An Experience Based Prediction Service for Internet Distance Estimation

Masterarbeit
der Philosophisch-Naturwissenschaftlichen
Fakultät
der Universität Bern

vorgelegt von
Benjamin Zahler
2007

Draft

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Contents

Abstract

This document presents an experience based prediction service for Internet distances called XBAC. The service is based on a peer-to-peer network to quickly and accurately provide predictions for Internet distance metrics such as the round-trip-time between two Internet hosts.

To achieve this, XBAC uses four key concepts: Firstly, hosts which are close to each other in the topology induced by the concerned distance metrics are grouped. Then, the active measurements of XBAC are reduced by using data from traffic generated by other applications. The third concept is to perform measurements only to targets for which traffic of other applications has been observed and finally, the predictions are based on a statistical analysis of the data collected to provide more accurate predictions.

XBAC groups its members in a way that all members of a group are close to each other in a network topology. These groups collect measurement data to other Internet hosts and again group those remote hosts which have nearly identical measurements to the local group. This concept helps making XBAC scalable because less data needs to be stored and fewer measurements are necessary. Also, with this concept it is possible to make predictions to hosts which are not part of XBAC.

The peer-to-peer design of XBAC helps the network to meet two important objectives. On one hand, static infrastructure can be limited to only one point for making initial contact. This can be as simple as a publicly known URL. On the other hand, the peer-to-peer design means that the load can be well balanced between the nodes. Although some members need to perform administrative tasks, the load caused by these tasks is very small.

Based on these ideas, we implemented a prototype of XBAC whose architecture and implementation are described in detail in this document. The prototype only predicts round-trip-times, but it can be adapted to predict other distance metrics as well. To test XBAC, we deployed it on Planetlab, a network testbed which allowed us to run XBAC on computers located around the world.

The tests we ran showed that our prototype is functioning as desired and that XBAC is able to make useful predictions. In our tests, the average relative error observed was 5.3% while the average absolute error was smaller than 5 milliseconds with an average round-trip-time of 120.44 milliseconds. We also did identify areas where the system does not scale and caused more load than acceptable on the XBAC members. These issues are not resolved yet, but we propose ideas how they can be addressed and solved.

1 Introduction

1.1 Motivation

In today's Internet, there is no simple way to predict the quality of service between two hosts. If, for example, multiple servers are available for a download, one cannot find out which server is closest in terms of network distance or which server provides the best bandwidth to one's computer without active measurements.

These kinds of predictions would have significant advantages. Firstly, if multiple mirrors or peers in a peer-to-peer system are available to download a file, faster downloads are possible if the best server or peer can be determined. The second advantage is that these predictions could be used to optimize overlay networks. Finally, as a consequence of these two advantages, the overall performance of the Internet could be improved because servers that are close in network topology are used and if the route to a server becomes congested, a different server will be chosen for the next request, reducing the load on the congested route.

Providing such predictions brings one major challenge with it: making measurements between every pair of hosts and storing that data is far from scalable. To solve that issue, systems with two different approaches have been developed: One group of systems establishes virtual coordinates and predicts the distance between two hosts based on their distance in the coordinate space while the other systems group hosts with similar behavior and calculate predictions between these groups.

In this thesis, we will present an experience based prediction service which we call XBAC. It belongs to the second group of systems described above: hosts that are close to each other in network topology form groups in which they exchange data about remote hosts they have had contact with. This reduces the effort in two ways: On one hand, members of the group have access to the data measured by other members while on the other hand, the group only stores information about relevant remote hosts in the sense that at least one of its members has had contact with the remote host. Predictions in XBAC are then calculated by making a statistical analysis of that data. Ideally, XBAC would observe all the necessary measurements from the traffic generated by other applications, however, to have enough data to perform the statistical analysis, a certain amount of active measurements will be necessary as well.

Compared to actively measuring the distance, which to our knowledge is the only method currently available on the Internet, XBAC has the following advantages: In first place, prediction-ns based on statistical data give more information than a single measurement. Furthermore, XBAC increases efficiency by making measurements (both active and passive) from all members of the local group to all members of the remote server's group available for calculating predictions. Finally, if a remote server is not responding or the route is to the server is congested, this might be known to XBAC and therefore the information is available significantly faster than if a test measurement has to be made.

1.2 Basic Concepts

This section briefly outlines the terms used for basic concepts in XBAC. Where applicable, those concepts are elaborated more detailed in Chapter 4.1, "Terminology".

Distance: We define the distance between two Internet hosts as a function of quality of service (QoS) attributes between the two hosts. In general, when we use the term "distance from host A to host B", we refer to the latency or round-trip-time (RTT) measured from host A to host B. Note that with this definition, it is well possible that the distance from A to B is different than the distance from B to A.

Cluster: A Set of Internet hosts which have a similar network distance viewed from a specific host. Members of a cluster are not necessarily aware that they are part of the cluster.

Group: A Set of Internet hosts who are close to each other in network distance. Members of a group are aware of being part of the group.

1.3 Objectives

The main objective of this work is to design the communications architecture for XBAC and to implement a prototype of the system.

The objectives of the desired system are:

- XBAC needs no static infrastructure except for a point of initial contact. This might be as simple as a publicly known URL.
- XBAC makes predictions based not only on simple measurements, but analyzes a series of measurements to give a statistically based prediction.

- XBAC scales for Internet wide applications. This is achieved by grouping hosts into groups and clusters.
- XBAC is only able to make predictions for hosts contacted by a member of the host's local group.
- After an initial contact with a host, XBAC guarantees that predictions are possible to that host.
- XBAC provides a fast answer (< 2 seconds). Note that this answer might be that no prediction is possible.

1.4 Structure of this Document

This document is divided into eight main sections. Chapter 2, „Related Work“, gives an overview of both systems with similar objectives as XBAC as well as of systems that were evaluated to be used by XBAC. Next, Chapter 3 describes the architecture of XBAC while in Chapter 4, we show how this architecture is used to create a peer-to-peer design. Chapter 5 then deals with the implementations of XBAC and in Chapter 6, we describe the tests we performed and discuss the results of these tests. From the observations made before, we propose a number of additional improvements in Chapter 7, “Future Work” and finally draw conclusions in Chapter 8.

2 Related work

2.1 Systems with Similar Objectives

2.1.1 IdMaps

IdMaps [3] is a global architecture for Internet host distance (latency / RTT) estimation. The architecture is divided into two layers: a set of tracers, distributed all over the Internet, is constantly measuring and storing all distance information to each other. On the second layer, every AP (Address-prefix, a consecutive address range of IP-addresses, these host are assumed to be close to each other) knows its nearest tracer, which also stores the distance from itself to the AP.

The distance between host A and host B can then be calculated by taking the distance from host A’s AP (AP1) to its nearest tracer (tracer T1), adding it to the distance from host B’s AP (AP2) to its nearest tracer (tracer T2) and finally adding the distance between tracer T1 and tracer T2.

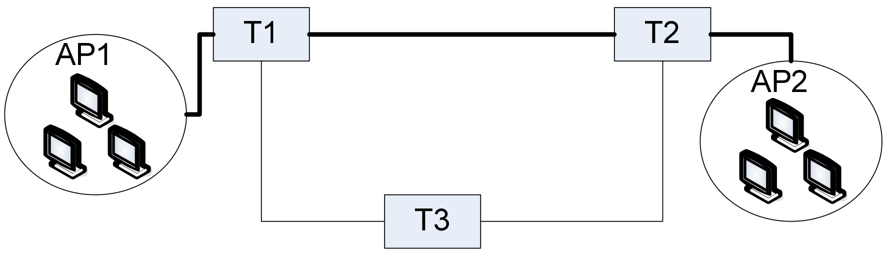

Fig. 1:Example of IdMaps, AP1 and AP2 are Address-prefixes, T1,T2 and T3 are Tracers

This calculation assumes that triangulation on the Internet is possible; the authors show that this is feasible in their case. Another key point is the placement of the tracers; since this decides how accurate the measurements are, the article contains a complex algorithm to solve this problem.

Strengths of IdMaps are its simplicity, its scalability and the fact that mobile nodes do not need any special treatment.

However, the system seems to have significant weaknesses: Firstly, the need for tracers is a problem since ISPs have to be convinced to agree on one single technology and since investments from ISPs are necessary. The second weakness is that hosts in an AP do not always perform similarly. This can happen because consecutive address ranges do not always mean that the nodes are geographically close. The authors propose to solve this problem by querying the ISPs routing table. This problem is especially apparent when a user connects to the Internet through a technology such as VPN, because then, the IP-address does not give any useful information about the location of a host. Also, hosts within an AP can for various reasons (different bandwidth, heavy processor load resulting in slow response times) behave different from other hosts in the AP. If one the three factors just mentioned applies to a host, the concept of APs does not work and predictions for this host will not be accurate.

2.1.2 GNP

GNP [4] is a mechanism to predict Internet network distance (round-trip time) based on virtual coordinates. To achieve this, a small, geographically distributed set of landmark nodes is chosen. The coordinates of these landmark nodes serve as a frame of reference towards which every participating host can compute its own relative coordinates.

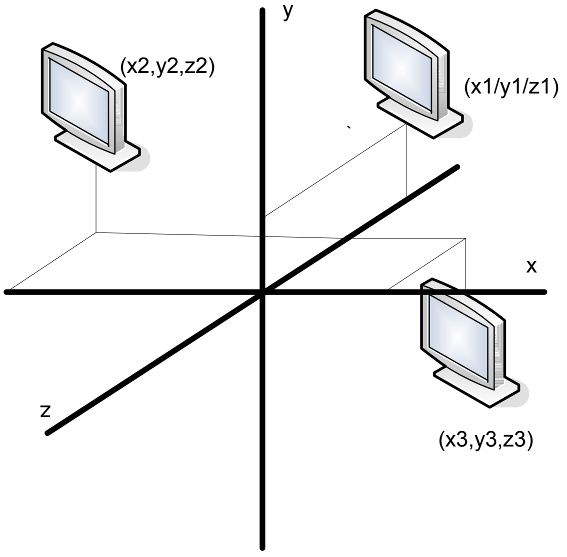

Fig. 2: Virtual coordinate space model of the Internet

In a first step, the landmark nodes measure the round-trip times to each other. One host (possibly a landmark node) collects all these measurements and computes a set of coordinates so that the squared error between virtual and measured distances is minimized. This calculation is performed using the Simplex Downhill method. Note that if the virtual coordinate space has a dimensionality of D, there must be at least D+1 landmark nodes to uniquely compute host coordinates.

Once the coordinates of the Landmarks are known, they are disseminated and ordinary hosts can compute their coordinates relative to the Landmarks. This is done by measuring the distances to all Landmark nodes and then computing coordinates that minimize the squared error. If the coordinates of the target are known, the host can now compute its distance to the target by calculating the Euclidean distance between the two coordinate sets.

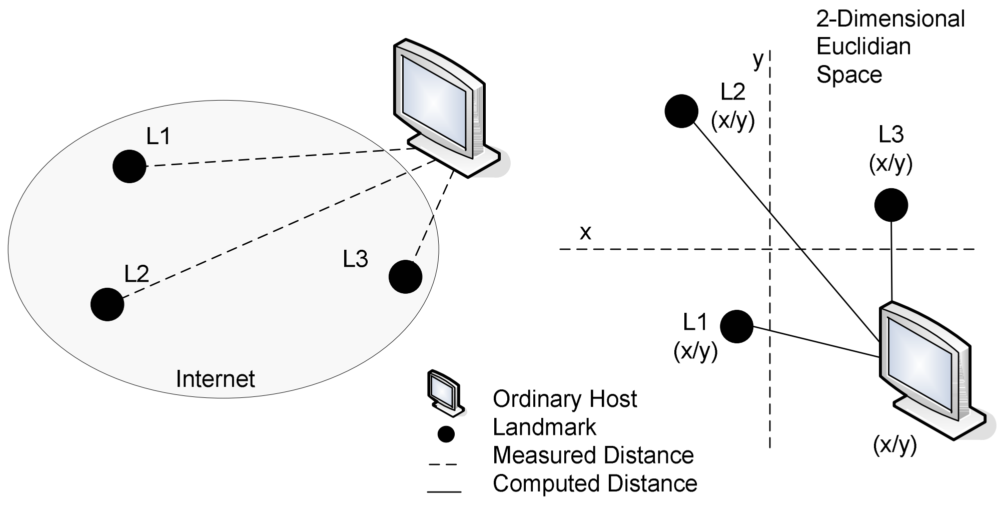

Fig. 3: An ordinary host computes its distance to Landmark nodes

The authors compared their mechanism to IdMaps and to the triangulated heuristic, another algorithm proposed for this problem. In their evaluation, GNP performs better than both of these methods; however, only very small data sets have been used. GNP seems to have a significant advantage in predicting short paths, making it a good choice to solve the problem of nearest server selection.

While GNP is able to predict RTT reliably, its design seems to have two major weaknesses. Firstly, its need of well-known Landmark nodes makes GNP depend on the operators of these nodes. As soon as too many Landmark nodes fail at the same time, hosts cannot compute their own coordinates. While such an event can be considered improbable under normal operations, the system would be very vulnerable to attacks because it relies on a small amount of hosts against which a Denial of Service attack could be launched. The second weakness is that to predict the distance to a host, the remote host's current coordinates must be known. This might work well in a system with a limited number of users, however, for Internet-scaled networks it is impossible to store and maintain all coordinates, meaning that some kind of a selection would need to be implemented.

2.1.3 Vivaldi

Vivaldi [5] is an algorithm to predict communication latency based on synthetic coordinates. The main difference to GNP is that while GNP relies on static landmark nodes, Vivaldi is fully distributed and needs no fixed infrastructure.

The Vivaldi algorithm tries to minimize the squared error of the predictions. This can be compared to a physical mass-spring system in which the energy in the network is minimized according to the laws of physics. For Vivaldi, the rest length of a spring between two nodes can be imagined to be equivalent to the RTT between hosts, thus minimizing the squared error is equivalent to minimizing the energy in the spring network.

At each node, the total force applied by the springs is calculated. To simulate the spring network's evolution, the algorithm considers small intervals of time. At each interval the algorithm moves each node a small distance in the coordinate space in the direction of that force. Then, the forces are recalculated.

An important role in Vivaldi has the parameter called timestep, which decides how far a node moves at every recalculation. If the timestep is high, a fast convergence to the optimal coordinates can be expected; however, if all nodes use high timesteps, the network oscillates and never converges. To address this issue, the authors propose an adaptive timestep based on the estimated accuracy of a node's coordinates. The timestep is implemented as

$$\partial = c \cdot \frac{localError}{localError + remoteError}$$

where c is a defined constant and c<1, for their experiments, the authors used c = 0.25.

The formula to calculate the timestep requires each node to know its estimated error. This is simply done by computing the average of relative errors of recent predictions.

Using simulations, the authors show that Vivaldi can cope with high error nodes, adjust to network changes and that its accuracy is similar to that of GNP even though Vivaldi does not need static landmark nodes.

Even though Vivaldi manages to solve one weakness found in GNP by avoiding Landmark nodes, the second issue mentioned in GNP remains the same: To make a prediction, a host needs to know the remote host's coordinates.

2.2 Peer-to-Peer Routing Protocols

Peer-to-peer (P2P) systems and applications are distributed systems without any centralized control or hierarchical organization, where the software running at each node is equivalent in functionality. One of the major challenges of P2P networks is efficient routing of messages. In this section, a selection of advanced routing algorithms is presented.

2.2.1 Pastry

Pastry [6] is a scalable, distributed object location and routing substrate for wide-area peer-to-peer applications. It provides the following capability: Each node in the Pastry network has a unique numeric identifier (NodeId). When presented with a message and a numeric key, a Pastry node efficiently routes the message to the node with a NodeId that is numerically closest to the key, among all currently live Pastry nodes [6].

The routing algorithm is based on the algorithm of Plaxton et al [11]. Each node is assigned a unique NodeId of k digits and keeps three tables: the Leaf Set, a Routing Table and a neighbor map.

NodeId 10233102			
Leaf set			
	smaller	larger	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232
Routing Table			
0-2212102	1	-2301205	-1203206
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	
Neighbourhood set			
13021022	10200230	11301233	31301233
2212102	22301203	31203203	33213321

Fig. 4: Sample tables of a pastry node

The routing table contains k levels, each one containing nodes whose Id is identical with the node's own Id in the first k digits but differ in digit k+1. For each possibility of digit k+1, the closest node (based on a network proximity metric) is stored. The neighbor map contains nodes which are closest to the node according to the proximity metric while the leaf set contains nodes which are closest in the NodeId space.

Given a message, the node first checks to see if the key falls within the range of NodeIds covered by its leaf set. If so, the message is forwarded directly to the destination node, namely the node in the leaf set whose NodeId is closest to the key (possibly the present node).

If the key is not covered by the leaf set, then the routing table is used and the message is forwarded to a node that shares a common prefix with the key by at least one more digit.

In certain cases, it is possible that the appropriate entry in the routing table is empty or the associated node is not reachable, in which case the message is forwarded to a node that shares a prefix with the key at least as long as the local node, and is numerically closer to the key than the present node's id. Such a node can be found in the node's Leafset.

With this routing algorithm, the expected number of routing steps is O(log N), where N is the number of Pastry nodes in the network. Despite concurrent node failures, eventual delivery is guaranteed unless multiple nodes (the number can be configured, usually 8 or 16) with adjacent NodeIds fail simultaneously.

A very important issue in P2P systems is balance of load. If key Ids are not well balanced, few nodes will have high demands in terms of bandwidth, CPU-load and storage. Pastry leaves creation of the key Ids up to the application layered on top, however, the implementation described below provides methods for creating such Ids using an MD5-Hash, resulting in the desired balance of load.

A free implementation of Pastry programmed in Java is available, the implementation is well-documented and support is provided through a mailing list.

2.2.2 PAST

PAST [7] is a large-scale, Internet-based, global storage utility that provides scalability, high availability, persistence and security. It is completely self-organizing and is based on the routing algorithm of Pastry. In PAST, files are immutable and cannot be deleted. This means that a new version of a file will have a different fileId than the file itself. If a file is not needed anymore, the owner can "reclaim" the storage space, after that, a lookup might still work, but this is not guaranteed by PAST anymore. To provide a reliable lookup, for each document a specified number of replicas is maintained. When a file is inserted in PAST, Pastry routes the file to the k (number of replicas) nodes whose node identifiers are numerically closest to the 128 most significant bits of the file identifier (fileId). Each of these nodes then stores a copy of the file. A lookup request for a file is routed towards the live node with a NodeId that is numerically closest to the requested fileId. PAST has also well developed security mechanisms, but since they are not of importance to XBAC, they will not be discussed here.

2.2.3 Chord

Chord [8] is a service very similar to Pastry: given a key, it maps the key onto a node. However, the two services achieve this with a quite different strategy.

The basic structure of Chord is as follows: Each node and key is assigned an m-bit identifier obtained by hashing the IP address of a node or the key, the hash method used in Chord is SHA1. Then, consistent hashing [12] is used to assign every key to a node. Identifiers are ordered in an identifier circle modulo 2^m. Key k is assigned to the first node whose identifier is equal to or follows the identifier of k in the identifier space. This system is very likely to provide a good load balance. For routing, only a small amount of information is stored. If the identifiers have a length of m bits, each node maintains a routing table (called finger table) with m entries. The i-th entry in the table of node n contains the identity of the node which succeeds n by at least 2^i-1 on the identifier circle. This means, that a node has more information about close nodes than about distant nodes. If it needs to look up a key for which it has no entry, it asks the node in its finger table whose identifier immediately precedes the identifier of the key. This node is closer to the target node and thus knows more nodes close to the target, so by repeating this process n learns about nodes closer and closer to the target until it finally finds it. The number of nodes that must be contacted in this network to find a desired key is O(logN).

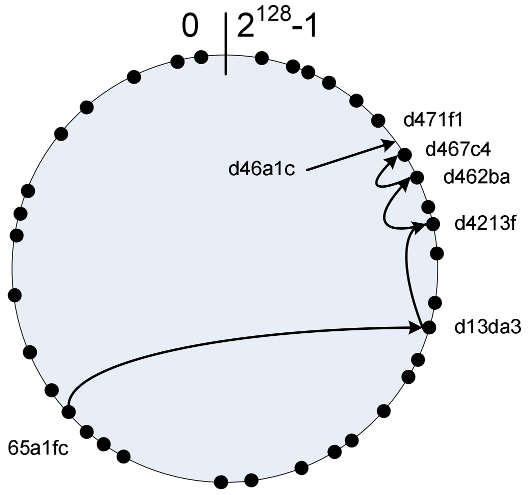

Fig. 5: Example routing of key d46a1c from node 65a1fc in Chord

Like Pastry, Chord guarantees delivery in a network with high churn rates, where the churn rate is defined as the number of nodes leaving the network in a given period of time divided by the average number of nodes in the network. In Chord, this is achieved by requiring only one entry in the finger table to be correct for a successful (if slow) lookup.

An official implementation of Chord written in C++ is available; however, even the official website admits that it is not well documented. Since January 2006, a Java implementation is available as well [8].

2.2.4 Tapestry

Tapestry [9] is a P2P routing service based on the routing and localization algorithms proposed by Plaxton et al [11], but adds dynamic operations.

Each node has a Node Id and a neighbor map with multiple levels, in which at every level n it stores close nodes (in network distance) whose Node Id is identical in the first n digits but differs in digit n+1. For every possible value of digit n+1, at least one node is stored. To route a message, the sending node (n) forwards it to the node (f1) in its neighbor table whose id matches as many digits of the beginning of the receiver node's id (r) as possible. Then the node f1, whose id matches the first k digits of the destination id, searches in its neighbor map the node who matches k+1 digits and forwards it to that node (f2). This process is repeated until the message reaches its destination.

While the routing mechanism is very similar to the one of Pastry, Tapestry provides an additional feature for storing objects. Each node which stores an object publishes this information by sending a message to the root. Each node on the way there sees that node n stores the object and keeps that information. If a node needs an object, it sends a request to the root node. If a node on the way there has information about that object, the request is automatically sent to the node who stores the object. This way, finding the closest replica of an object is made very easy, since a node who receives multiple „publishing“ messages for the same object can find out which replica is closest.

This feature has the advantage that any node can store every object while in Pastry, the object (or a link to it) must be at the node whose id is closest to the object’s id.

An implementation of Tapesty in Java is available.

2.2.5 Content-Addressable Network (CAN)

CAN [10] presents functionality which is similar to that of Chord and Pastry. It is based on a d-dimensional Cartesian coordinate space on a d-torus. The nodes in the CAN now self-organize into an overlay network that represents this virtual coordinate space. Every node in the system has the responsibility for the data in a clearly defined part of that coordinate space, and every part of the space is covered by exactly one node. For each key K, a point P in the coordinate space is found by hashing the key and the node responsible for point P now stores K. To route a message, a node calculates the straight line to the destination and then forwards the message to its neighbor in that direction. CAN also provides routines for bootstrapping, but assumes that the new node already knows one node in the network. To join, the new node simply chooses a random point in the coordinate space and finds the node responsible for that point. Then the area of that node is divided and half of it is assigned to the new node.

The CAN-Project does not seem to have a website and no implementation is available.

2.3 Security in Peer-to-Peer Systems

A major issue in peer-to-peer based applications is security. Since there is no centralized server and since any host can join the system, no node of the network can be reliably trusted on. Research in this area seems to be done almost exclusively for data sharing p2p systems, but even though XBAC is not a traditional data sharing system, most of the concepts found in that research can be applied to XBAC.

The paper “Open Problems in Data-Sharing Peer-to-Peer Systems” [17] by Daswani, Garcia-Molina and Yang looks at problems in to categories: search and security. Here, we give an overview only of the security section. The authors identify four key security issues in data sharing p2p systems: availability, file authenticity, anonymity and access control.

Availability of nodes in a p2p system is important for both the system and the node itself. A denial-of-service (DoS) attack attempts to make a node unavailable by overloading it. The most obvious DoS attack does this by using up all of a node’s bandwidth [17]. A malicious node can not only do this by directly sending messages to the victim, but it can abuse the p2p system for its purpose. In the once widely used p2p network Gnutella, malicious nodes maneuvered themselves into a “central” position in the network and then responded to every query that passed through it claiming that the victim node had a file satisfying the query (even though it did not). Thus, all nodes who submitted such a query then contacted the victim node, overloading the victim’s resources [17]. In addition to DoS attacks on bandwidth, it is also conceivable that similar attacks overloading CPU (sending a modest number of complex queries) or storage (submitting large bogus documents) could be attempted. Aside from DoS attacks, node availability could be attacked by malicious nodes infiltrating the victim and directly shutting it down.

To prevent attacks on availability, the authors propose that p2p protocols need to have tighter constraints to prevent them being used as amplification mechanisms or provide a level of fault tolerance in the face of node failure, something which protocols like Chord, Pastry and CAN provide.

The second key security requirement in p2p systems is file authenticity. In a data sharing p2p application, a query might lead to multiple files or to files that do not contain the desired information. Dasmanii, Garcian-

Molina and Yang propose four options to find out which file is the authentic one. The first approach is to simply consider the oldest document submitted to the network to be the authentic one. In the expert based approach, the node which originally submitted the document to the keeps track of all digital signatures of the file. However, if that node is unavailable at any particular time, infiltrated by an attacker or a malicious node itself, it may be difficult to properly verify files it authored. The voting-based approach uses many experts instead of just one expert node. Depending on the nature of the system, experts are either qualified humans or nodes that „vote“ for a file if it is stored on the node. The latter approach assumes that users delete the files that they do not deem authentic. The last approach is a reputation-based system which extends the expert based approach by additionally weighting the votes of experts according to their trustworthiness in previous votes.

Anonymity in p2p networks can be provided on the network layer as well as on the application layer. The authors of "Open Problems in Data Sharing peer-to-Peer Systems" only discuss application layer anonymity. They have identified four types of anonymity that can be desirable: Author anonymity means that adversaries cannot determine which user created a document, with server anonymity adversaries cannot determine which server stores a document., reader anonymity means that adversaries cannot determine which user accesses a document and finally document anonymity prevents adversaries from determining which documents are stored on a given node.

Anonymity always affects other aspects of the system; usually, there is a tradeoff between anonymity and performance. If, for example, server anonymity is desired, an efficient search is impossible. Currently, systems providing anonymity mostly achieve this by routing requests through a chain of intermediate proxies, like that neither the server nor the node serving as a proxy know whether the node they received a request from is the originator of the request or simply another proxy.

The last aspect of p2p security the authors identify is Access Control. Ideally, a p2p system should be able to restrict access to documents to those customers who have the right to access them or who have paid for them. However, the authors do not provide a concept how this could be achieved.

2.4 Overlay Networks

2.4.1 mOverlay

In common Peer-to-peer networks, usually a central server or a randomly generated overlay network is used. While a central server has the problem of not being scalable, a random overlay network can require many long-haul message deliveries.

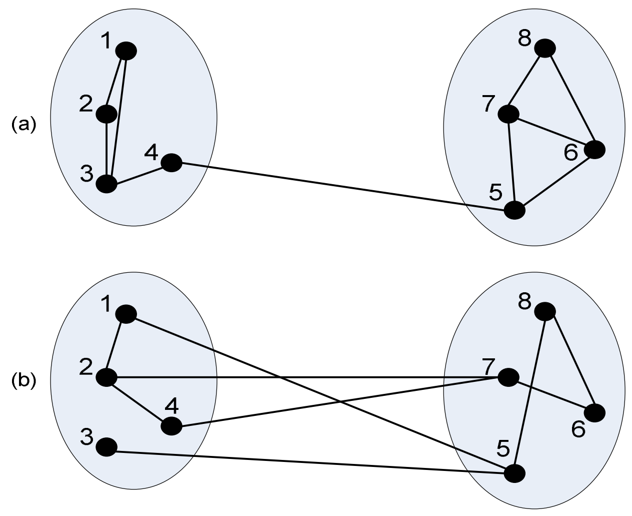

Fig. 6:Example of a locality aware overlay (a) and a randomly connected overlay (b)

With mOverlay [13], a network is constructed which takes into account the locality of the hosts by using dynamic landmarks. To achieve a locality-aware overlay, groups of hosts that are close to each other are formed based on the following criterion:

"When the distance between a new host Q and Group A's neighbor groups is the same as the distance between Group A and Group A's neighbor groups, then host Q should belong to Group A." [13]

Thus, the neighbors of a group act as dynamic landmarks. mOverlay also presents a detailed bootstrapping algorithm illustrated in Fig. 7. A new node needs to know a central rendezvous point (RP), which can supply the node with a bootgroup (Group 1), who in turn gives the new node information about its neighbor groups. The new host then makes measurements to these groups and checks if the grouping criterion for host 1's group is met. If this is the case, the new node belongs to this group, otherwise it evaluates which of the neighbor groups is closest and asks that group for its neighbors. This way, the new node gets closer to its correct group step by step, the process is repeated until the node either finds its group or a stop criterion is met (usually a number of iterations of the described process). If no suitable group is found, the new node creates a new group for itself.

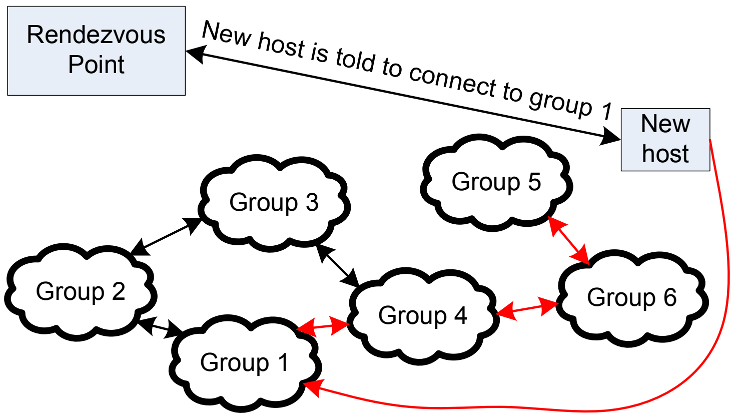

Fig. 7: Bootstrapping of a new host in mOverlay

In the example of Fig. 7, the new host first tries to connect to Group1, then Groups 4 and 6 until it finally finds its closest Group, Group5.

Even this simple example shows that finding a group in mOverlay can involve many steps and therefore many measurements.

2.4.2 Meridian

Meridian [14] is a framework for performing node selection based on network location. It can be used to address three commonly encountered problems: closest node discovery, central leader election and locating nodes that satisfy target latency constraints in large scale distributed systems.

Since for XBAC, only the closest node discovery is of interest we will discuss here only this aspect of Meridian.

Each Meridian node keeps track of a small, fixed number of other nodes in the system and organizes this list of peers into concentric, non-overlapping rings [14]. These rings represent the network distance of the peers stored in the ring, the radii of the rings increase exponentially. When the node has measured the distance to a peer, it places that peer in the corresponding ring. Since the number of nodes in a ring is limited and because the radii of the rings increase exponentially, a node keeps track of almost all nodes that are close to itself while of the more distant nodes only a small part is known. This allows a node to authoritatively answer geographic queries for its region of the network.

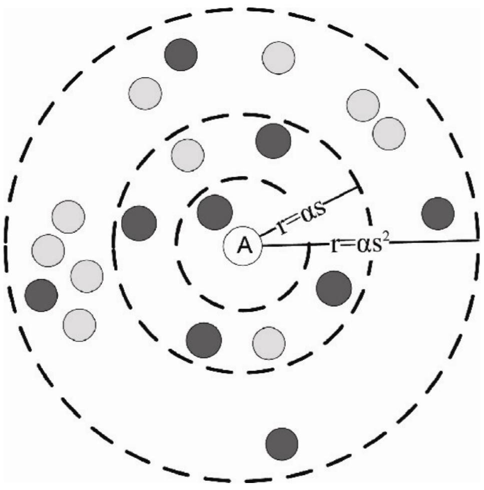

Fig. 8[14]: Each Meridian node keeps track of a fixed number of other nodes and organizes them into concentric, nonoverlapping rings of exponentially increasing radii.

Requests for closest node discoveries in Meridian work similarly to lookups in Pastry or Chord: the distance to the target is reduced exponentially with each hop. Other than in Pastry and Chord, the distance used for Meridian is not a numeric one but the physical latency. Since this latency can be measured for hosts outside of the Meridian network, all Internet hosts can be used as the target of a closest node request in Meridian.

When a Meridian node receives a request to find the closest node to a target, it determines the latency d between itself and the target. Once this latency is determined, the Meridian node simultaneously queries all of its ring members whose distances are within $(1-\beta) \cdot d$ to $(1+\beta) \cdot d$ where β is an acceptance threshold. These nodes measure their distance to the target and report the result back to the Meridian node. Nodes that take more than $(2\beta + 1) \cdot d$ to provide an answer are ignored, as they cannot be closer to the target than the Meridian node currently processing the query. If there are peers who meet the acceptance threshold, the one closest to the target is chosen to perform the same procedure again, if no such peer was found, routing stops and the closest node currently known is chosen.

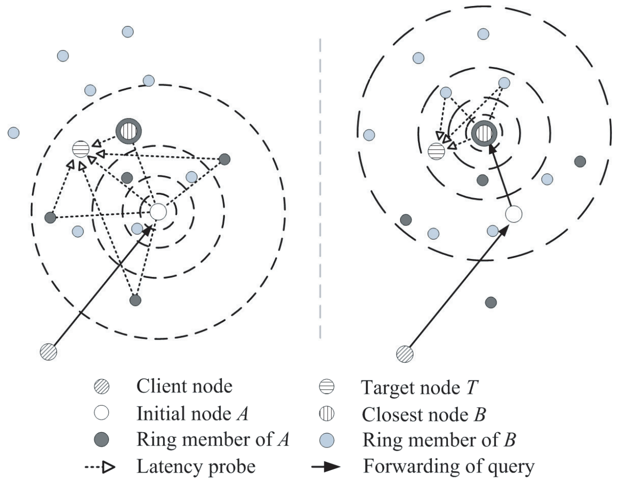

*Fig. 9[14]: A client sends a "closest node discovery to target T" request to a Meridian node A, which determines its latency d to T and probes its ring members between (1 − β) * d and (1 + β) * d to determine their distances to the target. The request is forwarded to the closest node thus discovered, and the process continues until no closer node is detected.*

Meridian is implemented in C++, additionally, a language for safely executing general-purpose localization queries on Meridian hosts called Meridian Query Language (MQL) is provided.

2.5 Planetlab

Planetlab [18], [19] is a network testbed that is designed to support both researchers that are developing new services as well as clients who want to use these services.

It adheres to four design principles. Firstly, services should be able to run continuously and should be able to access a slice of the overlay's resources. The second principle is that control over resources should be distributed while the third is that overlay management services should be unbundled and run in their own slices. Finally, APIs should be designed to promote application development

All hosts which are part of XBAC must be fully dedicated to Planetlab, thus making it possible to prescribe the permitted hardware configurations.

The development of Planetlab is planned to go through three phases. In the seed phase: starting with 100 nodes, only a small, known set of researchers works on Planetlab. This phase is followed by the "Researchers as Clients" phase, where Planetlab is opened to the research community and up to 1000 nodes are expected. Possibly, researchers begin to use primitive applications deployed on Planetlab. In the third and final phase, the goal is to attract real clients who are using the innovative services developed by the researchers.

Currently, Planetlab is in phase 2, consisting of over 600 node distributed over all five continents.

2.6 Statistical Foundations

2.6.1 Time Series

An important concept for XBAC is the concept of Time Series, which we rely on to provide statistically based predictions. Time Series are collections of data points spaced apart at uniform time intervals. In the case of XBAC, these data points are the measurements that are made. Many measurements in XBAC will be actively performed by XBAC and therefore can be made at the specified time intervals. However, since passive measurements are supported as well, a normalized time series is computed by performing a linear interpolation of the input series.

Time Series data is usually assumed to consist of a systematic pattern and noise (error), which usually makes the pattern difficult to identify. The pattern itself can be described in terms of two basic components: a long-term trend and seasonal changes.

Using the example of round-trip-time measurements between two hosts, the long term trend can be expected to be stable or maybe slightly decreasing over years because of faster processing times and new data link technologies. The situation for seasonal changes can be expected to be completely different: Depending on the route of the signal, the round-trip time is likely to change at different times of the day because the number of Internet users varies greatly.

2.6.2 Auto-Regressive Model (AR-Model)

With the help of an Auto-Regressive Model [21], a linear least squares regression of the current value of the series against one or more prior values of the series can be performed.

An AR-Model of order p (e.g. taking into account p previous measurements) has the following form:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t$$

where

- X_t is the predicted value,
- X_{t-1} to X_{t-p} are the previously made measurements,
- ϕ_1 to ϕ_p are the autoregressive model parameters.
- c is a constant and
- ε_t is called error term.

Simplified, a prediction is made up of a random error (ε_t) and a linear combination of prior observations.

The main task with AR Models is to specify appropriate parameters. For this, the equations of Yule-Walker [22] were used.

The Yule-Walker equations are defined as

$$\gamma_m = \sum_{k=1}^{p} \phi_k \gamma_{m-k} + \sigma_\varepsilon^2 \delta_m$$

where $m = 0 \ldots p$, yielding $p+1$ equations. γ_m is the autocorrelation function of X, σ_ε is the standard deviation of the input noise process, and δ_m is the Kronecker delta function.

The last part of the equation is non-zero only if $m = 0$, therefore the equation can be solved by representing it as a matrix for $m > 0$. This leads to the system of equations

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \ldots \end{bmatrix} = \begin{bmatrix} \gamma_0 & \gamma_{-1} & \gamma_{-2} & \cdots \\ \gamma_1 & \gamma_0 & \gamma_{-1} & \cdots \\ \gamma_2 & \gamma_1 & \gamma_0 & \cdots \\ \ldots & \ldots & \ldots & \ldots \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \ldots \end{bmatrix}$$

which solves all ϕ.

For $m = 0$, using

$$\gamma_0 = \sum_{k=1}^{p} \phi_k \gamma_{-k} + \sigma_\varepsilon^2$$

allows to solve the equation.

3 XBAC Architecture

In parallel to the work we present in this master thesis, Matthias Scheidegger has developed parts of XBAC (eXperience Based Admission Control) [1] [2]. While the goal of this thesis is to provide the design and the communications aspects of XBAC, Matthias Scheidegger worked on the architecture, the clustering algorithms and on the statistical background of the system. In this chapter, we will present his work.

"Endpoint Cluster Identification for End-to-End Distance Estimation" [2] proposes a peer-to-peer measurement service based on clustering endpoints that show virtually identical QoS properties when viewed from outside the Cluster. It presents a method for remote identification of Clusters which allows detecting Clusters containing hosts that are part of the peer-to-peer network as well as regular Internet hosts.

3.1 Clusters

3.1.1 Cluster Definition

In [1], a Cluster is defined as a group of nodes whose distance to all other nodes in the network is always similar. Given the set of endpoints in the network N and a distance function d, $C \subseteq N$ is called a Cluster with respect to d if and only if, for all points t in time

$$\left| d_t(o,n) - d_t(o,m) \right| < \varepsilon, \forall n,m \in C, \forall o \in N \setminus C \text{ [1]}$$

where ε is a threshold that depends on statistical and measurement errors. Additionally, a cluster hierarchy is introduced: Given two Clusters C and D where $C \subseteq D$, C is called a Subcluster of D and D a Supercluster of C. Two nodes of the same Cluster are called neighbors. Note that this is not the definition of a Cluster we use in our work. The differences between the definitions and the reasons for having a separate definition will be shown in Chapter 4.1.

The distance function used in [1] must comply with only one requirement: for all nodes n in the network, $d_t(n,n) = 0$ must be true. Neither symmetry nor the triangle inequation are required, thus it is possible to use network round-trip time as a distance function.

In each Cluster, a “Cluster leader” coordinates the measurements of the “subordinate nodes” and collects the results in a database. Based on these results, it can then build a Time Series and make predictions.

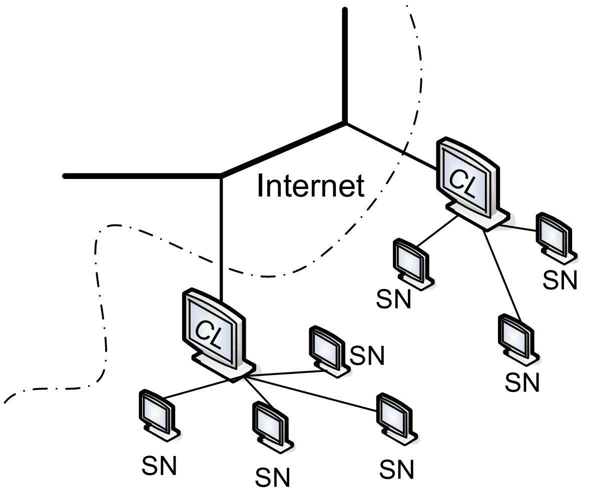

Fig. 10 Possible Cluster Structure. CL designates Cluster Leaders, SN subordinate nodes

3.1.2 Distance Calculation

To decide whether two nodes are neighbors and should belong to the same Cluster, a distance difference calculation between time series is applied. The parameters of this calculation can be adapted to suit the chosen distance function. Distance values close to 0 indicate a neighborship.

The distance calculation consists of two separate functions: $O_b(x,y)$, the first one compares individual measurements of the time series and checks if the values of the time series lie within an n-percent band of each other. The second function B(x,y) compares the mean values of the time series. Parameters are the size of the band in function $O_b(x,y)$ and acceptance thresholds for both functions.

3.2 Evaluation

Three experiments evaluated the Cluster recognition method in [2]. The first one used averaged round trip times: each value in a time series represented the mean of 10 RTT probes. After the Clusters were recognized, the results were verified by further observing the behavior of recognized neighbors. Depending on the used parameters, the ratio of correctly detected neighbors ranged from 100% to 85%. However, the parameter configurations that led to a better recognition ratio also recognized significantly fewer pairs of neighbors. A second verification method used was to compare the round-trip times between two neighbors to all other round-trip times in the network, showing also that a high percentage of the recognized pairs of neighbors are correct.

In the second experiment, non averaged round-trip times were used. Again, with certain parameter configurations a recognition ratio of 100% was achieved, however, the algorithm performed clearly wore with less strict parameters.

The third experiment used the available bandwidth as a distance function, achieving similar results as the second experiment, reaching 100% with strict parameter configurations. In this experiment, B(x,y), the second function of the distance calculation, showed to be significantly more influential than before.

4 XBAC Peer-to-Peer Design

In this Chapter, we present the design aspects of XBAC developed specifically for this thesis. We will define the design requirements, describe the design and show the mechanisms used for example for bootstrapping or for making predictions.

4.1 Terminology

4.1.1 Group

One of the basic concepts of XBAC is to treat hosts with identical or nearly identical behavior as a Group. Groups may contain only hosts that are part of the XBAC network. To form a Group, we analyze the behavior of its members towards hosts which are not part of the Group. Thus, we define a Group as follows:

A Group is a set of hosts which have nearly identical QoS behavior towards and from all other nodes in the network.

Supposed that hosts A1 and A2 are in the same Group, this means that

- concurrent prediction requests from a host B to A1 and A2 will always result in virtually the same prediction.
- concurrent prediction requests from hosts A1 and A2 to a host B will always result in the same prediction.

In XBAC, all members of a Group must be part of the XBAC-network and they must be aware of being a member of the Group.

Our definition of a Group is basically identical with the one used in [1]: In both systems, Groups are the basic organizational units and consist of hosts that can be treated equivalent in terms of distance measurements.

4.1.2 Cluster

In Chapter 3, we described the definition of a Cluster taken from [1]. In that definition Clusters are globally valid, which would mean for XBAC that all Groups use the same Clusters. This would lead to a highly complex communication between Groups to maintain and update information about Clusters. Therefore, we use here a simplified definition that defines Clusters only from the point of view of a single Group or node. The definition from [1] is best seen to be the theoretical model behind the definition we give here. From this point on, we will use only our definition in this document.

A Group can organize other Groups and single Internet hosts into a Cluster. Other than Groups, Clusters may contain hosts not part of the XBAC network. For the reasons given above, a Cluster exists only for one Group. We use the expression $\text{Cluster}_3(1)$ for the first Cluster of Group 3.

A Cluster is a set of hosts which have nearly identical QoS behavior to all hosts in the Group which uses the Cluster.

For XBAC, we require that Groups are atomic and therefore cannot be divided. This means that if host A belongs to a Cluster, hosts in the same Group as A must belong to the same Cluster as A.

The following three figures exemplify the concepts of Groups and Clusters. In Fig. 11, a network containing three XBAC Groups and many non-XBAC nodes are shown. Fig. 12 shows what this network looks like from the perspective of Group 1: Groups 2 and 3 as well as many single hosts are contained in one single Cluster, $\text{Cluster}_1(1)$. Fig. 13 then shows the same network from Group 3's view. Now, Group 3 is augmented by some single hosts to form Cluster 1, while Group 1 is identical with Cluster 2. In this case Groups 1 and 3 can impossibly be clustered together because they are at greatly different distances from Group 2.

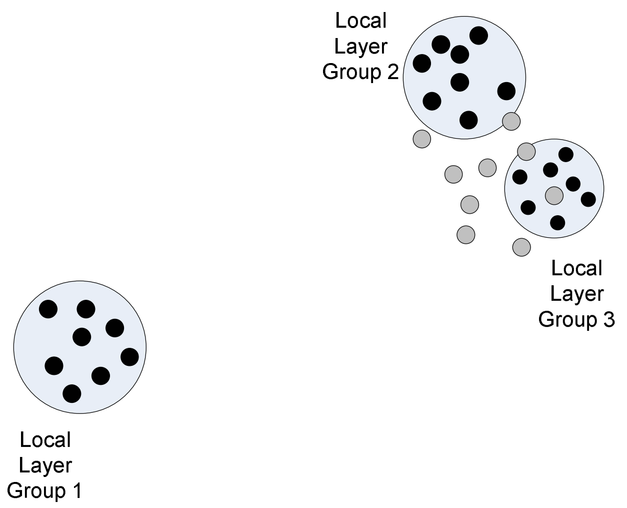

Fig. 11: Example network topology. XBAC nodes are shown black.

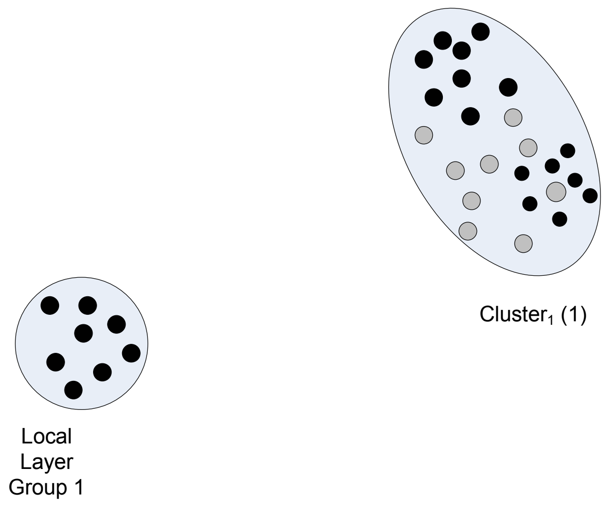

Fig. 12: the network viewed from Group1

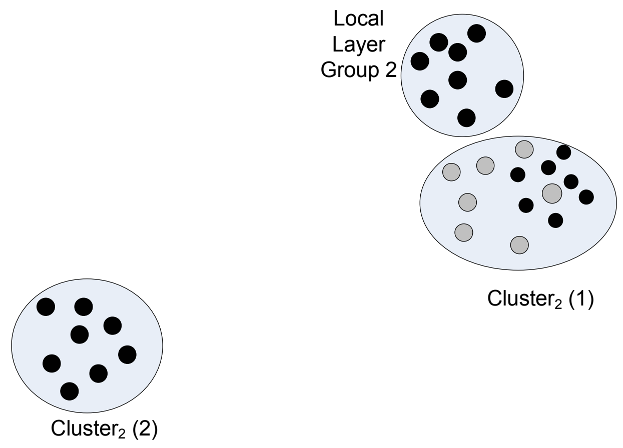

Fig. 13: The network viewed from Group2

The described concept of Groups and Clusters allows us to exploit almost as many similarities between hosts as possible. On one hand, we collect all the measurements made by hosts in the same Group, while on the other hand, a Group clusters hosts that seem to behave identical from the Group's point of view to achieve even better performance.

Since Clusters are intended to contain both XBAC- and non-XBAC-hosts, members of a Cluster do not need to be and generally are not aware of being part of the Cluster.

In section 7.1 of the Chapter "Future Work", we will show further possibilities to exploit similarities between measurements that were not implemented in XBAC.

4.2 Design Requirements

Most of the design requirements for XBAC are directly derived from the objectives described in Chapter 1. The objective influencing the design the most is that the static infrastructure should be limited to one connection host or URL. This helps making XBAC more robust since node failures or denial of service attacks would need to affect large parts of the network, additionally, it reduces the effort to set up the network, but it means that all functionality must be distributed to the users of XBAC. Since these users will not use XBAC if the software significantly affects the performance of their computer, the load in terms of bandwidth, storage and CPU usage should be distributed as well as possible. Even though some nodes (the Group leaders) will have additional responsibilities, this must not be noticed by the user.

Because any user should be able to join and leave XBAC whenever he wishes to do so and because connections in the Internet cannot be relied on, the architecture must be stable in networks with high churn rates (many nodes are leaving the network) and with unexpected node failures. Unless multiple nodes fail concurrently, the system should not significantly suffer from such events.

4.3 XBAC Layered Design

4.3.1 Overview

To meet these requirements, we propose a two layered architecture. On the Global Layer, Group leaders are loosely connected, while on the Local Layer, well organized Groups are connected in peer-to-peer rings.

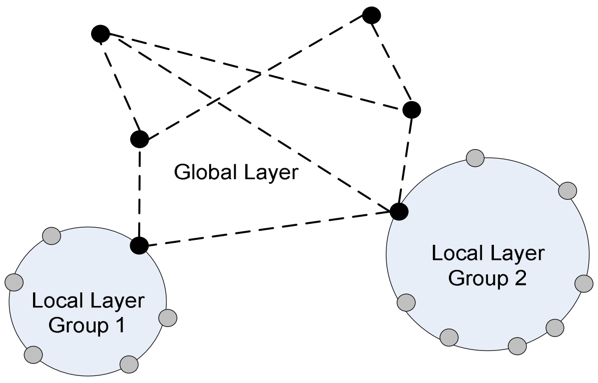

Fig. 14: Example Layer architecture. Group Leader nodes are shown black.

The main responsibility of the Global Layer is to assign a newly joining host to the closest Group. This is consistent with the requirement mentioned above of distributing the load in the network as well as possible. Additionally, every XBAC node provides basic information about itself and its Group on the Global Layer.

The Local Layer on the other hand has a much wider area of responsibilities. It has to store distance information and prediction models, measure distance information and update stored data, provide requested prediction models and ensure self-organization of the Group.

In addition to the two layers, every XBAC node can be contacted directly with requests for basic information about itself and its XBAC Group.

4.3.2 Identifiers

Ids in XBAC are 128 bit identifiers. They are derived from an id-string using MD5 hashing. Therefore, creating two ids from the same string will result in two identical ids.

We define four types of ids: HostIds, NodeIds, GroupIds and ClusterIds. A HostId identifies a host on the Internet. From every IP address, it must be possible to derive a unique HostId. NodeIds are also derived directly from the IP address of the computer, but only apply for XBAC-Groups and are only used within an XBAC Group. Thus, a computer can be assigned both a NodeId and a HostId. Every XBAC Group has a globally unique GroupId. This id is derived from the Group leader’s IP address and from the date of creation. Finally, a ClusterId is assigned to every Cluster in XBAC. Since each Cluster exists only for one Group, ClusterIds do not need to be globally unique. We derive them from the first host in a Cluster and the time of creation.

To be able to make predictions, a Group has to be able to perform the following tasks: For a given IP address, the corresponding ClusterId must be found if the IP address belongs to a Cluster. Similarly, the corresponding ClusterId must be found for a given GroupId, if the Group belongs to a Cluster and finally, for a given ClusterId, the corresponding prediction model must be found.

From the first of these tasks, we can derive an important requirement for HostIds:

The globally unique HostId of every Internet host must be derivable directly from nothing but the hosts IP address.

4.3.3 Roles

In order to be able to reference the roles an XBAC Node can have in the following sections, we will introduce those roles here. In XBAC, a node can have three roles: “Member”, “Leader” and “Backup leader”. Every node always has the role “Member” which is the role with the most functionality. In each Group there is one node to which the role “Leader” and a specified number of nodes to which the Role “Backup leader” are assigned to. A node cannot have both the roles “Leader” and “Backup leader” at the same time.

Role “Member”

The role “Member” contains all functionality every XBAC node has to be able to perform at any time.

Provide basic information
: Provide information such as the Id of the Group this node belongs to, the IP address of the node itself and the IP-address of the Group Leader.

Basic ring functionality
: Handle ring functionality to assure the ring as an organizational unit work (e.g. routing messages, ring maintenance).

Perform measurements
: Every member can be asked to perform a measurement to any IP address.

Handle Requests
: If a request for a ring-id is routed to the node, the node has to be able to take the appropriate action. This may mean to insert or delete an entry at this id or to fulfill a request related to the specified id.

Maintenance
Periodically, a node needs to make sure the data it stores is still up to date and useful. This includes checks whether stored Clusters need to be split as well as broadcasting a summary of the Cluster's data to other Group members so they can assess whether Clusters could be merged.

Cache Recently used Data Models (optional)
For optimization reasons, the member can store recently used data models and provide a faster lookup if the same model is requested soon after the first request.

Role "Leader"

The leader of a Group is responsible for managing the members of the Group and for ensuring that vital information about the Group is sent to one or more Backup leaders.

Provide Neighbor Info
Provide a list of Group Leaders that are close to the Group in terms of network topology.

Node Management
Handling of nodes that wish to join or leave the Group.

Backup
Periodically, data about the Group is sent to the Backup leaders. In return the Backup leaders acknowledge that they are still alive, if the Leader does not receive such an acknowledgement from one of its Backup leaders, it has to choose a different one.

Role Backup Leader

The Backup leader is responsible for storing backup data and for taking over the Group if the Leader fails.

Backup
The Backup leader receives and stores the data needed for the survival of the Group. In return the Backup leader acknowledges it being alive to the Leader.

Take over Group
If the Backup leader has not received an „alive" message from the leader for a specified time, it broadcasts the Group that it is the new Group

leader and it contacts other Group leaders to represent the Group on the Group layer.

4.4 Bootstrapping Process

To bootstrap to the network, a new node essentially has go through two steps: First, it has to find out which Group is the one closest to itself, and then it needs to find out if it meets the criteria for joining that Group.

The first issue is a technological issue, which we will discuss in Chapter 5.

For the second step, we chose to apply mOverlay's grouping Criterion [13] to find out whether the new host belongs to this Group or should form his own Group. The criterion is stated as follows (see Chapter 2.4.1 for a more detailed description of mOverlay):

"When the distance between a new host Q and Group A's neighbor Groups is the same as the distance between Group A and Group A's neighbor Groups, then host Q should belong to Group A." [13]

This means that the Group leader chooses a set of other nearby Group leaders, the new host then measures its distances to them and compares them with the distances of the Group leader. As of now, we assume that the distance between two hosts is always measured as the round-trip time between then.

Additionally to those criteria derived from mOverlay, we used a simple threshold for the distance between Group leader and new host:

If the new host meets all of these three criteria for the Group leader found through Meridian, he joins the Group; otherwise he forms his own Group.

A problem could possibly arise with this algorithm if XBAC takes into account QoS attributes other than RTT and the grouping criterion is changed. Imagine two hosts very close in network topology but with different bandwidths to their ISP. If XBAC promises to predict bandwidth, this must be considered for the grouping criterion and the two hosts should not be in the same Group, so we assume that they are the leaders of their respective Groups. When a new host connected to the same ISP joins, only one of the two Group leaders is chosen by Meridian, with a 50% possibility that it is the wrong one. However, this problem could be solved quite simply by having Meridian return multiple Group leaders or by fetching other nearby Group leaders if the first Grouping attempt fails.

4.5 Quality of Service Predictions

If XBAC receives a request to make a QoS prediction for an IP address, the Group where the prediction is requested has to find the prediction model appropriate for the provided IP.

This involves the following steps:

- Derive the globally unique HostId from the provided IP address. As mentioned before, any node of the XBAC network has to be able to do that given only the IP address.
- Find out whether there is an entry in this Group which maps the HostId to a ClusterId.
- If no such entry exists, contact the remote host to find out whether it is part of an XBAC Group.
- If this is the case, find out whether there is an entry which maps the remote hosts GroupId to a ClusterId.
- If no ClusterId could be obtained with these measures, the prediction request cannot be fulfilled and the remote host is added to a Cluster of which will enable the Group to fulfill future prediction requests.
- If a ClusterId has been obtained, the corresponding prediction model is fetched and the prediction is made.

The main difficulty in perfoming these steps is that all information is stored distributedly in the Group. The following example illustrates this process, assuming the situation of a host for which no direct mapping to a Cluster exists, but whose Group is known in the local Group.

Node A needs QoS information from itself to a given IP address (host F).

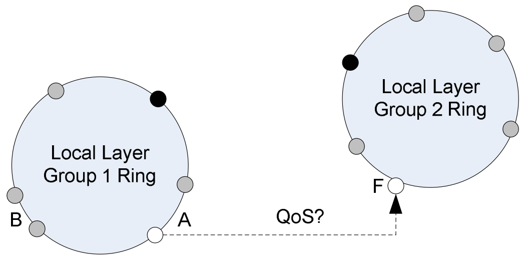

Fig. 15: Desired prediction

Node A derives F's HostId from it's IP-address and initiates a lookup for the HostId to find out to which ClusterId that HostId is mapped. B, the node which is responsible for this HostId, returns that the Id is unknown.

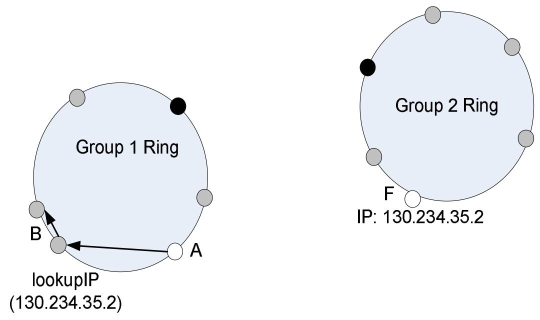

Fig. 16:The lookup for the HostId

Then node A directly contacts F and asks it, to which Group it belongs. Since F is part of XBAC, it returns this information.

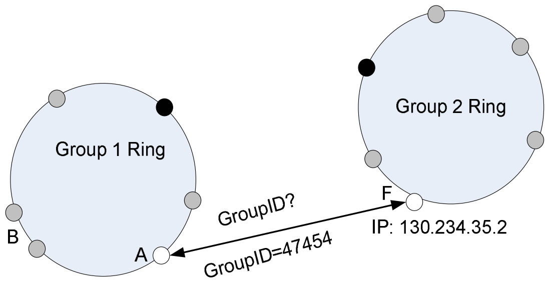

Fig. 17: Direct request for the GroupId

Then, A initiates a lookup in its Group to find out to which Cluster this Group belongs to.

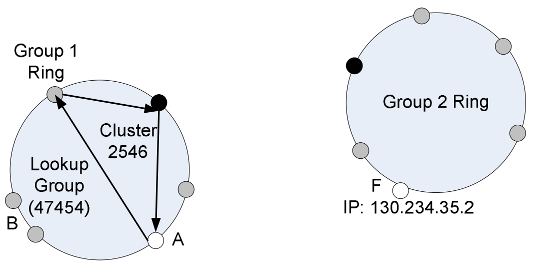

Fig. 18: Lookup for the GroupId

Now A can search for the Prediction Model associated with this Cluster and download it from inside its own Group.

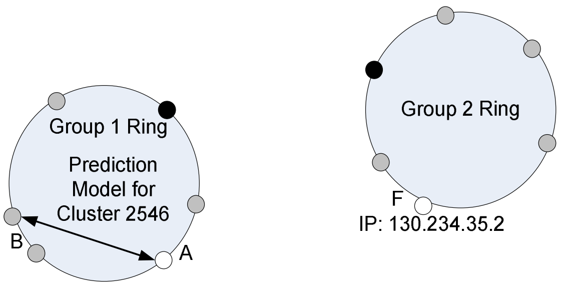

Fig. 19: Retrieve the Prediction Model

4.6 Building Clusters

When a new Group in XBAC is set up, it initially consists of only one host and contains no Clusters. This means that no predictions are possible. As soon as either measured data is supplied by an external application or the external application requests a prediction, the first Cluster is formed containing only the concerned remote host. This happens every time a new host is recognized. The new Cluster's id can be chosen randomly but it needs to be unique and the ids should be well distributed over the range of possible values. The Group member who is responsible for this ClusterId is assigned the responsibility for the new Cluster.

Immediately after the creation of the Cluster and periodically later, the node responsible for the Cluster (Node A) broadcasts a summary of the measurements to the Cluster to all nodes in the Group. Every node compares this summary with all Clusters it is responsible for. If a node (Node B) has a Cluster which appears to be similar, the following procedure is performed:

- Node B sends a merge request with the complete information about its Cluster to Node A
- Node A collects these merge requests and performs a detailed analysis of the Clusters to find out if and with which Cluster its own Cluster should be merged. We assume here that the Cluster submitted by Node B is chosen to be merged.
- Node A dissolves the two Clusters and performs a re-clustering of all the Hosts which were part of the two Clusters. This will result in one or more new Clusters.
- Node A sends the data for the new Clusters to the nodes which are responsible for those new Clusters.
- Node A broadcasts the new mappings between HostId/GroupIds and ClusterIds for all affected Hosts.
- Every node in the Group updates its mapping entries between Host- or GroupIds and ClusterIds if they are affected by the merge.
- As soon as all Clusters are updated, Node A deletes its old Cluster and notifies Node B which then deletes its unused Cluster as well.

The two main difficulties of this procedure are to avoid Clusters being in multiple merging processes at the same time and to handle requests for Clusters that are in a merging process. The first issue is handled rather simple in XBAC: Whenever data of a Cluster is broadcasted or a Cluster is a candidate to merge with another Cluster whose data was broadcasted, it is added to a list of protected Clusters, which means that it will not be involved with any further merging operations while it is on that list. After a specified amount of time, the Cluster is again removed from the list and will be treated as a normal Cluster if it still exists. The second issue we mentioned, requests to Cluster that are in the process of being merged, are handled by throwing a special exception used only in this case. Since merging two Clusters might take a long as a few minutes, this leaves it up to the calling application whether it wants to try again or accept that there is no prediction possible at the moment.

A second measure to improve the quality of a Cluster is to periodically assess the Cluster and split it into multiple smaller Clusters. This is especially important since hosts in a Cluster by definition are only similar viewed from the Group which uses the Cluster. Due to network topology changes, this may change at any time and therefore, Clusters need to be reassessed periodically.

4.7 Node Failures

One of the key requirements for XBAC is that the system needs to be stable in networks with high churn rates. This chapter therefore describes how the case of a regular node failure is handled in XBAC. Failures of Group leaders will be discussed in the next chapter.

First of all, we look at which data is stored on a regular node: This data can be divided into three categories. We have mappings between HostIds/GroupIds and ClusterIds, actual data about Clusters and all data necessary for the Group organization.

Assuming that we base our application on an existing framework for Group organization, we only have to consider the first two categories since we expect this framework to be stable in networks with high churn rates. While a loss of either Id mappings or Cluster data would hurt the performance of XBAC, it would not essentially harm the system: XBAC will be unable to fulfill at most one request for each address, but later queries will work as usual. Therefore, we propose a relatively simple mechanism which only ensures that no data is lost unless two or more adjacent nodes fail simultaneously.

All data in XBAC is tagged with an Id. Thus, every node (Node A) periodically sends all data to the node which would be responsible for the data if the currently responsible node failed (Node B). Obviously, Node B is Node A's direct neighbor in either clockwise or counterclockwise direction. This means that when Node A fails, any lookups for the data will automatically land at Node B, which then realizes that it is now responsible for the data item.

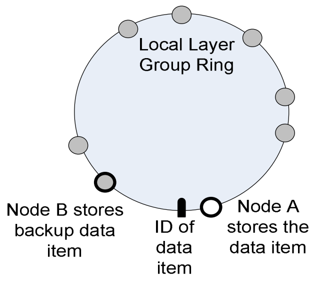

Fig. 20: Example of Data Id with closest Node A and second closest Node B

Since especially Cluster data can be rather large, we add an "expiration time" to each backup data item. As soon as a newer version of the same data item is received, it replaces the older one. When a backup data item expires (i.e. no more recent version was sent), this can have the following two reasons: Either the node responsible for the data item failed, or another node whose Id is closer to the data item's Id was inserted into the ring and the backup data item can be discarded.

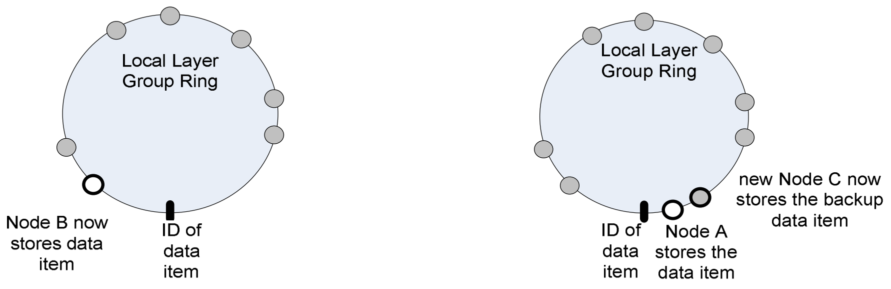
Local Layer Group Ring
Node B now stores data item
ID of data item
Local Layer Group Ring
new Node C now stores the backup data item
ID of data item
Node A stores the data item

Fig. 21: The same Group with the scenarios Node A failing (left) and a new Node C closer to the data item's Id than Node B.

To find out which case applies, the node initiates a simple lookup for the data item's id. If the fist case applies, this lookup will be directed to the local node and will automatically activate the data item, while in the second case, the lookup will be successful and no action is necessary. Therefore, the backup data item is now obsolete and can be deleted.

4.8 Group Leader Failures

While the failure of a regular node in the worst case only affects a few prediction requests, the failure of a Leader node can lead to a Group which is not reachable anymore. This means that no new nodes can join the Group and therefore is it possible that two very similar Groups exist. Even though this is not critical for the operation of XBAC, it is obvious that special measures are necessary in this regard as well. Furthermore, in future versions as XBAC, a Group leader might have even more responsibility such as monitoring the quality of Group members and discovering dependencies between Clusters.

For regular nodes, we propose a system which guarantees the reachability of the Group unless the leader and the backup leaders simultaneously fail. The number of backup leaders (n) can be specified as a configuration parameter. As a default value we used three backup leaders.

The Group Leader randomly chooses n nodes to which it periodically sends all data relevant for the survival of the Group. A node which receives this data automatically assumes the role “Backup Leader” and assures itself periodically that it has received updated data from the Group leader.

If the backup leader notices that no updated data from the leader is available, this might be because either the leader failed (consequently one of the backup leaders should assume the role „Leader“) or because the leader was unable to send the data due to network problems. In this case, the leader randomly chose a different node as Backup Leader.

To find out which of the two cases applies, the Backup Leader sends a request to the leader to check if he still is alive. If this is the case, the Backup Leader can destroy the data received from the leader and becomes a normal member again.

If the Leader does not reply, the Backup Leader assumes that it is the new Leader of the Group. Since every Group member needs to know the Group Leader, the new Leader broadcasts this information in the Group. Additionally, the new Leader has to connect to the network of Group Leaders; the information necessary for this is included in the data it received from the former Group Leader. From then on, the new Leader functions just as a normal Group Leader, choosing a new Backup Leader and sending it the Group data.

Since there are usually multiple Backup Leaders, it is necessary to find a way to determine which of the Backup Leaders should be the new Group leader. This decision is currently made without taking into account the reliability of the possible leaders, but it should be possible to find a method of using such properties in later versions. The current protocol works as follows:

- The initial leader stores a so-called „leader value“ for itself. This leader value is regularly broadcasted in the Group. It then assigns every Backup Leader a unique leader value which is slightly higher than its own leader value.
- Every Backup Leader which notices that the leader is dead, assumes it is the new leader and broadcasts its contact data together with its leader value.
- All nodes who receive this message, check if the leader value received is higher than the one it received from the current leader. If so, the new leader is set and the roles Leader and Backup Leader are dropped if the node had one of these roles.

With that simple protocol, the Backup Leader who has had the role for the shortest time will become the new leader, all other Backup Leaders who assumed the role Leader will drop this role again quickly.

The new Leader then randomly chooses new Backup Leaders and again assigns them “leader values” slightly higher than its own one. Using a data type similar to long in Java, we can now assume that this value can be safely increased for the time the Group exists.

In very rare cases, it is possible that a Group has two leaders. This might happen when the connection between the Leader and a Backup Leader is temporarily broken. Then the Backup leader neither receives messages from the leader nor can it contact him. As long as the connection is broken, this problem cannot be solved. To make sure this state is resolved as soon as the connection is established again, the Leader periodically broadcasts its status to all nodes in the Group. That way, multiple Group Leaders are detected and the Leader with the higher “Leader value” takes over the Group.

The most critical moment for a Group is the one when a new Leader takes over the Group. If the Backup Leader with the highest “Leader Value” broadcasts its information and then immediately fails, all previous Backup Leaders discard their backup data, assuming that the new Leader has chosen new Backup Leaders. Since he did not live long enough to do so, no node will have the data necessary to connect to the network of Group Leaders.

Since this event should be very rare and since currently not having a Group Leader only means that no new nodes can join the Group, no measures against this event have been taken. If in the future such measures are necessary, this could be done by having the old Backup Leaders wait for a specified time to allow the Leader to choose new Backup Leaders and then perform a final check to assure that the new Leader is still alive.

5 Prototype Implementation

5.1 Software Architecture

The software architecture of XBAC consists of four layers:

- Interface Layer: responsible for incoming requests from other XBAC nodes and from applications using XBAC.
- Driver Layer: provides functionality for the roles a node can have
- Model Layer: handles the data which is accessed by multiple roles
- Utility Layer: various tools necessary for the layers above

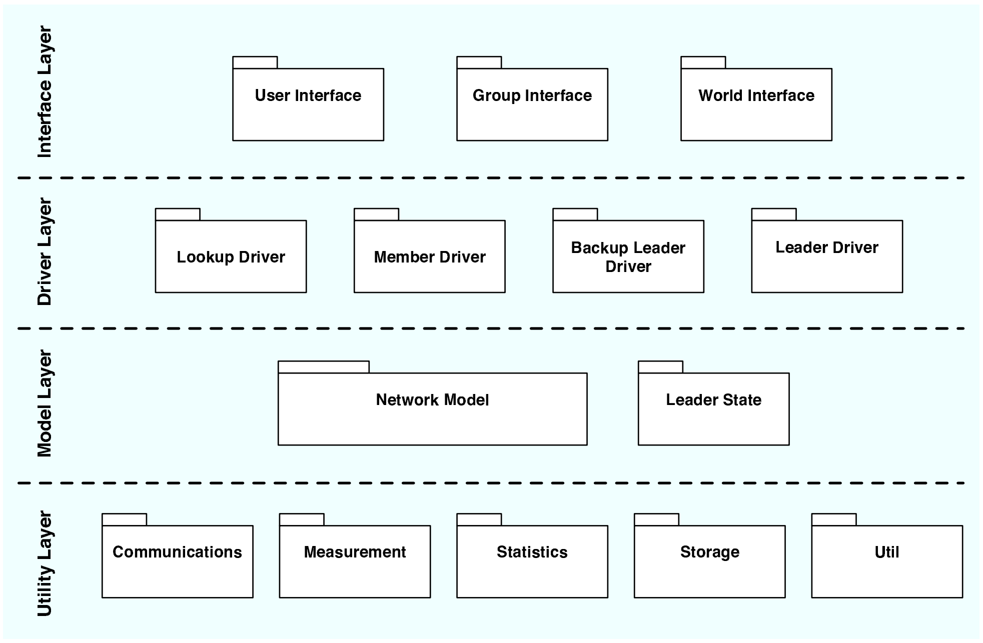

Fig. 22: Software Layers in XBAC

By using a layered architecture, we ensure that the software remains maintainable and can be adapted to changing requirements. If for example a different technology were used for the underlying p2p network, these changes would only affect the package communications.

The following sections give a brief description of each package in the architecture.

5.1.1 Interface Layer

The Interface layer defines three interfaces which can be used by other applications or XBAC nodes. These interfaces are independent of the technology used to call them, they simply register with the appropriate Communicator object (implemented in the “utility Layer”) which then handles the actual communication.

UserInterface
Provides functionality for applications using XBAC such as making requests for predictions and submitting external data.

GroupInterface
Handles requests of other nodes in the same XBAC-Group.

WorldInterface
Handles input from XBAC nodes that belong to a different Group than the local node.

5.1.2 Driver Layer

On the Driver Layer, the role behavior of a node is implemented.

MemberDriver
Handles the basic functionality of the role “Member”.

LookupDriver
The LookupDriver is responsible for everything concerning lookups. While it relies on the NetworkModel for collecting and maintaining the data about Clusters, the mappings between HostIds/GroupIds and ClusterIds are managed directly in the LookupDriver. Additionaly, the LookupDriver is also responsible for most of the backup management.

LeaderDriver
Handles the functionality of the role “Leader” if the node has that role.

BackupLeaderDriver
Handles the functionality of the role “BackupLeader” if the node has that role.

5.1.3 Model Layer

On the model layer, data about the network and about the local Group are modeled. Therefore, most of the data is stored and managed by the Model Layer.

Network Model
This is the actual Network Model in which all Data about Clusters and the Hosts belonging to them is stored and maintained.

LeaderState
Contains everything the Leader needs to know about the Group and about other Groups. The LeaderState is also sent to Backup Leaders so they can access the information if necessary.

5.1.4 Utility Layer

The Utility layer contains tools and algorithms used by the higher layers.

Communications
Handles the actual communication with other computers.

Measurements
Performs measurements to remote hosts.

Statistics
Provides statistical functionality for example to calculate predictions.

Util
Contains various helper classes.

5.2 Used Mechanisms

In this section, we will describe existing mechanisms and software tools we used in implementing XBAC and present our argumentation why we used them. All of the used mechanisms as well as some of those we decided against are described in more detail in Chapter 2, “Related Work”.

5.2.1 Lookup Service

For the Local Layer Ring, we need a scalable, robust lookup service. In Chapter 2, we described p2p lookup services such as Pastry, Chord, CAN and Tapestry together with PAST, a global storage utility based on Pastry.

CAN was quickly ruled out because of the fact that no implementation was available, Tapestry on the other hand seemed to be inferior to Pastry because its additional functionality (object routing) is of no interest for us and only makes Tapestry more complex.

This left us with Pastry, Chord and PAST, which meant we first had to decide whether a simple lookup service or a global storage utility made more sense. PAST has the advantages of providing a tested framework for storing objects and of a built-in security mechanism, but it also has a major weakness when used for our purpose: PAST files are immutable, a new version of a file receives a different FileId than the previous version. Imagine our situation where we need three steps to get a data model from an IP address:

- derive the hostId from the IP address
- find the ClusterId for the given HostId
- find the data model for the given ClusterId

Since the data model file would be immutable, every change would give it a different FileId, making necessary a change for all mappings pointing to that data model and thereby affecting the FileId where the new mapping is stored. However, then the first step is impossible, because a constantly changing FileId cannot be distinctively derived only from the host's IP address.

A possibility to use PAST would be to use both a lookup service and PAST. That way the first steps of our mechanism would be performed by the lookup service, while PAST only stores the actual data model. However, this would mean that the data model is not anymore stored on the node responsible for updating it (the data model's id would change constantly) and therefore induce significant traffic in the Group at a very moderate gain. Based on these considerations, we decided against using PAST in our implementation of XBAC.

The final decision was therefore between Pastry and Chord. While they seem to be very similar in terms of features and performance (an actual performance analysis was not available), Pastry has many advantages. The most important advantage of Pastry is, that the implementation is well documented, while for Chord even the official website discourages from working with the source. Although we did not plan to actually modify Pastry, the documented source greatly helped us understanding and using the framework. Another advantage of Pastry is that it was implemented in Java, while the only implementation of Chord available at the time we evaluated the services was written in C++ (a Java implementation is available since 2006 [20]). Even though C++ most likely performs better, the platform independency of Java seemed to be more important for us. Finally, the Pastry team operates a mailing-list, in which the developers of Pastry promptly and competently answer questions.

All these reasons led to our decision to use Pastry as a routing service in XBAC.

5.2.2 Bootstrapping

In Chapter 4.4, we described the bootstrapping process, using mOverlay's grouping criterion. However, we did not specify on which technology we rely on to organize the Global Layer; that is to find the neighbor Groups for a given Group and to find the closest Group for a joining host.

Since every Pastry node by default maintains a list of its neighboring nodes in terms of Network topology, we assumed to be able to use that list. Unfortunately, we then found out that this list is maintained, but not available through the Pastry API. We therefore had the choice between modifying the Pastry source code and choosing a different technology.

The reason that spoke against modifying the Pastry source was that these modifications would be lost if a new release of Pastry were used. Also, the authors of Pastry clearly stated that they do not intend to make these functionality available in future version of their software.

Thus, based on the analysis of Matthias Scheidegger [1], we used Meridian to find to closest Group for a given node.

To decide whether the grouping criterion (see Chapter 4.4) is met, we use the following parameter values. We denote the distance from the Group leader to its neighbor n with $d_L(n)$ and the distance from the new host to neighbor n with $d_H(n)$.

- The relative difference between $d_L(x)$ and $d_H(x)$ must be smaller then 5% for all neighbors x.
- The absolute difference between $d_L(x)$ and $d_H(x)$ must be smaller than 20 ms for all neighbors x.

The additional threshold for the distance between the new host and the Group leader was defined as follows:

- The absolute distance between the Group leader and the new host must be smaller than 20 ms.

Our experiments showed that these values lead to Groups that correspond with those we expected to be formed. Obviously, these parameters have a great impact on the overall performance of XBAC (see also Chapter 7.4).

5.2.3 Remote Communication

To enable initial contact and communication across XBAC Groups, we need a technology for Remote Procedure calls running on all nodes. We evaluated Axis/SOAP and XML-RPC, two protocols which provide this functionality and which are both based on XML.

Axis[23] is an implementation of a Soap[24]-Engine providing web services. With Axis, it is possible to invoke web services passing any serializable Java object as a parameter. While this flexibility is the greatest strength of Axis, it also has two drawbacks. On one hand, it is known that SOAP XML messages are significantly larger than the passed information itself. The transferred data often is increased by a factor of 25 for requests and a factor of 100 for replies, affecting not only bandwidth usage but also processing time of the XML data. The second problem is that to provide web services, a distinct directory structure is necessary. Since we build XBAC to be able to run on a large amount of nodes, configuring the web services seems to be rather time-consuming.

While the strength of SOAP is its flexibility, XML-RPC's advantage is simplicity. At the time we evaluated XML-RPC, only basic Java types (String, Integer, Double, Boolean, Date, Hashtable and Vector) could be transmitted. However, this has changed recently and the new version of XML-RPC could be used in future versions of XBAC. Compared to Axis/SOAP, XML-RPC is simpler to use and uses a more efficient XML encoding.

Since we use the remote procedure calls only rarely and never need to pass complex objects, we decided to use XML-RPC in XBAC.

5.3 Security

XBAC to our knowledge has two critical security areas: On one hand, all measurements submitted to XBAC are public inside a Group, on the other hand malicious users might connect to the network with altered software, compromising the XBAC network.

With this in mind, we take a look at the four security aspects identified in [17] which we discussed in Chapter 2.3: availability, file authenticity, anonymity and access control.

5.3.1 Analysis of Security Weaknesses in XBAC

If a malicious node aims at hurting the availability of a node, XBAC unfortunately does provide the kind of amplification mechanism described in Chapter2.3: A malicious node can intercept all queries for ClusterIds that pass through it and always reply with the same id, meaning that the node responsible for that Id will receive significantly more requests for Clusters than it should. Even worse, the malicious node is able to make innocent nodes point their entries to the target id.

For XBAC, the resulting unavailability of the target node should not have a great impact. Assuming the underlying Pastry network works as specified and detects the unavailability, the queries simply are delivered to the node which stores the backup. This also means that the target node is relieved from the traffic and should be able to recover. Additionally, since every XBAC Group maintains its own ring, we can expect those rings to be rather small compared to other p2p systems, which will make it more difficult to induce the load necessary to overload a node. Still, it obviously is not desirable that a node can be attacked by using XBAC.

The issue of file authenticity is closely linked to the problem described above. Because any node can reply to any query passing through it, a malicious node can return false predictions and insert false measurements. This is possible for outside applications as well, making the detection even more difficult.

Anonymity is also an important issue in XBAC. When a measurement is submitted to XBAC, other nodes than the one who made the measurement have access to that measurement. This way, they can see which computer accessed which Internet resources. Because the user most likely is not able to control which measurements are submitted, the measurements might include data to sites the user does not like others to know he visited them, so XBAC currently violates data security regulations.

However, of the four areas of anonymity identified in [17], Server and Document anonymity can be neglected because the server where a document is stored is randomly chosen in XBAC. Therefore, nobody is interested in this information being hidden while making the information available significantly simplifies the design of XBAC. The two areas that cause the problem described above therefore are Reader and Author anonymity: A user might not want to reveal information for which websites he requested predictions (Reader anonymity) and which measurements were submitted from his computer (Author anonymity).

The fourth and last security issue identified in [17] is access control, a topic which can be neglected in XBAC because its strength lies in making all measurements available to all nodes.

5.3.2 Desired Security Improvements for XBAC

From weaknesses we have identified, we derived four areas where XBAC should be improved in future versions:

- Nodes which submit false measurements should be detected
- Nodes which make false predictions should be detected.
- A node should be able to answer only requests it is responsible for.
- Submitting measurements and requesting predictions should be done anonymously.

In Chapter 7.2 of the “Future Work” section, will make proposals which show how these issues can be solved.

6 Tests

Testing XBAC was divided in three sections:

- Testing the bootstrap process.
- Testing the functionality of XBAC.
- Simulating the behavior of an Internet user and estimating the benefits of using XBAC as well as the load on XBAC caused by the network.

6.1 Testing the Bootstrapping Process

6.1.1 Test setup

To test the bootstrap process we chose a Group of 9 Planetlab nodes that were all located at the same university and had similar IP addresses, thus we can assume that they are connected in a local area network and behave similarly. Additionally we used reference nodes which were clearly located at a significant distance from each other and from the Group of 9 nodes.

To perform the test, we first started up an XBAC network with the reference nodes and then made the nodes in the Group connect to the network. The test was run with 0, 1, 3 and 5 reference nodes, in each run the 9 nodes connected and then disconnected three times. Additionally, we ran one test with 5 reference nodes in which we intentionally chose one node which we knew to have inconsistent QoS behavior. It should be noted that even with five reference nodes set up, the algorithm only chose the closest 3 Groups in the network for the actual bootstrapping.

The results of this test we were interested in were the percentage of successful bootstrappings by the 9 nodes and the number of false positives, i.e. nodes connecting to an incorrect Group.

6.1.2 Test Results

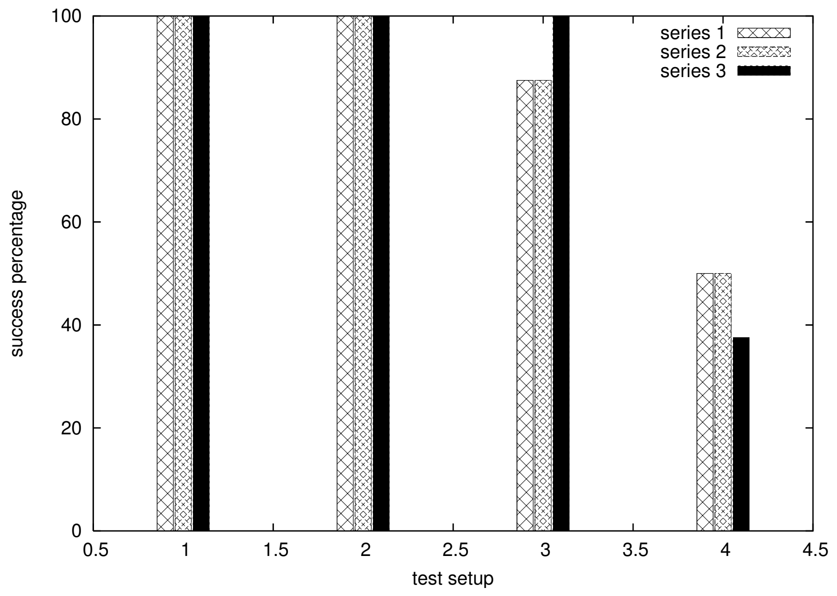

Fig. 23: Results of the Bootstrapping Test

The results show that with reference nodes behaving more or less stable, an almost perfect grouping can be achieved regardless of the number of reference nodes. It is not surprising that the bootstrapping is excellent if there are no reference nodes, because that means that only the distance to the leader is considered. The fact that there were 2 occurrences of unsuccessful bootstrappings with 5 reference nodes is mostly likely caused by one of the nodes' measurements not being as consistent as necessary.

Very interesting are the results of the last test, the one where a reference node with inconsistent measurements was used. As expected, the success rate was significantly lower, but the system nicely recovered: after the first new Groups were created, the inconsistent reference node was not anymore among the closest leaders, thus the nodes that joined later had a much better success ratio.

There were no false positives observed in any of the tests.

6.2 Testing the Functionality of XBAC

6.2.1 Test Setup

To test the functionality of XBAC, we set up an XBAC network consisting of 41 planet-lab nodes. The nodes were chosen in a way so that they are expected to form Groups of different sizes. This is necessary to make sure that the algorithms work in Groups with only one member as well as in Groups with 10 or more members. Additionally, one planet-lab node was set up as an inspecting node. It remotely called XBAC functions at other nodes using the XBAC User Interface (see Appendix B). For testing purposes, it kept track of all the Hosts and Groups each Group has had contact with. The function to be executed next was chosen randomly, however, different probabilities were assigned to the functions.

The following functions were performed:

- Predict the rtt to a random host (probability 40%).
- Predict the rtt to an unknown host which is part of a known Group (probability 40%).
- Shut down a node (probability 10%). If the Group had a size larger than 5, the node was shut down without the node notifying XBAC, otherwise it shut down properly.
- A node joins the network (probability 10%).

After each call to such a function, the network was given 60 seconds to process the new information, then the inspection node used special services used exclusively for testing purposes to assure that:

- all mappings between HostIds /GroupIds and ClusterIds were stored on the correct node.
- all Clusters were stored on the correct node.
- all backup entries were stored on the correct node.
- the Group had exactly one Group leader.
- the Group leader was known to all the members of the Group.

In this test, only predictions to other planet-lab nodes that were part of the XBAC-network were performed. To be able to test both small and relatively large Groups, we tried to choose the nodes in a way so that they would form many small but also some larger Groups. The tests were run for a duration of eight hours.

6.2.2 Test Results

The objective of this test was to show that XBAC works as expected. This was tested by evaluating all mappings and entries stored in the affected Group every time after a function was executed.

In each of these evaluations the results were exactly as they should have been. There was only one instance, in which the result was not what we expected it to be: One time, a “prediction to a host whose Group is known“ did not yield a valid prediction. This happened because the target of the prediction refused the connection when it was asked to return its GroupId, therefore the requesting node did not have that information necessary to make the prediction.

During the test, 343 times a testing function was chosen. Since it was not always possible to perform the function selected (i.e. perform a join operation when no idle node is available), only 190 operations were actually performed. The following table shows which function was performed how many times:

Function	
Prediction to a random host	133
Prediction to a host whose Group is known	40
A node joins	7
A node fails without notification	3
A node fails with notification	7

Table 1: functionality test function distribution

In the test setup, we stated that we want to have both relatively large and very small Groups in the network. Table 2 shows the sizes of the 15 XBAC Groups formed by the 41 nodes used in the test.

Group Size	1	2	3	4	5	6	7	8	9	10	11	12	13
Number of Groups	9	3	0	0	0	1	1	0	0	0	0	0	1

Table 2:size of groups formed in the functionality

Thus we achieved our goal of having both relatively large and small Groups.

6.3 Simulating the Behavior of an Internet User

6.3.1 Test Setup

This test is based on a traffic log collected at the University of Bern. Entries in that log do provide the time at which the request was made and the target of the request, but the host which made the request is not specified.

```
1152611039.453     11 TCP_HIT/200 3574 GET
http://www.weblaw.ch/pictures/thema_jusletter.gif -
NONE/- image/gif
```

The test scenario imitates the behavior of the Internet users whose connections were traced in the logs. We assume that they all use XBAC and, since they all connected to the Internet through the University of Berne's network, that they all are part of the same XBAC Group. Since the log contained approximately 10 million entries that were recorded in a span of 24 hours, it was obvious that we had to process the log before using it.

Our main idea was to choose the target for each test request based on the distribution of targets in the log. Additionally, we simulated the changes of traffic over the course of the day by evaluating the distribution of targets for each hour. Therefore, we used only two elements of each log entry: the time of the request and the target.

In a first step, we divided the log into sets which contained entries for one hour. We calculated for each hour the percentage of the number of its entries compared to the total number of entries. From this percentage, we derived at which interval the test requests would be made for that hour. We defined a target average interval of 60 seconds. This average interval is designed to imitate the time between two requests to different web pages by a user. From the average interval, we derived an interval for each hour based on the percentages calculated before so that over 24 hours, the average interval was the desired one. The formula used was:

$$i_h = \frac{i_a}{p_h * 24}$$

where i_h is the interval calculated for hour h in seconds, i_a is the average interval in seconds and p_h is the percentage of requests for hour h. The number 24 is the number of hours for which the percentages were calculated.

This means that the interval between requests for hours where little traffic was recorded is longer than for hours with heavy traffic.

In a second step, we calculated for each hour the percentage with which the targets occurred. When we performed the tests, we had each node imitate one user. The test started at a specified hour and then requests were made at the specified interval for that hour. The target of these requests was selected randomly, but weighed with the percentages calculated for the targets and the active hour. After every hour of testing, the interval was replaced by the one for the following hour.

There are two categories of results from this scenario we are interested in: How good is the quality of the results of XBAC and how heavy is the load on the system induced by XBAC. Concerning the quality of XBAC, there are three results that are of interest for us:

- The percentage of requests for which a prediction could be made indicates how useful XBAC really is for a user.
- The quality of the predictions.
- The time necessary to make the prediction.

Regarding the load induced on the nodes we are interested in how much of the following resources were used: Bandwidth, CPU, memory and hard disk space.

To be able to compare the effects of the Group size and to analyze the load over time, we performed six tests which differed in the number of nodes in the network and in the time for which they ran.

Nr	**name**	**nodes**	**duration (h)**
1	Full quality test	3	24
2	Single node test	1	4
3	Group size test 1	3	4
4	Group size test 2	5	4
5	Group size test 3	8	4
6	Group size test 4	11	4

Table 3: test setup for user simulation

For tests 2 through 6, we chose the targets recorded between 8 and 12 am from the logs.

One weakness of the tests is that none of the remote hosts are part of XBAC. This can be expected to have the following impact on the results of the test:

- The percentage of possible predictions would be higher if some of the targets would be interconnected in XBAC Groups themselves. The reason for this is that if node A needs a prediction for target X and has data to host Z, this makes a prediction possible if X and Z are in the same XBAC group.
- The time necessary to make predictions is affected in two ways. Every new host is contacted to find out if it belongs to an XBAC Group. Because for non-XBAC nodes, a timeout (default is 1 second) has to be waited for, an XBAC node should reply quicker. On the other hand, the further processing of the request if the target belongs to XBAC takes time as well, so the accumulated effect cannot be predicted and also depends on the response time of the target node.
- The load can be expected to be slightly higher because two remote hosts that would be in the same Group will initially be in Clusters of their own in our tests, which means that there are measurements and CPU time necessary to merge the two Clusters.

6.3.2 Test Results

We first take a look at the results of test 1, in which we used a Group of three nodes running for 24 hours.

The first important aspect is how often XBAC has enough data available to make a prediction. Fig. 24 shows the development of the ratio of predictions made versus prediction requests for a sample node.

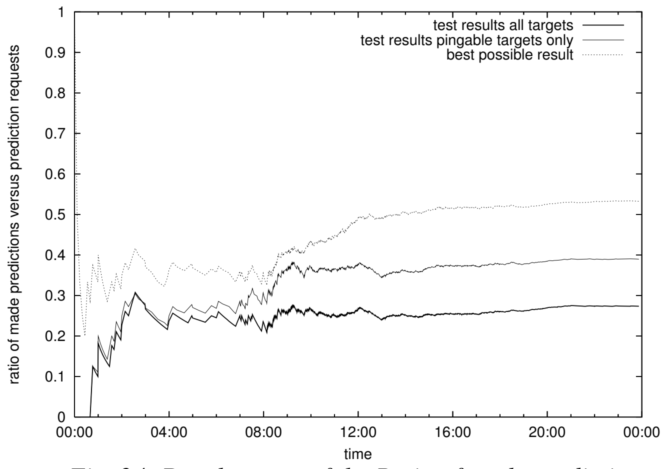

Fig. 24: Development of the Ratio of made predictions versus prediction requests over time.

Initially, there are no targets known, so it is not surprising that the first prediction requests cannot be met. Soon, the rate rises quickly until it levels at about 28% made predictions. A problem we encountered in our test setup was that many targets refused pings.

Fig. 24 shows the rate observed in the test considering all targets. The fact that many targets refused pings meant that the prototype could not make predictions to these targets. Since the desired usage of XBAC does not depend on pings but gathers measurements by observing regular traffic, we eliminated this effect and calculated the same rate when only pingable targets are considered. The development of that rate can be seen in the middle line of Fig. 24. Finally, we replayed the test with a simulation to find out how good the system possibly could be. This simulation depended only on the Url of the targets, if two targets had the same Url the simulation assumed that there was a prediction possible for the second target. There are two issues about domain name resolution which influence that simulation: Firstly, if a target uses a Dns-distribution system, it is possible that resolving the same Url results in two different IP addresses. In this case, XBAC treats the targets as different while our simulation assumes that the targets are equal. This can explain why the test results are clearly worse than the ones in the simulation. The second influence is that multiple Urls can be resolved to the same IP address. In this case, XBAC might be able to make a prediction while the simulation assumes that the targets are different. To eliminate this factor, we counted each successful prediction in the test as a success in the simulation as well. The results of this simulation are shown in the highest rate of Fig. 24.

Apart from the ratio of possible predictions, the quality of predictions is a very important issue. In our test, we have reached exceptionally good results: In test 1, the average relative error was 5.3% while the average absolute error was smaller than 5 milliseconds. In the shorter tests, the results were generally even better. Table 4 shows the absolute and relative errors for the nodes of test 4.

Node	**Average absolute error per node (ms)**	**Average relative error per node (%)**
Node 1	6.7565227	3.825658
Node 2	2.6981647	3.932368
Node 3	0.6323018	3.166891
Node 4	0.6323018	3.166891
Node 5	6.4923935	4.419246
Average all nodes	2.5856657	3.584343

Table 4: Average and relative errors for test 4.

Fig. 25 shows the predictions and the development of errors for node 2 of test 4. We can see that few predictions (between 2 and 3%) have very high relative errors while the other predictions are useful. Some but not all of the bad predictions can be explained with extremely small absolute values, where an absolute error of only 1 millisecond can lead to a high relative error.

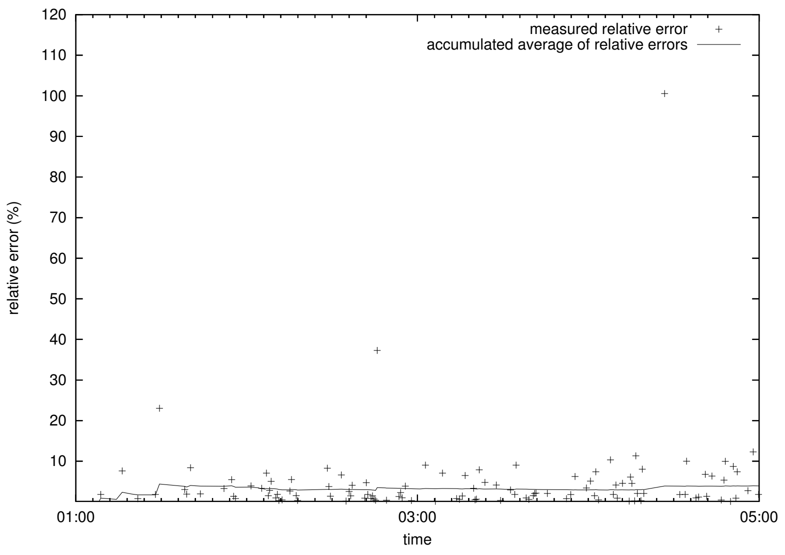

Fig. 25: relative error measured in test 4

However, all these results are based on regular active measurements. We cannot expect to have this kind of data when using only passive measurements, therefore it has to be expected that these results are too optimistic. Still, we have shown that with that data available, XBAC is able to make excellent predictions based on statistical data.

The next issue we are interested in is how high the load on the system induced by XBAC is. Fig. 26 - Fig. 29 show these values for a sample node of test 1. In Fig. 26 we can see that CPU usage stays very low during the whole test and Fig. 27 shows that hard disk usage is moderate as well, with the maximum being less than 20 megabytes. The values are rather high for virtual and physical memory as can be seen in Figs. 26 and 28. However, this could be optimized easily by using a different threading model. Currently, for simplicity each Cluster a node manages is assigned its own thread, which takes up more resources than expected. In Fig. 26, a sudden increase of virtual memory usage from 17% to 31% can be observed. We tried to identify a reason for this increase, but we were not able to identify a reason. Therefore, we have to assume that it originates in a irregular behavior of the Java Virtual Machine on which the program was running.

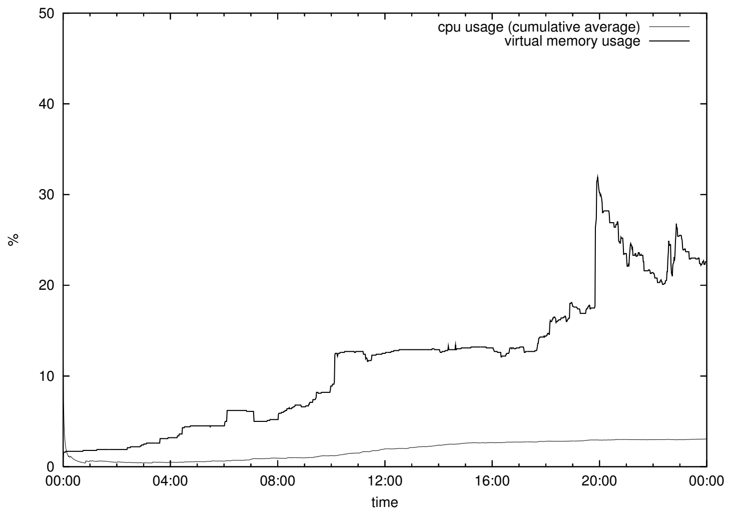

Fig. 26: Development of CPU and memory usage over time

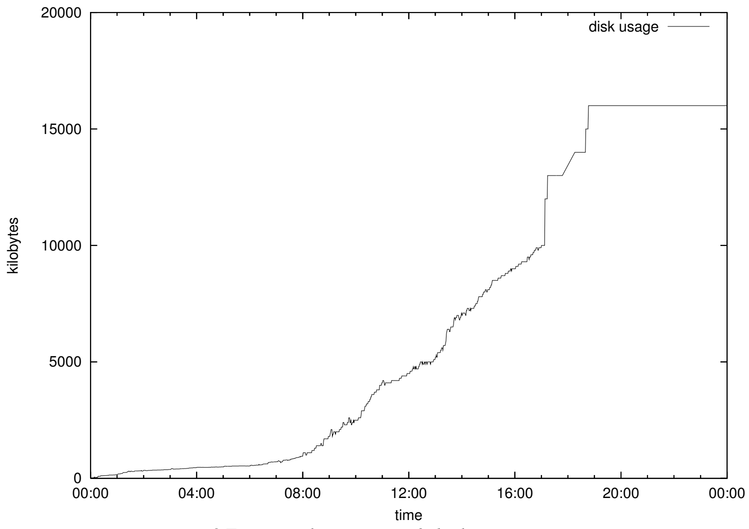

Fig. 27: Development of disk usage over time

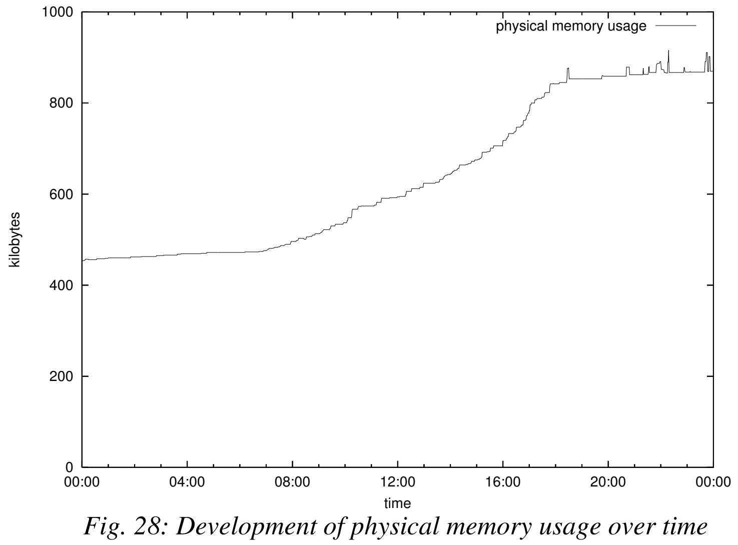

Fig. 28: Development of physical memory usage over time

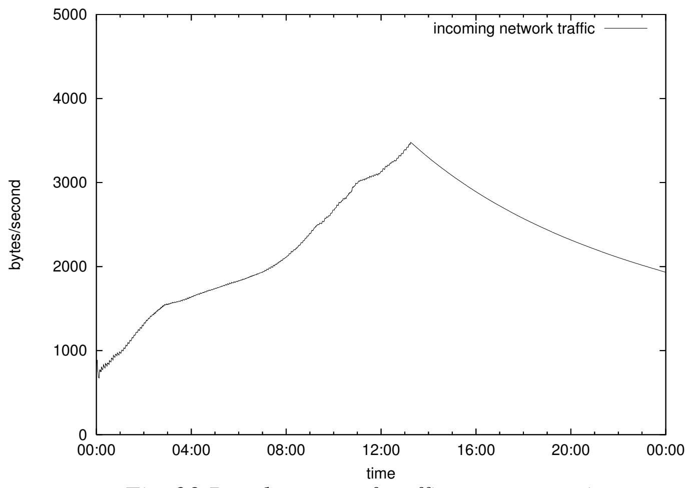

Fig. 29 Development of traffic usage over time

The most critical area is the traffic generated by XBAC. Fig. 29 shows the development of incoming traffic for a sample node, which shows a steep increase until the early afternoon and then decreases again. This behavior is probably caused by the fact that XBAC stores targets for four hours if no requests for them are made. That means that in the early afternoon, the first significant number of targets is discarded. Additionally, the decrease can be explained by the clustering constantly performed by XBAC which reduces the number of stored Clusters.

Since XBAC is intended to be a background application, the observed traffic in test 1 of up to 3.5 kilobytes or 28 kilobits per second is clearly too much. As Fig. 31 shows, traffic even increases in larger Groups. We presumed the broadcasting of information about Clusters to be the main source of this traffic. To assert this presumption, we set up a small Group of four nodes and registered the size of the messages each node received from these broadcasts. With less than 200 Clusters known in the Group, the traffic observed was 2.27 kilobytes per second, which clearly shows that broadcasting Cluster information is the main cause of traffic in our network. In Chapter 7.5, we will show possible solutions how this issue can be improved.

A further effect of this high traffic is that the Pastry network is overloaded. Pastry uses only one socket, which means that if multiple messages are to be sent with Pastry, they have to be sent one after another. If there are more requests than Pastry can handle, this means that message delivery takes a long time. This can be seen in Fig. 30, where we compare the duration to make requests in Groups of different sizes. In the beginning, both the small and the large Group show good results of less than 2 seconds for an average request. After about one hour, the values of the large Group rise significantly and after three hours, predictions in the large Group are essentially useless because it takes 5 seconds and more to make them.

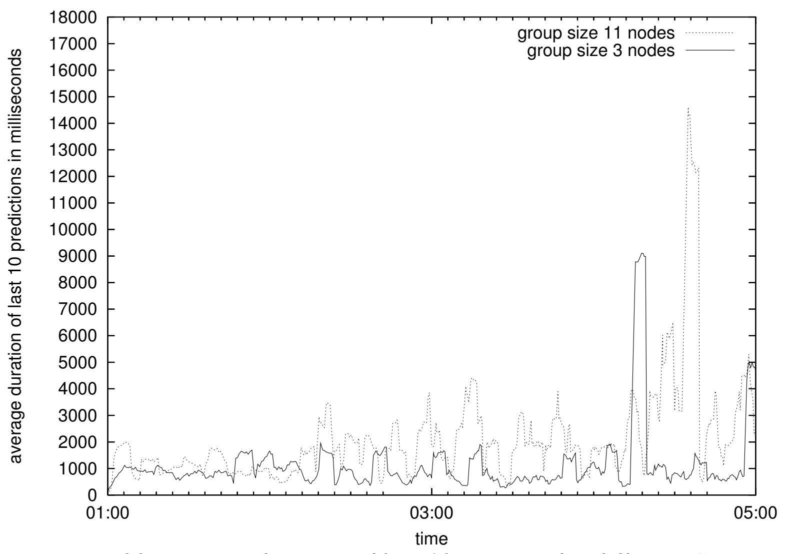

Fig. 30 :average duration of last 10 requests for different Groups

This leads us to the next observations: As Fig. 31 shows, larger Groups generate significantly more traffic than small Groups. This can be attributed again to the traffic generated by broadcasting information about Clusters: The more members a Group has, the more Clusters exist and more traffic is generated. As mentioned, we will propose a solution to this issue in Chapter 7.5.

The final observation is that the overloading of the network also seems to affect the rate of made predictions. For small Groups, the rates rise as the Group size is increased; we can assume that this trend would continue if the traffic generated by XBAC was reduced.

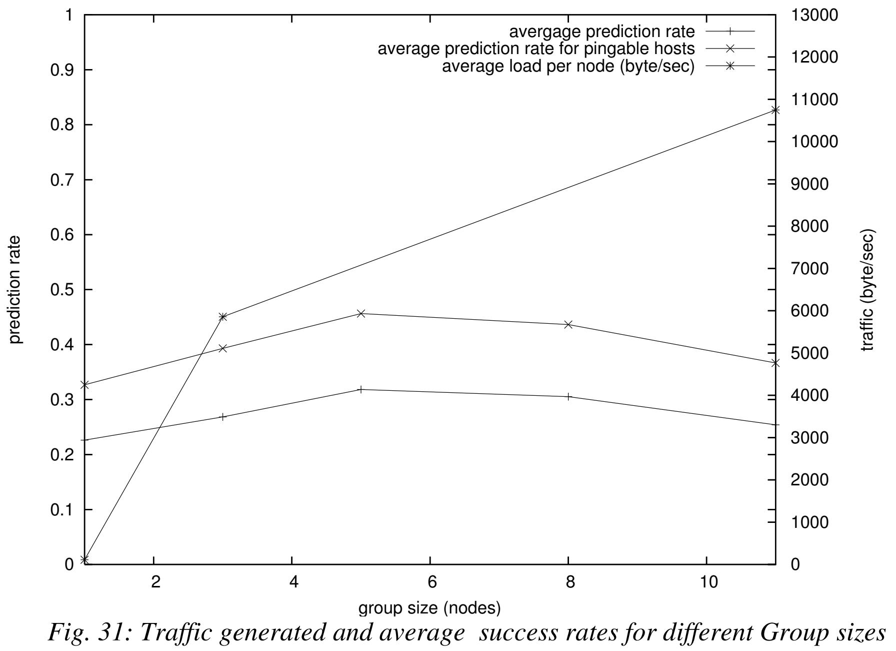

Fig. 31: Traffic generated and average success rates for different Group sizes

6.3.3 Additional Tests

In addition to the tests performed for this thesis, Matthias Scheidegger has performed tests with the same prototype. The results of these tests are described in [25], where he assessed the clustering of remote hosts. In an experiment similar to test 1 described in Chapter 6.3.1, he measured the size of the Clusters that were created in an XBAC Group. Fig. 32 shows that while the majority of hosts were assigned to small Clusters, there were also Clusters containing more than 80 hosts. Compared to storing data to each target separately, the clustering resulted in 62.06% fewer repository entries [25].

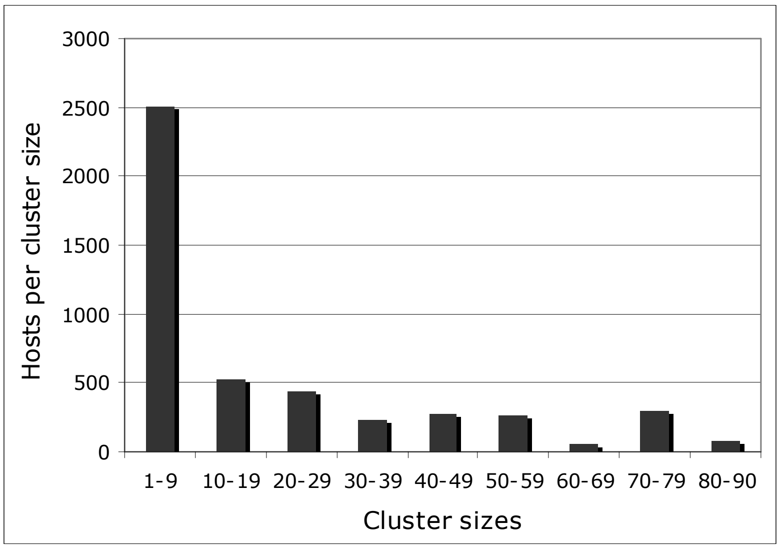

Fig. 32 [25]: The number of nodes assigned to Clusters of a given size

7 Future Work

In this section, we will show how XBAC could be further improved.

7.1 Design Changes

The changes described here are changes which would enhance the performance of XBAC. They would influence the design of XBAC and introduce significant complexity to the architecture.

7.1.1 Exploiting Measurement Similarities across Groups

In Chapter 4.1, we described how a Group collects measurements made by its members and uses them to form remote Clusters. There are two ways in which this concept could be improved even further. First, measurements from Group A to Group B could be also used for the other direction. This should work easily for round-trip time measurements; however, since XBAC is intended to work for other QoS attributes such as bandwidth, it might lead to problems because different routes are chosen for the two directions. The second improvement could be that if multiple Groups find remote Groups or even Clusters toward which they have nearly identical measurements, they could delegate making measurements for that remote Group to a single Group. In Fig. 33 for example, it is likely that the nearby Groups 2 and 3 have similar measurements to Group 1. While this idea seems to be feasible, it would complicate the system significantly which is why we abandoned the idea for the time being.

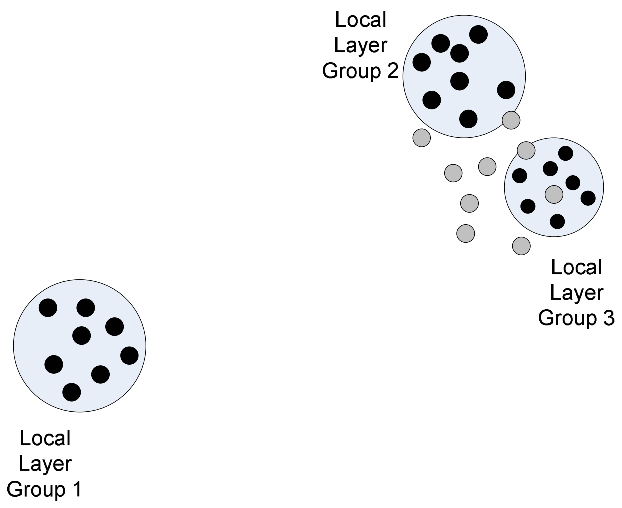

Fig. 33: (as in Chapter 4.1.2)Example network topology. XBAC nodes are shown black.

7.1.2 Updating Group Assignments for Remote Hosts

A weakness of the current XBAC design is that once Group A associates a host with Group B, this remains that way for the time Group A exists. Since this might be for a very long time, it is well possible that the remote host changes Groups in the meantime, either by dis- and reconnecting to XBAC or because of bad predictions (possibly caused by bad initial Group, evolution of the Group or because the host itself moved).

Possible solutions to this problem are to either periodically contact all members of a Group and ask to which Group they belong or to assess the measurements to a Group and only take action when the Group seems to be not well formed anymore. A third option would be to just remove references to Groups after a specified amount of time. While the first option would generate significant traffic, the second one requires computations on the node responsible for the Group's Cluster. The third option would not have an impact the load on the system, but it would often cause useful information to be deleted. For these reasons and because the problem should not significantly affect the performance of XBAC, this issue remains unresolved for the moment.

7.2 Security Improvements for XBAC

In Chapter 5.3, we identified four areas, in which the security of XBAC should be improved. These were:

- Nodes which submit false measurements should be detected
- Nodes which make false predictions should be detected.
- A node should be able to answer only requests it is responsible for.
- Submitting measurements and requesting predictions should be done anonymously.

To address these issues, we formulate here three proposals which solve one or more of the points above.

7.2.1 Proposal 1: Voting / Expert Based Systems

For the first two of these requirements, a voting or expert based system assessing the quality of the data submitted by a node provides a straightforward solution. The measurements for a Cluster are collected on a single node. That node also usually performs active measurements to the remote Hosts and therefore can assess the quality of the measurements. If many measurements made by the same node do not fit into the Cluster, this means that the node should either belong to a different Group or that it is purposely manipulating the measurements. In both cases, the node can simply be forced to leave the Group.

The second case of a node making false predictions seems to be harder to detect since these predictions are partly based on measurements which the node did not perform itself and on Clusters which may not be well formed. Therefore, it is important that the node has the possibility of identifying false measurements before it gets punished for making false predictions. To assess the quality of predictions, the feedback from applications using XBAC is required. If this feedback is repeatedly bad for a Cluster, this is reported together with the node which made the prediction to the Group Leader. He then can decide whether or not the node should be excluded from the Group.

The two solutions just presented have two significant flaws: If numerous organized malicious nodes are part of a Group, they are able to manipulate the voting system. This is especially critical if the Leader node is a malicious node itself. The second problem is that currently, XBAC nodes cannot be reliably identified, meaning that a malicious node can make false predictions pretending to be a different node. While the first problem is a very basic problem in p2p networks and cannot be solved here, the second one is closely linked to the third desirable security improvement that requests can only be answered by the node they are intended for and will be addressed by proposal 2.

7.2.2 Proposal 2: Node Identification Using Public Key Encryption

In many p2p networks, reliably identifying a node could be achieved by using public key encryption for all relevant messages. Because of the way ids are assigned in XBAC, this is not possible. In the following section, we show why this is not possible and how public key encryption can nevertheless be useful in XBAC to provide node identification.

In most p2p networks, it is possible to ensure that only the targeted node can receive a message by encrypting a security token with the receiver's public key and then sending it together with the message. Only the node the message was intended for can decrypt the security token which is then encrypted again with the public key of the node which made the request. If that node receives the reply to its request, it can check if the token is still correct. The main issue with this approach is to find out the receiver's public key. If the id of a node is chosen randomly, it can be derived from a private key and used directly as the public key. Since in XBAC, the id must be derived from the node's IP address, this approach does not work because if the public key (the id) is derivable form the IP address, this is also the case for the private key. To solve this, a new NodeId which is independent from the HostId would have to be introduced. While this seems to be a significant effort, it should not affect the concept of XBAC.

However, there remains a second problem: For many requests, the target node is not known and therefore the Id which is the private key used for encryption is not available. This is for example the case in a request to find the ClusterId for a given HostId. Unless a node in the network knows all members, it cannot calculate which node should receive the message and therefore the requirement that only the correct node can answer requests is simply not achievable.

Still, public-key encryption can be useful in this context. Instead of encrypting requests, the node which answers a request signs the answer message with its private key. If that node which originally made the request can decrypt the method using the answering node's public key (e.g. its node id), the answering node was reliably identified. Thus, malicious nodes are still able to answer requests not intended for them, but they can be identified with a voting-based system as in proposal 1 and later excluded from the Group.

7.2.3 Proposal 3: Anonymous Request

To meet the fourth requirement, anonymous operations, a system similar to Tarzan[16] needs to be implemented. With Tarzan, the sender of a message is obscured by routing the message through a series of proxies, each one only knowing one previous proxy and thus unable to find out whether that proxy was the original sender. However, this would come at a high cost in terms of performance.

Ideally, this functionality should be provided by Pastry, since all message routing is done in Pastry currently. As long as Pastry does not provide anonymous messages, a second peer-to-peer network based on Tarzan would need to be implemented. Pastry could then be layered on top of Tarzan, however, it might make sense to use only Tarzan because it provides the all the functionality of Pastry while the performance advantages of Pastry would be lost anyway.

Therefore, relying on Tarzan instead of Pastry would have the following disadvantages: Routing in Tarzan is significantly less efficient and replies to a request would have to be routed through all the proxies instead of being able to directly contact the requesting node.

7.2.4 Security Conclusions

We have identified four areas where the security of XBAC could be improved. To address these issues, we proposed three security improvements:

- Implementation of a voting system to detect nodes who act maliciously.
- Use of public key encryption to reliably identify the node who answers a request.
- Use of a network such as Tarzan to provide anonymous messages.

Because XBAC is only in prototyping phase and because all of these proposals would take a significant effort to implement, we decided not to implement them at the current stage.

7.3 Additional Features

This section describes two additional features that could be implemented in XBAC: an interface for applications which use XBAC to give a feedback and support for predicting additional distance measures.

7.3.1 Prediction Quality Reports

When an application uses XBAC, it may be able to judge whether the predictions given by XBAC are useful or not. To make sure that XBAC Groups remain well formed if nodes move or network topology changes, it would be useful if applications report their experiances. Currently, a very simple mechanism is implemented: if a specified number of low ratings are reported, the node which received these reports disconnects from its XBAC Group and performs the bootstrapping procedure again. However, since the quality of a prediction is greatly influenced by the node which made the measurements for the target, it can happen that the wrong node leaves the Group. Thus, a more sophisticated algorithm taking this into account and which also registers positive quality report would be desirable.

7.3.2 Provide Predictions for further Quality of Service Attributes

Knowing the round trip time to a remote host is probably the most important QoS attribute at the moment, but applications might be interested in attributes such as bandwidth as well. XBAC is designed to be able to make these predictions; however, especially predicting bandwidth raises a few additional problems. For one, it is almost impossible to make active measurements for bandwidth, which means that data from other applications must not only be available but it must be gathered regularly for it to be statistically analyzed. Another issue is that hosts that can easily be grouped if only latency is considered can behave very differently in terms of bandwidth. This has to be considered in the bootstrapping process and will also affect the performance of XBAC because those two hosts cannot use each other’s data anymore.

7.4 Large Scale Testing

While the tests we performed are appropriate for testing a prototype, the configuration of the system parameters should be evaluated in a large scale test we did not have the resources to perform.

An example for this are the parameters used to decide whether a new node should join a Group or not. If those parameters are chosen so that Groups are joined easily, this leads to large Groups. This in turn means that more data is available, so a prediction is possible more often and fewer active measurements are necessary. Also, the Clusters of a Group can be expected to contain more target hosts, which means that fewer Clusters per member node exist and the overhead for Cluster management is smaller. However, larger Groups also mean that distances between Members are greater, which will have a negative impact on prediction quality. Also, in a bigger Group, messages have to pass through more intermediate nodes, thus the overhead in the Group is increased.

Finding an ideal configuration for these parameters depends on a great variety of influences such as the targets of the requests, the frequency of requests, the stability of the physical network and the churn rate in a Group. Therefore, this kind of experiment could not possibly be performed with the relatively small number of nodes we had available.

7.5 XBAC-Implementation Improvements

Based on the tests we performed, we have identified areas in which XBAC could be improved.

7.5.1 Reducing the Network Traffic Caused by XBAC

We have observed in out tests that XBAC causes a large amount of network traffic. Specifically, we identified the broadcasting of information about Clusters as the main source of traffic.

In a first step, we will describe how traffic in general can be reduced. In XBAC, we use a Pastry version which in turn uses standard Java serialization. Serialization is necessary because Java objects need to be transformed to byte code in order to be transmittable over the Internet. However, it seems that Java serialization is utterly inefficient: If, for example an object of type “Boolean” is serialized, the resulting data requires 47 bytes compared to 1 byte of the primitive Boolean value. When this Boolean object is inserted into an Object of type “Vector”, the result is even worse: Out of the one byte Boolean value, data using 209 bytes was created.

The first improvement would therefore be to simplify message objects as much as possible and to search for ways to diminish the overhead in messages. An example of this would be removing the Vector we used to wrap the load of each message. We did this to be able to handle messages in a simpler way and to provide the same interface as XML-RPC, however, considering the drawbacks of using Vectors in messages, making the effort to simplify this structure would make sense.

The second improvement would obviously be not to use Java Serialization at all. This is facilitated by the fact that the developers of Pastry realized the inefficiency of Java Serialization as well and have implemented the newest version of Pastry using an alternative serialization method. This new version of Pastry is currently in a beta state, which is why it was not yet used in XBAC.

Even though these two measures could significantly reduce the traffic generated by XBAC, they do not solve a very basic issue: Broadcasting information about every Cluster to every node in the Group simply does not scale. When a Group grows, more information has to be sent to more nodes. To solve this problem, Clusters have to be broadcasted less often, however, if they are broadcasted too rarely, this means that Clusters are not merged soon enough which then leads to more Clusters in a Group and to more data to be broadcasted.

To solve this problem, we propose an exponentially decreasing rate at which Cluster data is broadcasted. When a new Cluster is created, chances are that a similar Cluster exists. Therefore, its data needs to be broadcasted a few times in rather quick succession. If these broadcasts do not find a similar Cluster, it is not very likely that this changes, therefore the Cluster data is then broadcasted more and more rarely until it finally is not broadcasted anymore. Remember that data about newly created Clusters is still broadcasted, which then might still lead to a merge with the old Cluster. With this approach, we can reduce the traffic caused by Cluster broadcasts significantly while still ensuring that Clusters are merged when it is useful.

Additionally, it would be possible to introduce a limit for the amount of data a node is allowed to broadcast. This limit could be set by the leader based on the number of nodes in the Group to ensure that the messages necessary for providing the functionality of XBAC are delivered in time.

8 Conclusions

In this document, we have proposed an architecture for XBAC, a novel prediction service for Internet distances. After we stated the goals of our service in Chapter 1, we looked at similar systems and concepts useful for XBAC in Chapter 2. We saw that the existing systems with similar goals had two basic weaknesses: While some systems required significant static infrastructure to be deployed, others that did not have that requirement were able to predict only the distance to hosts that actively take part in the system. Based on the insight gained, we then described the architecture and the basic concepts of XBAC in Chapter 3: Groups of nearby Internet hosts collect their data about remote hosts and thus can make predictions based on statistical data. In Chapter 4, we provided a detailed account of how the hosts in XBAC are organized and how the different operations in XBAC are performed distributedly. We conveyed implementation issues and security considerations in Chapter 5 and then described how we tested the different aspects of XBAC in Chapter 6. We showed that the bootstrapping procedure implemented leads to a high rate of nodes joining the correct Group if the landmark nodes behave consistently and how the system adapts if there is a landmark with unsteady distance measurements. In further tests, we showed that the system is functioning as expected and finally we tested the usefulness of XBAC. Through these tests, we identified areas in which XBAC needs to be improved in order to be deployed on a large scale. However, we also were able to show that even with only a small number of nodes in the network, XBAC is able to predict up to 40% of pingable targets and to make useful predictions for these targets. Finally, in Chapter 7, we proposed a number of improvements which could be implemented in future versions of XBAC.

To sum up, we have showed that XBAC is an architecture which has many advantages: XBAC is easy to deploy because due to its peer-to-peer design, it does need very little static infrastructure. Furthermore, the focus on local Groups implies that the system is able to make useful predictions even if only a small number of hosts are part of XBAC. Because a host communicates mostly with hosts very close to it in network topology, predictions are generally quickly available and they are also accurate since they are based on statistical data. Finally, XBAC is a very flexible architecture which can be extended to predict many different distance measures such as bandwidth or delay jitter.

Appendix

A Test Nodes

A.1 Boostrapping Test Nodes

The following Nodes were used as reference nodes in the bootstrapping test:

Number of reference nodes	**0**	**1**	**3**	**5**	**5***
planetlab-1.cs.princeton.edu			X	X	X
planetlab02.mpi-sws.mpg.de			X	X	X
planetlab2.cs.ubc.ca				X	
planetlab02.cnds.unibe.ch				X	X
planetlab01.ethz.ch		X	X	X	X
planetlab1.lsd.ufcg.edu.br					X

The following nodes were used as test nodes:

planetlab1.millennium.berkeley.edu	planetlab4.millennium.berkeley.edu
planetlab10.millennium.berkeley.edu	planetlab11.millennium.berkeley.edu
planetlab12.millennium.berkeley.edu	planetlab13.millennium.berkeley.edu
planetlab14.millennium.berkeley.edu	planetlab15.millennium.berkeley.edu
planetlab16.millennium.berkeley.edu	

A.2 Function Test Nodes

List of planetlab nodes used for the Function Test

planetlab-1.cs.princeton.edu	planetlab-10.cs.princeton.edu
planetlab-3.cs.princeton.edu	planetlab-4.cs.princeton.edu p
lanetlab-5.cs.princeton.edu	planetlab-6.cs.princeton.edu
planetlab-7.cs.princeton.edu	planetlab01.cnds.unibe.ch
planetlab01.ethz.ch	planetlab01.mpi-sws.mpg.de

planetlab02.mpi-sws.mpg.de	planetlab04.mpi-sws.mpg.de
planetlab05.mpi-sws.mpg.de	planetlab06.mpi-sws.mpg.de
planetlab1.iii.u-tokyo.ac.jp	planetlab2.iii.u-tokyo.ac.jp
planetlab1.lsd.ufcg.edu.br	planetlab2.cs.ubc.ca
planetlab1.nbgisp.com	planetlab3.nbgisp.com
planetlab4.nbgisp.com	planetlab5.nbgisp.com
planetlab6.nbgisp.com	planetlab7.nbgisp.com
planetlab8.nbgisp.com	planetlab1.millennium.berkeley.edu
planetlab10.millennium.berkeley.edu	planetlab11.millennium.berkeley.edu
planetlab12.millennium.berkeley.edu	planetlab13.millennium.berkeley.edu
planetlab14.millennium.berkeley.edu	planetlab15.millennium.berkeley.edu
planetlab16.millennium.berkeley.edu	planetlab2.millennium.berkeley.edu
planetlab3.millennium.berkeley.edu	planetlab4.millennium.berkeley.edu
planetlab6.millennium.berkeley.edu	planetlab7.millennium.berkeley.edu
planet01.hhi.fraunhofer.de	planet02.hhi.fraunhofer.de
plab1.cs.ust.hk	

A.3 Simulation Test Nodes

List of planetlab nodes used for the Test simulating the behavior of an Internet user:

Test number	**1**	**2**	**3**	**4**	**5**	**6**
planetlab1.millennium.berkeley.edu	X	X	X			X
planetlab10.millennium.berkeley.edu	X					X
planetlab11.millennium.berkeley.edu	X					X
planetlab12.millennium.berkeley.edu		X	X		X	X
planetlab13.millennium.berkeley.edu					X	X
planetlab15.millennium.berkeley.edu					X	X
planetlab2.millennium.berkeley.edu				X	X	X
planetlab3.millennium.berkeley.edu				X	X	X

planetlab4.millennium.berkeley.edu		X	X	X	X	X
planetlab6.millennium.berkeley.edu				X	X	X
planetlab7.millennium.berkeley.edu				X	X	X

B User Interface

The following methods are available to applications in the User Interface:

void	addMeasurement	String host, float[] measurements, String measurement-Type	Adds a series of measurements to be used by XBAC
Vector	connect	String address	initiates the bootstrapping procedure to the specified address
AR-Predictor	getRTTPredictor	String address	attemts to make an RTT prediction for the specified address and returns the predictor object.
Vector	measureRTT	String address	Measures the Round Trip Time to the specified address.
Vector	predict	PredSpec spec	Attempts to make a prediction for the given Prediction Scecification
Vector	predictRTT	String address	Predicts the RTT to the specified addess
Vector	reportPrediction-Quality	Double prediction Double realValue	an exeternal application is encouraged to report the Quality of predictions. Warning: This mechanism ist not yet implemented in an intelligent way.

C Configuration Parameters

The following table contains a selection of parameter settings that were used in the tests to simulate the behavior of an Internet user:

Parameter name	**Value**
MeasurementUpdateTick	3600
MeasurementTicksBeforeArchive	3
MinimumTicksForNewFile	1
ModelUpdateTick	3600
ModelUpdateAccelerationFactor	1.66
ClusterInfoTimeStep	60
ClusterInfoDuration	3600
ClusterDetectionTimeStep	60
ClusterDetectionBandwidth	0.02
ClusterDetectionOOBThresh	0.1
ClusterDetectionBiasThresh	0.015
ClusterDetectionRelaxFactor	1.05
DefaultMeasurementInterval	30000
DefaultMeasurementBroadcastInteral	90
DefaultInitialMeasurementDuration	5000
DefaultInitialMeasurementInterval	5000
DefaultInitialMeasurementInterval	5000
AutoStartMeasurementControllers	true
DefaultPredictionStep	60
DefaultModelOrder	8
DefaultPredictionTimeHorizon	60000
DefaultMeridianTimeout	10000
DefaultRPCTimeout	1000
DefaultPastryTimeout	10000
BroadcastTimeout	60000

DefaultBackupInterval	300000
MaxClusterBackupInterval	3600000
DefaultNumberOfBootstrapNeighbours	3
DefaultPastryPort	4001
DefaultXMLRPCPort	3001
DefaultMeridianPort	3758
TimedEntryTimeout	86400000
DefaultLeaderAliveInterval	15000
BootstrapDistanceToLeaderThreshold	20
BootstrapRelativeErrorToNeighborThreshold	0.05
BootstrapAbsoluteErrorToNeighborThreshold	20
BootstrapAbsoluteErrorToNeighborTolerance	5
ClusterBroadcastInterval	300000
ClusterProtectionTime	120000
MergeRequestWaitTime	10000
NumberOfBackupLeaders	3
DefaultClusterTransferTimeout	3600000
CleanupInterval	300000

D Bibliography

[1] M.Scheidegger, “Prediction of Internet Characteristics for Distributed Applications “, PhD Thesis at the University of Bern, February 2007

[2] M.Scheidegger, T. Braun and F. Baumgartner, “Endpoint Cluster Identification for End-to-End Distance Estimation“, *International Conference on Communications*, Istanbul, Turkey, 2006.

[3] P. Francis et al, “IDMaps: A Global Internet Host Distance Estimation Service“, *IEEE/ACM Transactions on Networking*, 9(5):525–540, October 2001.

[4] T. S. Eugene Ng and Hui Zhang, “Predicting Internet Network Distance with Coordinates-Based Approaches”, in *IEEE INFOCOM'02*, New York, USA, June 2002.

[5] F. Dabek, R. Cox, F. Kaashoek and R. Morris, “Vivaldi: A Decentralized Network Coordinate System“, in *ACM SIGCOMM’04*, Portland, USA, 2004.

[6] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems“, *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms,* Heidelberg, Germany, 2001.

[7] P. Druschel and A. Rowstron, “PAST: A large-scale, persistent peer-to-peer storage utility“, in *Eighth Workshop on Hot Topics in Operating Systems*, Schloss Elmau, Germany, May 2001.

[8] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kasshoek, F. Dabek, and H. Balakrishnan., “Chord: A scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003

[9] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, J. D. Kubiatowicz, and A. D. Joseph, “Tapestry: A resilient global-scale overlay for service deployment.“, *IEEE Journal on Selected Areas in Communications,*22(1):41–53, January 2004.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “CAN: A scalable content-addressable network”. In *SIGCOMM’01*, pages 161–172, August 2001.

[11] C. G. Plaxton, R. Rajaraman, and A.W. Richa, “Accessing nearby copies of replicated objects in a distributed environment“, in *Proceedings of the*

ninth annual ACM Symposium on Parallel Algorithms and Architectures, Newport, Rhode Island, USA, pages 311–320, 1997.

[12] D. Karger, E.Lehmann, F. Leighton, M. Levine, D. Lewin and R Panigrahy, “Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web“, In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pp. 654–663, El Paso, TX, 1997

[13] X. Y. Zhang et al, “A construction of locality-aware overlay network: mOverlay and its performance”, *IEEE Journal on Selected Areas in Communications,* 22(1):18–28, January 2004.

[14] B. Wong, A. Slivkins and E. G. Sirer, “Meridian: A Lightweight Network Location Service without Virtual Coordinates”, *ACM SIGCOMM Computer Communication Review,* 35(4):85–96, October 2005

[15] BitTorrent. http://bitconjurer.org/BitTorrent/protocol.html.

[16] M.J. Freedman and R. Morris, “Tarzan: A peer-to-peer anonymizing network layer”, *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Washington, D.C., USA, 2002

[17] N. Daswani, H. Garcia-Molina and B. Yang, “Open Problems in Data-Sharing Peer-to-Peer Systems”, *Proceedings of 9th International Conference on Database Theory (ICDT’03)*, volume 2572 of LNCS, pp. 1–15, Siena, Italy, 2003.

[18] Planetlab. www.planet-lab.org.

[19] L. Peterson, T. Anderson, D. Culler and T. Roscoe, “A Blueprint for Introducing Disruptive Technology into the Internet”, in *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, USA, October 2002.

[20] Open Chord, A Java Implementation of Chord. http://www.lspi.wiai.uni-bamberg.de/dmsg/software/open_chord/

[21] P. J. Brockwell and R. A. Davis, “Introduction to Time Series and Forecasting”, 2nd Edition”, Springer, March 2003.

[22] G. U. Yule, “On a method of investigating periodicities in disturbed series with special reference to Wolfer’s sunspot numbers”, *Philosophical Transactions Royal Society London Ser. A*, 226:267–298, 1927.

[23] Axis. http://ws.apache.org/axis/

[24] SOAP Specification. http://www.w3.org/TR/soap/

[25] M. Scheidegger, B. Zahler and T. Braun, “An Internet Distance Prediction Service based on Passive Measurements”, *IEEE Globecom 2007 General Symposium*, submitted for publication, 2007.