did not suffer a single security issue through the whole lifetime of the testbed.

Also, our two-sided security model appeared to be safe against internal attacks, which can occur in case a user does not protect the secrecy of her password. Although some users view Java security policy rules as a restriction, our configuration appeared to be loose enough in order not to limit the functionalities' features, and at the same sufficiently restrictive so that the data of the computers' owners as well as of other testbed users are safe in case a testbed account would be compromised.

2. JXTACh: improving JXTA

The growth of popularity of P2P networks raised up the need to have the proper instruments to develop services and application over this kind of structure. Sun Microsystems tried to provide users with such an instrument beginning project JXTA in 2001. The aim was to provide a general framework for developing P2P applications. Within AEOLUS, UDRTV has developed a *pure* DHT based version of JXTA framework which we call JXTACh [NCL09]. We say pure because in JXTA there is a mechanism resembling a DHT [TAA+03], but its characteristics are not respecting the properties of DHTs. JXTA developers explain the use of such a mechanism with the expensive maintenance costs of DHTs. In our opinion, instead, the framework could benefit from such a structure. That is why we decided to replace the old mechanism with Chord [SMK+01], which is one of the most popular DHT protocols available in the literature. This replacement has been done in a completely transparent way with respect to the JXTA framework. This property is very important from the point of view of the project in order to let the partners work independently from the new version of the framework.

JXTACh defines a new rendezvous network implementing Chord DHT in place of the loosely consistent DHT. This is done by replacing the old three components of the rendezvous service (Rendezvous peer view, Shared Resource Distributed Index and Walker; their description can be found in [TAA+03]) with:

- *Finger table*: it is exactly the Chord finger table applied to the JXTA rendezvous network.
- *Shared Resource Distributed Index*: the concept is the same as the original one, so that we keep the old name; the main difference consists of the interaction with the finger table, which follows the Chord protocol.
- *Chord walker*: it has the same task as the old walker, but this version follows the Chord protocol instead of linearly forwarding queries.

From the edge peer point of view, the situation remains unchanged: in fact, each edge peer still establishes a loose connection with a rendezvous and still has to use it to publish and look for advertisements.

The old rendezvous peer view occasionally exchanged information in order to keep a loose consistence: the finger table replaces these occasional exchanges with the procedures necessary to maintain the consistency of Chord finger tables, namely the *stabilize* and the *fix fingers* procedures [SMK+01].

When a peer publishes an advertisement, it has to contact the rendezvous it is connected to: it then sends to this rendezvous the key that will be used for the publication process and that will be joined together with the peer identifier to build the reference. The rendezvous applies a hash function (SHA-1) to the key: the result will be used to decide which rendezvous will receive the reference, this time according to the Chord protocol.

Technical recommendations

We have tested JXTACh against JXTA in order to check if the performance could benefit from the new implementation. We have chosen to use the following metrics in order to perform our analysis (these

metrics are also proposed in [HDT04] and are, in our opinion, the most relevant ones for a performance comparison of this kind of protocols):

- *Lookup time*: the time required to get the result of an advertisement research operation.
- *Memory load*: the percentage of memory used by a single rendezvous peer.
- *CPU load*: the percentage of CPU time used by a single rendezvous peer.
- *Dropped query percentage*: the percentage of queries lost.

In order to trace and measure the memory and the CPU load, we have used the `top` linux command. Moreover, we have distinguished between a static and a dynamic environment: in the first case, rendezvous peers are stable and do not disconnect from the overlay network, while in the second case rendezvous peers can disconnect making the system unstable and forcing the protocols to react to the situation. In particular, we have analyzed two different ways to introduce dynamism into the network: the first one is to let the peers disconnect in a “gentle” way (that is, by announcing their departure from the network), while the second one is to make the peers fail, so that the protocol cannot predict when a peer will abandon the network (we call that an “abrupt” disconnection). Finally, we have chosen to use the following set of parameters to be varied in order to study the network behavior under different conditions:

- *Query rate*: it shows how much query load the protocol can stand.
- *Presence of negative queries*: it shows if the presence of negative queries implies increasing or decreasing performance.
- *Gentle or abrupt disconnections of rendezvous peers*: it shows how good the protocol is to react to a search miss and how good the system is to solve an “unexpected” situation.

The results obtained from our experiments on JXTACh are very positive and confirmed our assumptions which were raised from theoretical considerations. The old JXTA rendezvous service was in fact based on a linear mechanism, while Chord has been proved to have a logarithmic behavior. The tests show an increase of performance of one order of magnitude [NCL09].

In addition, the results lead us to the conclusion that it is possible to use a pure DHT for the publication/research of advertisements in JXTA: this means that the rendezvous protocol can be implemented with a pure DHT. In fact the main result of our work is the integration of Chord as the JXTA rendezvous protocol: this integration required a deep study of the JXTA project and a hard work of reverse engineering. JXTACh source code is available at <http://www.dsi.unifi.it/~nocentin/jxtach/>

In a local area network, our tests witnessed an improvement on every measure we considered and in each kind of environment, static or dynamic, with “gentle” or “abrupt” disconnections. This means that not only the injection of a pure DHT can be performed in JXTA, but also that this would result in a better behavior from the point of view of the response time, of the memory and CPU usage, and of the reliability.

3. P2P storage systems

Within AEOLUS, CNRS has studied long-term reliable storage systems over peer-to-peer networks, where the goal is to achieve high reliability and fault tolerance at low cost. The key concept of such systems is to encode the user data into a set of redundant fragments and to distribute them among peers.

Several efforts to build such systems have been made in the past, e.g., Intermemory [GV98], Ocean Store [KBC+00], Freenet [CSWH01], CFS [DKK+01], Total Recall [BTC+04]. However, none of these systems is really used in practice or has a limited range of users. Hence, a proof of concept of the feasibility of such systems is still to be presented before they can be massively adopted. Therefore, the

importance of new system designs (as proposed in Deliverable D2.2.4) and of experimentations in order to answer questions about the reliability, durability, availability, and confidentiality of the data is apparent.

In our experiments, we have been using the CNRS nodes of the AEOLUS platform to validate the analytical studies made in Deliverable D2.2.4 and to assess the feasibility of such storage systems. The main goals of the experiments are to observe the distribution of data-block reconstruction time, the overhead of the metadata and control messages, and to measure the number of unrecoverable files due to node failures. The results obtained by these experimentations give important insights on the distribution of the bandwidth usage, which is very difficult to model analytically and by simulations. These observations are then used to establish relations between the MTTF (Mean Time To Failure) of nodes, the average reconstruction time of data, and the number of lost files. The protocol was here to emulate node failures according to a churn model, and to observe the recovery of the lost data. The experiments allow us to validate simulation studies done in [DGMP09, GMP09]. They are still in progress: in the future, we plan to measure the availability of the data by measuring the number of fragments stored at each node during time.

Technical recommendations

These experimentations along with the simulations and analytical results, allow us to derive practical recommendations for system designers and administrators. They are currently tested by a French startup, UBISTORAGE which commercializes a P2P backup system. In the following, we describe our recommendation in three different contexts.

First, in the context of different data placement strategies, we compared a global random strategy against two local strategies [GMP09]. The former achieves better performance, in particular a lower probability to lose data. But this strategy is hard to implement in practice as the number of neighbors that each node has to keep track of is very large. On the other hand, local strategies have numerous practical advantages: fragments of data can be stored locally, thus leading to low control traffic. However, each peer failure becomes a local disaster putting stress on the neighborhood. The variations of the bandwidth consumption are high and the probability to lose data increases. In [GMP09], we propose a method to choose a good neighborhood size that allows having a data loss rate close to the one of the global policy while keeping the practical advantages of the local policies. Moreover, we show that, if the data have to be stored locally, the reconstruction process can be done globally. This allows more efficient sharing of the load of the reconstruction among the peers.

Second, in the context of low bandwidth consumption in P2P storage systems using erasure encoding, where the data-blocks are cut into s fragments and r fragments of redundancy are added in such a way that the block can be recovered from any subset of s fragments among the $s + r$, we have found that the overhead of bandwidth that could be high, since in order to rebuild a block after some fragments were lost, a peer has to retrieve s fragments and then to resend the missing fragments. Therefore, in order to reduce this overhead, we have proposed and analyzed a system in which an original copy of the data-block is kept by one of the peer in addition of the erasure fragments. This allows the system to quickly reconstruct a missing fragment without having to download the whole block. Furthermore, in file systems where users access constantly their data, the access time and the access bandwidth is greatly improved, as again, a small file can be retrieved alone.

Finally, we have shown that, when building a system, it is not sufficient to estimate the average bandwidth consumption when reconstructing the data, since when a node fails or leaves the network, it induces a high peak of bandwidth consumption. These variations can be high, even for large systems. We show that provisioning decisions that do not take these variations into account can lead to disastrous data loss rate [DGMP09].

4. Publish/subscribe functionalities

In the context of the AEOLUS project, UOI has implemented a publish/subscribe functionality for the overlay computing platform enhanced with ranking capabilities. Generally, in publish/subscribe systems, users describe their interests via subscriptions and are notified whenever new interesting events become available. Typically, in such systems, all subscriptions are considered equally important. However, due to the abundance of information, users may receive overwhelming amounts of events. In our work, we propose using a ranking mechanism based on user preferences, so that only top-ranked events are delivered to each user. Since many times top ranked events are similar to each other, we also propose increasing the diversity of delivered events.

Users can provide events to the system (acting as *publishers* of information) and/or enter subscriptions to the system and consume events (acting as *subscribers*). User subscriptions are formed using an intuitive way as sets of constraints on the values of specific attributes. Subscribers can use the `subscribe` functionality to connect to the system and upload their subscriptions. A user that wishes to provide events to the system can employ the `publish` functionality and upload new information. Events are formed as sets of typed attributes.

Subscribers are notified whenever an event that matches their subscriptions is published. Matching events are ranked based on user preferences. They are delivered in a number of different ways, namely a periodic, a sliding-window and a history-based one. Subscribers can also opt to receive diversified results, i.e. results that exhibit high content dissimilarity. A skyline option is also available for numerical events. When used, notifications about top-ranked events that are dominated by other top-ranked ones are suppressed.

Technical Recommendations

Result diversification is a problem that has been shown to be NP-hard. Thus, in our work, we employed a greedy heuristic to locate diverse top-ranked notifications for the users. The heuristic was relatively fast to compute. However, the overhead of result diversification is still higher than the overhead of result ranking based solely on user preferences.

We used a hierarchical topology for the nodes that implement the matching service. Users can connect to any node in the system and inject their publications and/or subscriptions. However, one could consider other topologies as well, especially to increase redundancy. This would make the system more robust to node failures but highly increases the complexity of the maintenance cost.

5. Arigatoni

During AEOLUS, CNRS has designed, has validated through simulation, and has implemented the Arigatoni resource discovery protocols in overlay networks. Such overlays are built over a large number of distributed computational agents, virtually organized in colonies, and ruled by a leader (broker) who is either elected “democratically” or imposed by system administrators. Every agent asks the broker to log in the colony by declaring the resources that can be offered (with variable guarantees). Once logged in, an agent can ask the broker for other resources. Colonies can recursively be considered as “evolved agents” who can log in an outermost colony governed by another super-leader. Communications and routing intra-colonies goes through a broker-to-broker PKI-based negotiation. Every broker routes intra- and inter-service requests by filtering its “resource routing table”, and then forwarding the request first inside its colony, and second outside it, via the proper super-leader (thus applying an “endogenous-first-exogenous-last” strategy). Theoretically, queries are formulae in first-order logic equipped with a small program used to “orchestrate” and “synchronize” atomic formulae (atomic services). When the client agent receives notification of all (or part of) the requested resources, then the real resource exchange is performed

directly by the server(s) agents, without any further mediation of the broker, in a pure peer-to-peer fashion. The proposed overlay promotes an intermittent participation in the colony, since peers can appear, disappear, and organize themselves dynamically. This implies that the routing process may lead to failures, because some agents have quit or are temporarily unavailable, or they were logged out “*manu militari*” by the broker due to their poor performance or greediness.

Technical Recommendations

During AEOLUS, CNRS has extensively studied resource discovery mechanisms in heterogeneous overlay networks. This has led to improvements of the protocol in order to work also in DHT-based overlay networks, leading to the second version called the Arigatoni-Synapse release. More precisely, the Arigatoni protocol was improved in order to perform information retrieval over the interconnection of heterogeneous overlay networks. Applications using on the top the Arigatoni-Synapse see those intra-overlay as an unique inter-overlay network. Scalability was achieved via co-located nodes, i.e. nodes that are part of multiple overlay networks at the same time. Co-located nodes, playing the role of *neural synapses* and connected to several overlay networks, give a larger search area and provide alternative routing. Synapse can either work with “open” overlays adapting their protocol to synapse interconnection requirements, or with “closed” overlays that will not accept any change to their protocol [LTB09, LTV+09]. To keep the protocol as lightweight as possible, we have implemented a standalone proof-of-concept Arigatoni overlay network. The protocol can run on a very small devices such as Internet tablets and very soon also on the iPhone platforms. Although we do not see any potential problems to include the proof-of-concept as a functionality of the AEOLUS testbed, this may reduce the number of potential mobile devices that can use it because of the size of the libraries used in the testbed (e.g. JXTA).

CNRS has also put “on the road” the recent Arigatoni-Synapse theoretical protocol and routing algorithms with CarPal [CLV10], a proof-of-concept for a mobility sharing application that leverages a distributed hash table to allow a community of people to spontaneously share trip information without the costs of a centralized structure. The peer-to-peer architecture allows moreover the deployment on portable devices and opens new scenarios where trips and sharing requests can be updated in real time. Since Arigatoni-Synapse release allows to interconnect different overlays/communities, the success rate (number of shared rides) can be boosted up thus increasing the effectiveness of our solution.

6. The importance of performance estimation in overlay computers

The issue of performance assessment is a crucial one when setting up and operating a peer-to-peer Overlay Computer (OC). In fact, for a user to be willing to join the OC and hence devote part of its computing capabilities and communication bandwidth to the communal platform, the OC must hold promise of higher performance and capabilities than the user’s individual computing equipment, at least for a subset of the computational tasks that the user is interested in executing. It then follows that the provision of tools that may be used for providing quantitative estimates of the capabilities of the OC is tantamount to its same existence. Also, such tools can be employed at the system level to take performance-conscious decisions, e.g., the balanced dispatch of tasks and other load-levelling activities.

Performance measurement tools must rely upon a methodology aimed at providing a reasonably accurate yet non-intrusive and lightweight estimate of the key factors affecting performance of a distributed application, such as interconnection *latency* and *bandwidth*, and available *computing power* of the peers. The approach to measurement pursued within AEOLUS has been the use of microbenchmarking techniques. The UNIPD site deployed a suite of microbenchmarks, fully integrated in the AEOLUS testbed, which can be employed to identify the main characteristics that impact performance in a P2P system based on JXTA. The tests have been implemented using the JXTA *socket* interface, which

provides reliable bi-directional communications. Lower level interfaces (e.g., Pipes), although faster, have not been used because of their unreliability. Our objective has been to measure key performance quantities at the user level, such as latency, bandwidth, and computing power.

Technical recommendations

One of the lessons learnt during the development of the suite is that the JXTA API introduces a considerable amount of software overhead which often has detrimental effects on performance, especially for communication-intensive distributed applications. To exemplify this phenomenon, we report the following very basic yet highly explanatory example. To measure point-to-point latency, the core of the microbenchmark is a simple iterated *ping-test*, where a peer sends a small packet (8 bytes) to a selected peer, which then replies as soon as it receives the message. The initiating peer then computes the round-trip time by averaging over the iterations. In a series of experiments conducted among peers residing on machines of different capabilities, we were able to assess that this JXTA-level ping is slower than the canonical ICMP one *up to three orders of magnitude*, because of the software overheads introduced by JXTA. One of the consequences is that ping times are not only influenced by the communication infrastructure connecting the peers, but also by the *computational capabilities of the machines on which these peers are executed*. This is a counterintuitive phenomenon which shows how intricate the interplay between computing power and communication bandwidth is in OCs.

Another lesson learnt relates to the subtleties of measuring communication time and bandwidth. In fact, since JXTA receives are posted before the actual data arrive, receiving times must be measured from the reception of the first byte till the end of communication, which are clearly identified instants. Since the IP protocol stack in fact does not allow the buffering of large amounts of data before actual transmission, a JXTA send can be considered to be *blocking*. Hence, sending times must be measured by timing the beginning and the end of the application-level send operations. Communication time can then be defined as the time a peer spends executing sends and/or receives, and bandwidth can be computed as the sum of the outgoing and incoming bytes divided by the measured communication time. This bandwidth measure captures both the capability of the peer, the state of congestion of the network, and provides a uniform and clear way to compare performance of different communication patterns.

A basic bandwidth measurement performed by one of our microbenchmarks is *point-to-point* bandwidth and is implemented by letting one peer send a given amount of data to another peer and wait for an acknowledgement from the receiver. Several message sizes have been employed to identify the point where the bandwidth saturates. Again, results showed a high dependency on the underlying architecture: when employing fast computers, the bandwidth is quite close to the maximum available bandwidth offered by the physical communication medium, while slow computers dramatically hamper communication speed. We also noted that fast-to-slow communication is consistently worse than slow-to-fast communication, and indeed the former is even worse than slow-to-slow communication. This observations confirm once again the high intrusiveness of the JXTA software layer, where certain pairings between processors of different computational power induce definite performance penalties.

We also measured the execution times of *gather* (i.e., all-to-one) and *scatter* (i.e., one-to-all) communication patterns, to evaluate the OC communication capabilities with respect to typical patterns arising in distributed applications. To measure the execution time of the gather pattern, the alignment algorithm is used to make all peers send the data to the *collector* roughly at the same time. The collector uses a number of receiving threads equal to the number of peers sending data to it. Similar considerations are valid for scatter: a *distributor* sends data to a number of involved peers. The distributor employs a number of threads equal to the number of receiving peers. All the bandwidth measurements were carried out with different configurations involving fast and slow machines as collectors/distributors to provide insights over the software overheads. Results of these experiments point out that the performance of these patterns is characterized by a steep degradation, occurring when exchanging an amount of information that saturates the system. Again, it was observed that saturation point often occurs at much lower volumes of

those that could be theoretically supported by the physical medium under a low-level protocol, thereby confirming once again the high overhead incurred in the use of the JXTA API.

Finally, a global measure of bandwidth was provided by running an *all-to-all* communication pattern, where every peer sends the same amount of data to all other peers. We want to remark that applications heavily employing this pattern are likely to prove not suitable for execution on an OC, since the availability of very large network bandwidths would be required for the OC execution to be competitive with a local one.

To measure the computing power of a remote host in an uncooperative P2P setting we used a quiz-like approach, where the remote peer is required to perform a given computation and the execution time is measured by the inquiring peer. In order to be effective, the computation must transform a small input into a small output through a given amount of computation that cannot be avoided. This ensures that the network does not become a bottleneck, and that the returned measure can be trusted, since the inquired peer cannot employ faster algorithms to provide the right answer. A computation that matches these requirements is given by a random number generator where the input is the seed and the output is the number generated after either a given number of iterations or a given amount of time. The experiments indicated that such a test provides very reliable estimates if a sufficiently large number of iterations are executed by the remote peer to deal with the clock resolution. Clearly, precision in the measurement comes at the cost of a computational overhead which may become substantial on slow machines.

In conclusion, we developed and deployed a fully integrated suite of microbenchmarking experiments for measuring performance of an OC built over JXTA. Results obtained using the suite on various configurations have shown that different communication patterns exhibit highly different behaviors, suggesting that the efficient execution of an application requires a careful choice of the executing nodes. This suggests that the dispatching of jobs should be extremely performance conscious or otherwise there is a high risk that the revenue of distributing an application for execution on the OC may be minimal, at best. We also remark that measuring systems should provide adequate countermeasures against selfishness and free riders, especially for what concerns estimating the computing power of a given node.

A full report on the design and the capabilities of the suite can be found in [BBPP08].

7. Replication services for heterogeneity and dynamicity

Replication is important whenever data are frequently read and needs to be accessed from many different nodes of a large – unstructured, irregularly structured, or even ad-hoc – network. Replicating data in distributed systems can improve both availability and performance. In unstructured overlay networks, with epidemic messaging for query routing, replicating popular data items is also crucial in order to ensure high probability of finding the data within a bounded search distance from the requestor. In the AEOLUS project, MPII has developed the *P2R2* (peer-to-peer replication with 2-approximation) algorithm for dynamic replication [SNW08]. *P2R2* considers unstructured overlay networks and aims to maximize the probability of successful search by means of bounded random walks. Prior work along these lines, most notably the square-root-replication method by Cohen and Shenker [CS02], has analyzed the optimal degrees of replication for data items with non-uniform but global request rates, but did not address the issue of where replicas should be placed and was very limited in the capabilities for handling heterogeneity and dynamics of network and workload.

A salient property of the *P2R2* algorithm is the way it copes with *heterogeneity* and *dynamics* of realistic networks and workloads. *P2R2* maximizes the probability of successfully finding a data item in a query that randomly probes a bounded neighborhood of the query initiator. *P2R2* continuously adjusts the replication degrees and assignments of replicas to nodes to evolving network and workload properties. A near-optimal assignment is guaranteed for sufficiently long stable periods; load shifts or network changes

automatically lead to re-optimization. Within AEOLUS, we have developed software for the dynamic replication of data in global overlay computers. The P2R2 method itself is part of Deliverables D3.1.3 and D3.1.4. The software of Deliverable D6.2.9 had been based on a stand-alone implementation in C++ and was geared for a network simulation. The new Deliverable D6.2.12 provides a complete re-implementation of the P2R2 method. The software is written in Java and uses the JXTA-based interfaces of the AEOLUS testbed, on which the software has been tested. Simulations with realistic network configurations and workloads and measurements in the real network of the AEOLUS testbed have shown that the convergence rate is fast and that P2R2 outperforms prior methods by a large margin.

Technical Recommendations

Looking beyond the AEOLUS framework, we believe that availability and quality-of-service guarantees are a key issue for data-intensive networked applications at global-computing scale. To this end, replication is beneficial whenever data is read-mostly and needs to be accessed from many different nodes of a large – unstructured, irregularly structured, or even ad-hoc – network. With powerful mobile devices such as iPhones and the popularity of multimedia content (music, videos, etc.), replication will become a key technology for global computing. Our results on the P2R2 algorithm for dynamic replication is a particularly promising contribution for global overlay networks because of its robustness to high dynamics of data, load, and network characteristics.

A specific application that could be considered for exploitation beyond AEOLUS is replication for search engines. Web-scale search engines need multiple data centers, or could alternatively consider a large number of storage-and-compute nodes in a global overlay, with each data center or node serving queries for a certain geographic region of users. Each node can be optimized towards the typical queries of its local users. This could be in the form of query-result or index-list caching, adjusted for the expected local workload. For enhanced availability, some index data and precomputed query results are replicated across some non-local data centers. These nodes could be contacted in the occasional case when the local data center becomes temporarily overloaded because of load bursts. This kind of dynamic load distribution, based on replication, needs to consider the delays (network latencies and communication overhead) for forwarding queries to non-local data centers, the storage capacities of the data centers, and the geographical distribution of queries.

8. Scheduling services for OCP applications

Within AEOLUS, we have implemented a parametric scheduler to support applications that run on top of the Overlay Computing Platform. The scheduler aims to isolate the task scheduling functionalities from applications that need to perform such decisions. Instead, the scheduler is a dedicated server (to be thought of as a service running at some nodes of the testbed) which is allocated to applications in order to assist them by monitoring the execution of tasks on the available computing nodes and recommend scheduling decisions to them.

The underlying optimization problem to be solved can be abstractly defined as follows. There are a number of testbed nodes that provide the computational resources for the execution of the tasks; using the standard scheduling terminology, we refer to these nodes as the *machines*. Each task has particular load characteristics; the aim of the scheduler is to allocate tasks to the machines so that an objective that is desired by the application owner is optimized. The scheduler supports several different algorithms. It may run online algorithms that satisfy scheduling requests on demand or offline ones that collect tasks and schedule them in a batch way. The scheduler supports different algorithms for makespan minimization, minimization of the L_p norm of the machine loads, minimization of average completion times, etc.

It supports commands for the insertion/deletion of a machine, for the initiation and termination of a task or its migration from a machine to another, for querying the scheduling status, and for providing scheduling

reports. These commands are issued by either the application owners, or the nodes controlling the tasks, or the machines. The scheduler also has two different modes of operation. In the basic mode, it assumes that the nodes controlling the tasks truthfully report load requirements when they communicate with the scheduler and bases the scheduling decisions on this information only. This part of the protocol is extremely light and induces almost no overhead to the overall computation. The second mode has additional features for the monitoring of the execution of tasks on the available nodes and bases its scheduling decisions on the load as it is reported by the machines in which the tasks are assigned. The degree of monitoring that can be used depends on the requirements of the application. Monitoring is performed by special messages exchanged between the machines and the scheduler and this may introduce some overhead to the computation time.

The scheduler has been tested either using a user-friendly graphical environment that simulates different scheduler clients (e.g., the application, different machines or tasks) or by automated experiments that have allowed for large-scale experiments on the testbed. Currently, the scheduler is used by the AEOFORGE application in order to balance the load among the edge peers AEOFORGE uses.

Technical recommendation

Clearly, outsourcing the scheduling functionalities from applications can clearly decrease the complexity in the development of applications. Furthermore, in this way, the application designer has access to a menu of different scheduling algorithms that may fit her needs (depending on the particular performance objective that has to be optimized). So, our experience with the schedulers in AEOLUS suggests that such functionalities should be outsourced to several special-purpose nodes. Of course, several different such nodes should exist in order to achieve fault-tolerance.

The operation mode that would better fit a real global computing application is clearly the one that uses monitoring of the machines. In our schedulers, the degree of monitoring that can be used depends on the requirements of the application. In very dynamic situations where tasks or machines come and go very frequently or tasks migrate from one machine to another, there is a trade-off between the accuracy of information the scheduler has (and, consequently, on the quality of the scheduling decisions) and the degree of monitoring the designer of the application requires by the scheduler. A recommendation for application designers here would be to use the scheduling services with the appropriate degree of monitoring depending on their estimates for the dynamicity of the system. In the future, we envision a scheduling service with additional features that automatically adapts (increases or decreases) the degree of monitoring (and, consequently, the quality of information necessary) depending on the dynamicity of the system.

Besides the efficiency of the schedulers in terms of communication messages exchanged, there are obvious issues regarding the efficiency of the schedules themselves. Our most interesting experiments regard the online case of the problem (which is the one that naturally appears in practice) on unrelated machines with the objectives of minimizing the L_2 norm of the machine loads or the makespan [AAF+97, AAG+95]. An interesting result that justifies theoretical investigations within AEOLUS [Car08] is that a greedy algorithm for minimizing the L_p norm of the machine loads also produces very good schedules with respect to the makespan when the parameter p is proportional to the logarithm of the number of available machines.

Concerning offline algorithms, greedy solutions seem to work better in practice compared to well-known approximation algorithms (e.g., [LST90]) that are based on linear programming and rounding techniques. The latter are more computationally demanding and rarely produce significantly better solutions than the greedy ones. The situation becomes even worse in special cases of the problem (e.g., related machines) where polynomial-time approximation schemes are known. Unfortunately, these algorithms are usually impractical to implement in a global setting. Fortunately, greedy solutions provide very efficient schedules in these cases.

9. A toolset of security primitives

Within AEOLUS, UNISA has developed a series of security primitives; some highlighted ones are briefly described in the following.

Certified Information Access. The primitive, introduced in [MRK03], deals with the problem of guaranteeing the consistency of the answer, sent by the database to the user, with the information actually contained in the database. We assume the possibility that a database would be willing to give wrong answers to queries issued by the user. Since the user does not know in advance which is the actual information she will receive from the database, there would be no way to distinguish between a correct answer from a maliciously modified one.

Certified Information Access primitives force the database to publish a snapshot of its current contents, which we refer to as the Public Information, on a trusted entity. After such information is available, the user may issue queries to the database. The answers to each query have to be consistent with the public information published before the query was issued.

The implementation of a certified information access has to provide the users with a database service in which each answer to a query consists of the actual query results and a proof that such information corresponds to the actual content of the database. The verification of the proof can be accomplished by using some public information that the database provided before the query was issued. Such public information should not reveal anything about the actual content of the database.

Secure Timestamp Authority. A Secure Timestamp Authority is responsible to emit the timestamp certificates for each request she receives. The parties involved in the service are the following:

- *Client*: it issues timestamps requests for her documents.
- *Secure Timestamp Authority* or STA: it is the party that produces the timestamp certificates and the information needed for verifying a timestamp certificate.
- *Registry*: it holds information needed to verify the authenticity of a timestamp.
- *Verifier*: upon receiving a document and a timestamp certificate it verifies that the timestamp is authentic by accessing the information held by the Registry.

The timestamping method chosen uses a binary tree structure as described in [HS91] and [HS97]. The root of this tree contains the so called “Round Hash Value”. This method works by rounds. In each round, a binary tree is constructed with the requests filed during it. The rounds have a fixed duration that will be fixed in advance. The timestamp certificate for a document contains all the values necessary to rebuild the corresponding branch of the tree built in the round where the document was accepted.

Secure Function Evaluation. This security primitive is based on the work of [MNPS04] and [BNP08]. The service comprises n parties, who want to engage in multi-party secure function evaluation (SFE) and a mediator who controls the computation protocol flow. Each party must trust the mediator. Prior to executing the SFE protocol, the parties must define and coordinate the function-to-be-evaluated. In order to do that, they use the Secure Function Definition Language (SFDL). The SFDL Specification we have chosen is that of version 2.0 [SFDL]. The SFDL is a high-level programming language, which allows the parties to specify the function-to-be-evaluated in the form of a computer program. The SFDL program is then translated by an SFDL compiler to an SHDL circuit file. In our scenario, the mediator chooses a party among the n parties involved in the computation. This party will be called the Executor. It evaluates the circuits for all the parties but it is not able to extract the clear output of the computation. Instead, the garbled output is sent to the Mediator which can extract the clear output. Because the Executor needs the inputs of all the parties to evaluate the circuit, a public key infrastructure, based on the ElGamal crypto system, is used to hide this information.

Secure Searching on Outsourced Data. Predicate encryption schemes are encryption schemes in which each ciphertext C_i is associated with a binary attribute vector $x = (x_1; : : : ; x_n)$ and keys K are associated with predicates. A key K can decrypt a ciphertext C_i if and only if the attribute vector of the ciphertext satisfies the predicate of the key. Predicate encryption schemes can be used to perform search on encrypted data. Using [IP08], SSOD allows to store data in encrypted form on a remote server in such a way that it is possible to perform search without having to download (and decrypt) the whole data set. The parties involved in the service are the following:

- *SearchableStore*: holds the encrypted data and performs the searches on it based on the query keys generated by the DBOwner.
- *DBOwner*: is the party who wants to outsource the data.
- *Writer*: is the party who wants to store some data into the outsourced repository.

sCAPTCHA: Secure CAPTCHA. A denial of service attack (DOS) is an attack against a network service where an adversary abuses a service in such a way that it becomes unusable for other users. When the attack is driven by a large number of machines distributed over the network, it is called distributed DOS (DDOS).

To combat a DDOS, a web server can introduce a barrier between the service and the users. Only by resolving a puzzle it is possible to gain access to the service. A common used puzzle is the CAPTCHA which is an image with distorted text. The user is asked to extract the distorted text from the image and supply this information in a form. Only if the user has supplied the right text she gains access to the service. This task is very hard for computer programs.

A man in the middle attack can be mounted against the CAPTCHA. Suppose that an attacker wants to break `www.site.com` which is protected with CAPTCHAs. The attackers can build a new site `www.freestuff.com` with lots of free and valuable stuff. Every time a user goes to `www.freestuff.com`, the attacker gets a CAPTCHA from `www.site.com` and submits it to the user. The solution provided by the user can then be used to access `www.site.com`. Secure CAPTCHA provides the following enhances:

- It includes in the CAPTCHA the URL of the page which shows the CAPTCHA. In this case, the user must verify that the URL in the CAPTCHA is the same of that shown in the browser's address bar.
- Assuming that the CAPTCHA is shown on a page accessible only using the https protocol, it signs the CAPTCHA using the key pair coming with the certificate used for the https connection. A Firefox plugin automatically verifies the signature.

Technical recommendations

Our experience with the development of the above functionalities have lead to the following technical recommendations regarding the JXTA communication layer, its latency, and Java serialization.

Communication Layer: The JXTA API is quite often too verbose to express even simple statements like the one used to establish a connection with a remote peer. A simplified communication layer can remove those redundancies and verbirosities. Furthermore, using a communication layer it becomes very simple to switch to other transport technologies without changing the business logic of the application. We have implemented a very simple to use communication layer which hides all the coarse details of JXTA and lets the user to switch to standard sockets without any changes to the business logic of the application. This is very useful for testing purposes.

Latency: Sometimes JXTA has a very long latency for closing remote connections. A separate thread can be asked to close these connections avoiding any latency. This thread can be integrated directly into the communication layer.

Serialization: One of the common ways to send Java objects through the network is by using Java Serialization. It is a standard mechanism provided by the Java Development Kit (JDK) which lets the programmer convert any Java object in a stream of bytes. Unfortunately, the mechanism has two drawbacks. First, the serialization process is a very memory consuming task. Furthermore, it introduces a considerable overhead in terms of extra bytes added to the stream of bytes and used to guide the deserialization process which converts the stream into a Java object. For this reason we have introduced as part of the communication layer a set of classes which tries to fix those drawbacks bypassing the Java Serialization. Using that set of classes the programmer can specify directly how to convert an object into a stream of bytes and vice-versa. This removes the memory overhead and the extra bytes needed for the deserialization process. This new serialization mechanism is transparent to the application. Thus, no changes to the business logic of the application are needed.

Using those recommendations, we have successfully deployed and tested our applications/libraries and have achieved minimum total amount of memory requested and network bandwidth used.

10. SybilGuard

SybilGuard assumes the existence of a *friendship network*. Every user exchanges with his friends a symmetric key. In terms of SybilGuard, friendship means a pair of friends believes each other that they will not start a Sybil attack. The design of SybilGuard depends on the fact that the establishment of friendship relations is particularly hard for the adversary. This assumption can be ensured by real life interactions. Furthermore, every node has a global, permanent and unique node ID. However it is not hard to generate several user IDs for a user.

The prototype has been implemented in Java using JXTA technology. It runs on the AEOLUS testbed environment and offers its service to other JXTA services. However, the prototype is meant to demonstrate the principal mechanisms of a social network-based Sybil protection protocol and, as such, it is very limited in its functionality. In particular, the functionalities that are needed for making friends have not been implemented. Friendship relations are randomly generated and distributed by a super node. Furthermore, it is assumed that communication lines are secure and thus all communication is not encrypted. These all seem to be very limited, but give us already a wide playground to understand this type of Sybil protection protocols. The super node gives us the possibility to inject differently shaped friendship networks, and the network implementation gives us an idea about network traffic and calculation overhead of single nodes. However there is much space for extensions of the prototype in the future.

Technical recommendations

The lessons learnt from these implementations are under which conditions the SybilGuard protocol can protect a grid computing environment from Sybil attacks and it turns out that a native implementation could effectively circumvent attacks; however, the implemented protocol has several privacy issues, since an attacker can learn within a short time the friendship relations of the network. In [PFSG07], we have presented a privacy friendly version of the protocol. The main challenge for the implementation of this new variant of the protocol is the much higher communication load for every node.

Furthermore, there are new protocols that protect from Sybil attacks by exploring the social network graph, namely Sybil limit [YGKX08], which are proven to be much more efficient in theoretical settings. It would be interesting to prove or disprove their practicality and compare their performance with the existing implementation. To the best of our knowledge, privacy analysis and privacy friendly versions of these protocols have not been considered until now.

11. A cryptographic protocol analyzer

A protocol is a convention that enables the connection, communication, and data exchange between several computing entities. A cryptographic protocol is a protocol, which performs some security-related function and applies cryptographic methods. Naturally, the cryptographic protocol is expected to satisfy certain security properties. Showing that the protocol is secure with respect to certain properties is essentially convincing that no attack (known or unknown) breaching these properties exists.

The running of an overlay computer in an efficient way will bring about extensive communication between a large number of host computers that may have their own goals and may want to act in a manner that brings the most utility to them, even if it reduces the global effectiveness of the computation. Thus, one can expect many different cryptographic protocols to find their use in the overlay computer. Verifying the security of these protocols by hand will be cumbersome and error-prone. It is also likely that new protocols are added over the lifetime of the overlay computer. Hence, we need a way to automatically or semi-automatically check that the protocols satisfy their security requirements.

There exist two essentially different models for defining the semantics and security properties of cryptographic protocols, and for showing the non-existence of attacks. One of them – the perfect cryptography or Dolev-Yao model – is suitable for formal arguments and automated or semi-automated analysis. Its downside is the cognitive distance from the “real-world” implementations of protocols, e.g. it makes significant, not always well-justified abstractions of exchanged messages and scheduling. The other – the *computational model* – is close to the real world and has better coverage of different cryptographic primitives, but the semantics of protocols and the definitions of security properties are complex, resulting in very complex and hard to verify security proofs. Still, recent years have seen some progress in “taming” the proofs, hence, within AEOLUS, we have decided to implement a computationally sound analyser for cryptographic proofs that may be used in an overlay computer.

The automatic analyser has been reported in deliverables D4.1.3 and D6.2.7. The analyser has been implemented in OCaml; currently it consists of about 11000 lines of code. The source language of the analyser is a rather standard process calculus, which is then translated to the intermediate representation of the analyser (a dependency graph). The security proof itself is organized as a series of local transformations of that graph, none of which distinguishably changes the semantics of the graph. Many graph transformations have been programmed, some corresponding to the security definitions of cryptographic primitives, the others performing simplifications of the graph. The transformations are iteratively applied to the graph in a “sensible” order, until the graph does not change any more. The final graph, which the adversary cannot distinguish from the initial graph, is analysed in some simple way, e.g. by checking that it contains no references to secret data.

Technical recommendations

For the set of protocols that we have experimented with, our automatic analyser achieves success (termination with a final graph that can be analysed in a simple way) through the application of various heuristics and through the careful choice of the order of the transformations. The heuristics can certainly be developed further in order to be able to handle even more protocols, but this might not be the most fruitful approach. Currently, we are of the opinion that best results can be achieved by combining the automatic simplifications with the directions provided by the protocol researcher. To achieve this combination, work has to be done for figuring out how to present the current protocol (or current step in the sequence of dependency graphs) to the researcher in the most descriptive way, such that he/she could suggest the next steps to be performed. On the other hand, the “mundane” transformations or sequences of them should still be performed automatically.

We strongly believe that sequence-of-games based security analysis of protocols is capable of producing machine-verifiable security proofs. As such, a recommended future work is to integrate the analyser with a proof assistant, e.g. Coq. A security proof produced by the tool would then consist of the following parts:

- The statement and proof of a theorem stating that a graph fragment may be substituted with another that has indistinguishable semantics as well as the statement and proof of transitivity of indistinguishability.
- For each of the defined transformations: the proof that the two graph fragments they specify are indistinguishable.
- For each transformation step in the protocol analysis, referring to a specific transformation: the proof that the first fragment occurs in the graph before the transformation, and that the transformation replaces it with the second fragment.

12. TRUST-X and trust negotiation

Trust-X is a comprehensive framework for trust negotiation. It defines both a negotiation architecture and a language for specifying the policies which protect the exchanged resources. The Trust-X framework provides the capability to perform trust negotiations among different type of entities. More specifically, it allows to perform negotiations between two peers, in a conventional P2P environment, and between a single peer, on one side, and a group of peers, on the other side.

In both negotiating settings, the prototype developed in AEOLUS by UOI/Insubria provides the possibility to recover a negotiation crashed for an external reason, such as the failure of a negotiating party or the temporary unavailability of the network. Also, we extended the recovery support to provide to the negotiating parties the capability to voluntarily suspend an ongoing negotiation in order to retrieve some missing resources required by the counterpart.

Another feature provided by the Trust-X prototype is the possibility to refine the initial request after a negative reply by the counterpart. Such a feature is available only at the beginning of the negotiation; it consents the parties to negotiate for a resource similar to the one initially required. Such a new identified resource will be less expensive in terms of requirements to be satisfied to access it than the original one.

If one of the negotiating parties is a group, the Trust-X prototype provides further capabilities with respect to the different language constructs supported, such as the possibility to define quantified and temporal requests. An example of quantified request is the requirement of at least three credentials attesting a non-disclosure agreement with the John Doe Ltd. in order to access a document containing the product specification. On the other hand, an example of temporal request is the requirement that the non-disclosure agreement must last for at least three months. The Trust-X prototype provides the infrastructure required to verify policies defined in this way in a distributed and collaborative way. It relies upon the collaboration of each peer belonging to the group for the discovery of the broken policy. Such collaboration is enforced by means of a software agent which resides in each peer which joined the group.

Because of the feature described above, it sounds natural to deploy the prototype to manage the joining of peers in groups. This has been done within the well known JXTA framework. In such an environment, the Trust-X prototype has been deployed as a group membership service. Hence, a JXTA peer will be able to create a peer group and to base the membership of other peers also on the resources they offer instead of just their identities. On the other hand, the joining peer will contribute providing to the peer group both resources and policies. They will also collaboratively participate to the verification of the group policies in the other peers of the group.

Being an extension of the underlying layer, the features provided by the Trust-X group membership service are available in each node of the testbed by including the appropriate Java library. Besides that, the Trust-X prototype is used as authentication method for the testbed. It consents mobile devices to access the testbed identifying the authorized devices and configuring the testbed access control infrastructure to accept such incoming devices. To achieve this, a slightly modified version of the original trust negotiation

architecture has been developed; in this version we converted the P2P architecture in a multithread client-server authentication system.

Technical recommendations

A further recommendation may be the evaluation of the Trust-X prototype in different network settings such as bluetooth network. In such a network environment it could be used to control the access of the resources available in a different Personal Area Network (or PAN). Trusting on the features provided by the JXTA network which have been enhanced by the services present in the testbed a global deployment should be quite simple. The developers should pay attention that the size of the messages exchanged during the negotiation does not grow too much in order to avoid misbehavior caused by the limitation of the network layer.

Another possible evolution is the deployment of the prototype in overlay architectures other than JXTA. In such scenarios, the developers may be aware that the prototype has been designed to be neutral with respect to the network layer but the verification mechanisms use the JXTA advertisement document format to identify the resources. Hence, new syntactic constructions for the resource specification may be required. In parallel, there will be the need for new resource verification methods which would exploit the features of the chosen overlay network.

Finally, an open issue is the fact that the policies database is a real RDBMS. It would be interesting to evaluate the possibility to exploit the available testbed nodes as remote repositories. Such exploitation should improve the scalability of the prototype but it would also introduce new issues, such as the reliability of the repositories content and the privacy of the replicated data.

13. Functionalities for extending the OCP to wireless users

As the scope of WSN applications continues to grow, in order to harvest their potential benefits, and ultimately become a part of everyday life, a vision needs to be realized which goes well beyond incremental and disconnected improvements of diverse (and often incompatible) implementations. It is our belief that they will have an important role in what is called global computing. To accomplish this, software for WSN applications must adapt to the new requirements posed. Global Computing is about computational infrastructures available globally, able to provide uniform services with variable guarantees for communication, co-operation and mobility, resource usage, security policies and mechanisms, etc., with particular regard to exploiting their universal scale and the programmability of their services. The seeming integration of the Internet and sensor domains and its benefits can be better described e.g., by the concept of a Sensor-oogle or a Google Earth Sense. A whole new layer of information can be used for offering new services to existing or new systems. Imagine a version of Google offering search capabilities for sensing services or a Google Earth with live sensing data coverage per square mile. The functionalities developed within AEOLUS can significantly contribute towards this direction. Some concrete examples follow.

Buffering services. Wireless networks often face network disconnections, due to several reasons (poor quality links, environmental noise, etc.). The nodes of the overlay network handle these intermittent network disconnections using some kind of buffering service. This buffering scheme applies to all data, i.e., both data and control messages. Also, this buffering applies to data that needs to be forwarded to another node of the overlay network as well. For example, assume that a node of the network wishes to get some information from another node in the network that collects information from a wireless sensor network. At some point after they start communicating, the connection between these two nodes is temporarily disrupted. The second node will hold its information and wait until the network connection is restored, then it will start transmitting again.

Gateway for environmental monitoring & actuator control. Wireless sensor and actuator networks can be deployed for local monitoring and control of micromechanical devices. Scattered sensor devices have the capability to collect data. This high-level service provides access to such wireless networks and acts as a controlling center where data are routed to. The service is responsible for the gathering of all the readings coming from the sensor networks and the forwarding of the queries from the data tier to the devices. In other words, they act as gateways between the wireless sensor networks and the fixed part of the global computer.

The queries on sensor networks may aggregate data over a group of sensors or a time window, contain conditions restricting the set of sensors from contributing data, correlate data from different sensors and trigger activation of an actuator. A user interacting with this service will typically issue a sequence of queries in order to obtain all the necessary information. Distributed query execution is optimized for both resource usage and reaction time.

On the hardware level, a typical service provider consists of one wireless device connected to a desktop PC or laptop along with a network connection to the global computer. The wireless device attached to the gateway is necessary to communicate with the wireless network. Alternatively, an embedded platform can be used. Data retrieved from the wireless networks is stored in a relational database fashion, with tables organized in three categories:

1. Device-related tables that are used to store information regarding the technical characteristics of the network nodes. We aim to support heterogeneous networks, i.e., networks that consist of different kinds of devices (e.g., TelosB, MicaZ motes etc.) which in turn may have different kinds of sensors and actuators attached to them (e.g., light, pressure, humidity, temperature etc.).
2. Query-related tables that keep track of active and historic queries. The service offers different type of queries that can be made by the user to the sensor and actuator network, and each query may refer to multiple devices and multiple sensors.
3. Sensor readings and actuator state tables that store the information received from the wireless network. Each record represents a reading coming from a specific device in the network concerning a single sensor or actuator.

SQL-like functionality. From a data storage point of view, one may think of a sensor network as a distributed database that collects physical measurements about the environment, indexes them, and then serves queries from users and other applications external to or from within the network. This service is responsible for the organization and storage of data after sensing actions, how program interfaces to the sensor database look like and how queries are processed and served in an efficient manner. The advantage of the database approach is that it provides a separation between the logical view (naming, access, operation) of the data held by the sensor network and the actual implementation of these operations on the physical network. Diverse sensor network users and applications can focus on the logical structure of the queries they intend to pose and are relatively isolated from the details of physical storage and data networking on the volatile physical infrastructure of the network.

Network statistics & management/control. Monitoring the condition of a WSN based on gross network metrics and also nano-peer local state is crucial for achieving good overall performance (i.e., at the application level) and also necessary to draw conclusions on the current state of the execution of a distributed algorithm. Also, we wish to allow the network administrator to adjust the operation of the network by modifying the parameters of the underlying protocols (e.g., cluster sizes, sleep schedules, encryption level etc.). The service collects information related to the operation of the nano-peers (e.g., information on the topology, available energy, neighborhood sizes, etc.), and allow the developer to program the network's performance to match the application's needs by modifying the operational parameters of the system. Such information can be used to further investigate the performance of the network and possibly draw conclusions on the current state of the execution of a distributed algorithm.

The service allows to control the operation of the network by sending control messages to the wireless nodes. The control may be limited to the energy saving specification of the devices (e.g., the percentage of sleeping period) or extended enough to cover network operation parameters for the size of clusters, the rotation frequency of cluster heads, the time synchronization accuracy, etc. Depending on the level of details provided, the information can be used to execute an offline algorithm (e.g., resource management, scheduling, assignment) in order to fine-tune the performance of the network.

Virtual sensor networks. This service offers the ability to manage multiple WSNs, each with a different control center, under a common installation. This is done by introducing the notion of a virtual sensor network that hides the actual network topology and allows the user to control the nodes as if they were deployed under a single, unified, sensor network. This abstraction significantly reduces the overhead of administering multiple networks. Furthermore, the idea of a unified, virtual sensor network allows the integration of totally heterogeneous sensor networks, i.e., not only regarding different kind of sensors attached to the nodes of the network, but also different kind of CPU architectures that communicate under different RF and optical devices. For each sensor network, a unique ID is given to its respective control center. This sensor network ID helps to distinguish one node from another, when they have the same node ID but belong to different sensor networks, thus making it possible to manage different networks in a unified way.

Technical recommendations

The vision of WSN itself is changing, as applied research in the field progresses, as is evident from the above services offered. This is not irrelevant to the practical difficulties programmers face while developing WSN applications and to the evolution of the sensing devices themselves, which are being integrated more and more in our everyday lives. Having these parameters in mind, there are some issues and requirements that are common in most of the WSN middleware platforms proposed so far, and the things that change are the abstractions provided to the developers, as a result of the different approaches taken. Efficient resource management is the definitive issue in designing software for WSN restrictions in processing capabilities, energy and communication bandwidth, simply put, cannot be easily ignored in WSN. But, when discussing in a global computer context, we should focus on a rather different context of challenges. We briefly discuss a number of the most significant ones.

Ease of application development and deployment. The question here is whether to provide a number of implemented features and functionalities to the final users, through which a certain sensor network application is implemented (i.e., have limited extendability), or to provide a set of tools and APIs to implement such an application and interface it to other environments (application development). To some extent, a complete WSN solution should be able to provide an application development environment, simplifying the development of custom WSN applications, without sacrificing generality of use. This means that network administrators will be able to leverage existing solutions for sensor networks along with creating new ones.

Adaptability & fault tolerance. Most self-organizing features revolve around providing services like time synchronization, etc., but in general network administrators have little control over such features. On the other hand, some applications require the intervention of the network administrator to carry out even simple tasks. There is also the issue of whether the policies providing these dynamic behavior, can be dynamically updated themselves. Fault tolerance in WSN is another concept whose semantics vary greatly depending on the level of the sensor network discussed. There should be a way to learn if a node has been assigned with specific sensing tasks, some sort of a reverse service discovery procedure. There is also the issue of separate sensor hardware failures. This aspect of fault tolerance is closely tied to data aggregation. There is finally the issue of the reliable transmission of data and binary code across the sensor network.

Scalability and heterogeneity. Another important feature is the number of distinct sensor networks which can connect to the global computer and whether these networks comprise different hardware and software subcomponents. The single-network approaches are limited in terms of heterogeneity, partly because they

assume use of specific OS platforms and hardware, although some provide mechanisms for defining e.g., new sensors. Multiple sensor network management, the concept discussed here, essentially translates to the notion of a virtual sensor network that comprises a number of discrete sensor networks that can be managed as a single one and build on these single sensor network environments to support geographically separated networks and connect them with applications providing a number of services. The key idea is to interface sensor networks and end-user applications to publish locally-generated data streams or request streams of interest, without worrying about the underlying sensor infrastructure.

Connectivity issues - Mobility of nodes and gateways. As the vision of future WSNs grows wider to include new applications, mobility is expected to play a crucial role. The architecture and implementation of a WSN platform will be heavily influenced in the case of adopting such concepts. The idea of using mobile devices e.g., mobile phones, to act as sensing devices or intermediates is an example of such a concept. We believe that the use of both mobility cases is justified. We think that the support of both mobile nodes and gateways will be a critical feature in tomorrow's WSN applications. Mobility constraints in present sensor network applications limit their applicability and usability. If WSNs are to play a part in our everyday life, they should somehow deal e.g., with mobile sensor nodes participating to different sensor networks from time to time.

Interfacing to the rest of the world. The first platforms for WSNs provided interfaces to the final user and the rest of the world through custom application interfaces or through a web page. This trend seems to be shifting in order to integrate them with other existing environments. Another approach of providing interfaces is through the use of P2P protocols. Also, there is the approach of sensor network gateways functioning as bridges, mapping internal WSN node IDs to IPv4 and IPv6 addresses. An emerging standard for interconnecting IEEE 802.15.4 and IPv6 networks is 6LoWPAN. We also believe that the use of interfaces like those provided by Google Maps, is an important step toward implementing Internet-scale sensing applications.

Dealing with massive data sets. Future systems that will exploit WSN infrastructures could be powerful distributed search engines that operate without any central database, but instead use geographically-separate networked computers to index documents locally within each node, using whichever processing resources are available. Of course, on this level, what becomes the most important is the data itself from all the available sensor networks. Managing, analyzing and understanding such data, through the use of suitable tools, poses new unique challenges.

Security and trust issues. By security and trust, we refer to encrypted communication and access authorization, respectively. On a sensor node level there is relatively few work implemented in these fields, when referring both to in-sensor-network trust and to inter-WSN application communication. For example, the problem of accepting a newly added sensor node as a trustworthy member of the network is of high importance. Furthermore, it is not clear how a sensor or a user from one WSN or an application client could or should have access to the functionalities and resources of another WSN.

14. AEOFORGE

AEOFORGE is a prototype software whose main goal is to combine scientific contributions of AEOLUS' subprojects in order to demonstrate the feasibility of building a real OCP-based application. An AEOFORGE user can store, edit (in a collaborative environment), search for, compile and run java projects on the AEOLUS testbed. In other words it is a distributed source forge environment.

AEOFORGE has a server and a client side. The server side component of the application is made of a combination of components all running on the AEOLUS testbed:

- *AEOFORGE service*: the main component that receives inputs from the client side and orchestrates calls to the other components via the API they expose
- *STA*: the component that is in charge of timestamping each file stored on the forge

- *FS-Coordinator*: the component that is in charge of managing file storage
- *FS-Storage*: the component, managed by the FS-Coordinator that stores files on the forge
- *SSOD*: the component that allows users to search AEOFORGE for projects
- *Scheduler*: the component that suggest AEOFORGE service where (i.e. on what partition node) a compilation or execution task must be performed
- *Compiler*: the component (i.e., a partition node) where a java source compilation can be performed
- *Executor*: the component where a java program can run

The client side component of the application is an extension of a JXTA shell, connected to the AEOLUS testbed, carrying on board all the necessary libraries developed by the AEOLUS project. The AEOFORGE commands are simple and most of them recall those of a versioning system; the most relevant are:

- *upload*: a zip file is uploaded to the forge; its content is unzipped, all files are time-stamped and stored, a project is created
- *download*: a zip file with all sources and executables is downloaded
- *add*: a file is added to an existing project
- *del*: a file is deleted from a project
- *co* (check-out): a single file is downloaded and it can be edited
- *ci* (check-in): a single file, modified, is uploaded back to the forge
- *find*: a list of projects stored on the forge is prompted to the user
- *compile*: a project is compiled on one of the dedicated testbed nodes and the related jar file is downloaded
- *execute*: a project is executed on one of the dedicated testbed nodes.

Technical recommendations

The effort to build a JXTA application to run in a real context shall address, first of all, latency issues. We faced several latencies during code development and while for most of them we could provide ways to avoid or, at least, reduce them, for some others we can only recommend increasing the provided resources (i.e., bandwidth, CPU, etc). In particular, we have experienced latencies on remote services connection opening; these latencies occur when connections among services are opened for the first time. This is a common JXTA issue and can be tolerated as they occur only once at startup. We have also observed high latencies on `JxtaSocket` closure and on closure of I/O channels built on `JxtaSocket`; this is a common JXTA issue. A solution is provided below.

Fine grained code tuning was carried out to achieve robustness and efficiency in AEOFORGE V2:

- We have delegated closure of i/o channels built on `JxtaSocket` to a separate and dedicated thread in order to avoid delays in the execution of the operation that follow such closures.
- We suggest to use i/o buffered streams to increase i/o performances: e.g. `new DataInputStream(new BufferedInputStream(socket.getInputStream()))`.
- When receiving a large amount of data, it is very likely that a small amount of bytes is read calling `int read(byte[] b)` on an `InputStream`. We suggest using `DataStream (Input and Output)` to

send/receive the amount of data transmitted and after the content of the stream. In this way the receiver can loop on the read call until all bytes are read or the sender closes the connection.

- delivery of messages exceeding 64KB, exchanged through `JxtaBiDiPipe`, is not assured. No feedback on message delivery (whether it has been delivered or not) is returned to the sending node so that it can hang waiting an acknowledgement.

Furthermore, we have observed better performance on startup, search and communication among services using JXTA 2.5 instead of 2.4.1. We have also observed communication issues on edge peers running on JXTA 2.4.1 on nodes with no public IP (the edge peer fails to contact the rendezvous peer). These problems do not appear with JXTA 2.5. Finally, configuration and network startup are different on JXTA 2.5 and 2.4.1 (e.g., `net.jxta.platform.NetworkManager` is not present in 2.4.1); yet 2.5 is compatible with 2.4.1.

References

- [AAF+97] J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3): 486-504, 1997.
- [AAG+95] B. Awerbuch, Y. Azar, E. F. Grove, M.-Y. Kao, P. Krishnan, J. S. Vitter. Load balancing in the Lp norm. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS '95)*, pp. 383-391, 1995.
- [BNP08] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS '08)*, 2008.
- [BBPP08] P. Bertasi, M. Bianco, A. Pietracaprina, and G. Pucci. Obtaining performance measures through microbenchmarking in a peer-to-peer overlay computer. *International Journal of Computational Intelligence Research*, 4(1): 1-8, 2008.
- [BTC+04] R. Bhagwan, K. Tati, Y. C. Cheng, S. Savage, and G. M. Voelker. Total recall: System support for automated availability management. In *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI '04)*, pp. 337 - 350, 2004.
- [Car08] I. Caragiannis. Better bounds for online load balancing on unrelated machines. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '08)*, pp. 972-981, 2008.
- [CLV09] V. Ciancaglini, L. Liquori, L. Vanni. CarPal: interconnecting overlay networks for a community-driven shared mobility. In *Proceedings of the 5th Symposium on Trustworthy Global Computing (TGC '10)*, 2010, to appear.
- [CS02] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the ACM SIGCOMM Conference*, 2002.
- [CSWH01] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proceedings of the Workshop on Designing Privacy Enhancing Technologies*, LNCS 2009, pp. 46 - 66, 2001.
- [DKK+01] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP '01)*, pp. 202 – 215, 2001.

- [DGMP09] O. Dalle, F. Giroire, J. Monteiro, and S. Perennes. Analysis of failure correlation impact on peer-to-peer storage systems. In *Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P '09)*, pp. 184 - 193, 2009.
- [GMP09] F. Giroire, J. Monteiro, and S. Perennes. P2P storage systems: How much locality can they tolerate? In *Proceedings of the 34th IEEE Conference on Local Computer Networks (LCN '09)*, pp. 320 - 323, 2009.
- [GV98] A. V. Goldberg and P. N. Yianilos. Towards an archival intermemory. In *Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries (ADL '98)*, pp. 147 - 156, 1998.
- [HS91] S. Haber and W.S. Stornetta. How to timestamp a digital document. *Journal of Cryptology*, 3(2):99-112, 1991.
- [HS97] S. Haber and W.S. Stornetta. Secure names for bit-strings. In *Proceedings of the 4th ACM Conference on Computer and Communication Security (CCS '97)*, pp. 28-35, 1997.
- [HDT04] E. Halepovic, R. Deters, and B. Traversat. Performance evaluation of jxta rendezvous. In *Proceedings of CoopIS, DOA, and ODBASE Confederated International Conferences On the Move to Meaningful Internet Systems*, LNCS 3291, pp. 1125–1142, 2004.
- [IP08] V. Iovino and G. Persiano. Hidden-vector encryption with groups of prime order. In *Proceedings of Second International Conference on Pairing-Based Cryptography (Pairing '08)*, pp. 75-88, 2008.
- [KBC+00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, et al. OceanStore: an architecture for global-scale persistent storage. *ACM SIGARCH Computer Architecture News*, 28(5): 190 - 201, 2000.
- [LTV+10] L. Liquori, C. Tedeschi, L. Vanni, F. Bongiovanni, V. Ciancaglini, and B. Marinkovic. Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks. In *Proceedings of Networking*, 2010, to appear.
- [LTB09] L. Liquori, C. Tedeschi, and F. Bongiovanni. Babelchord: a social tower of DHT-based overlay networks. In *Proceedings of the 2009 IEEE Symposium on Computers and Communications (ISCC '09)*, pp. 307 - 312, 2009.
- [LST90] J. K. Lenstra, D. B. Shmoys, E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46: 259-271, 1990.
- [MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - A secure two-party computation system. In *Proceedings of the 13th USENIX Security Symposium*, pp. 287-302, 2004.
- [MSQ99] H. Massias, X. Serret Avila, and J. J. Quisquater. Design of a secure timestamping service with minimal trust requirement. In *Proceedings of the 20th Symposium on Information Theory in the Benelux*, 1999.
- [MRK03] S. Micali, M. O. Rabin, and J. Kilian. Zero-knowledge sets. In *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS '03)*, pp. 80 – 91, 2003.
- [NCL09] C. Nocentini, P. Crescenzi, and L. LANZI. Performance evaluation of a chord-based jxta implementation. In *Proceedings of the 1st International Conference on Advances in P2P Systems (AP2PS '09)*, pp. 7–12. 2009.
- [PFSG07] S. Schiffner, M. Kohlweiss, and B. Preneel. Privacy Friendly SybilGuard, Technical report, 2007.
- [SFDL] Secure Function Definition Language 2.0.
<http://www.cs.huji.ac.il/project/Fairplay/FairplayMP/SFDL2.0.pdf>

- [SNW08] M. Sozio, T. Neumann, G. Weikum: Near-optimal dynamic replication in unstructured peer-to-peer networks. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS '08)*, pp. 281 – 290, 2008.
- [SMK+01] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pp. 149–160, 2001.
- [TAA+03] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Pouyoul, and B. Yeager. Project jxta 2.0 super-peer virtual network, 2003.
<http://research.sun.com/spotlight/misc/jxta.pdf>
- [Y86] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science (FOCS '86)*, pp. 162 – 167, 1986.
- [YGKX08] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A near-optimal social network defense against sybil attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P '08)*, pp. 3-17, 2008.