RobustTrav: NAT Optimisation for the RobustCooperation Suite

Christoph Beckmann, Tom Gross, Ferdinand Kastl
Human-Computer Interaction Group
University of Bamberg
96047 Bamberg, Germany
<fistname>.<lastname>(at)uni-bamberg.de

Abstract

Cooperative systems supporting remote teams are based on distributed software architectures. Underneath most of these architectures is the Internet Protocol version 4 (IPv4), which acts as technological backbone. Despite its strengths, the IPv4 also entails challenges for developers of such systems for getting system messages through firewalls without increasing latency and bringing delays for users. In this paper we introduce the RobustCooperation Suite for communication, coordination, and collaboration, as well as its RobustTrav mechanism for effective and efficient NAT traversal.

1 Introduction

The Internet has for decades provided the technological backbone for cooperative systems supporting social interaction among remote teams. In its current form, the Internet Protocol version 4 (IPv4) has limitations, which will be partly solved in the next Internet Protocol version 6 (IPv6). For instance, IPv6 allows direct connections without mapping of the addressing of inbound and outbound packets in the Network Address Translation (NAT) layer. Yet IPv4 is still predominant in all networks. Currently, in IPv4 servers typically have publicly routed addresses, while clients may reside in private networks and are connected to the Internet through routers via NAT [9]. It generally allows client-initiated connections, but prevents inbound connections. This impedes direct client-to-client connections (e.g., for file transfers) and server-initiated push notifications.

Since cooperative systems require flexible communication between servers and clients and among clients, a variety of NAT traversal mechanisms route between networks. For instance, port mapping performed by NAT routers maps specific ports on their public IP address to specific client ports in the private network. The mapping can either be created manually or by an automated method (e.g., using the Universal Plug and Play-based IGD protocol [19]). However, manual modifications of router and firewall configurations reduce convenience and flexibility, and employing automated methods may raise security concerns. Long polling is a method that allows a client to receive notifications from a server [15]. The client initiates a connection that remains idle until the server has a notification for the client. After a notification is sent, the connection is closed and a new one is established immediately. This reduces the communication overhead of polling, where the client is regularly requesting new notifications and receives an empty response if the server has none. However, compared to true push long polling still has a considerable overhead.

In this paper we introduce the RobustCooperation Suite for communication, coordination, and collaboration and its RobustTrav mechanism for effective and efficient NAT traversal. We first describe related work. We then give an architectural overview of the RobustCooperation Suite. We present the RobustTrav NAT traversal mechanism, and the RobustTravOpt optimisation. We conclude with lessons learned.

2 Background and Related Work

In this section we glance at related research on NAT traversal mechanisms, performance of NAT traversal, its measuring, and groupware and awareness support.

2.1 NAT Traversal Mechanisms

NAT traversal mechanisms leverage on the advantages of Network Address Translation (NAT) and improve the connectivity of the nodes. NAT per se allows building private networks with efficient routing of inbound and outbound traffic. It also increases the overall amount for participating network nodes.

NAT traversal mechanisms have been successfully applied in *peer-to-peer systems*. Peers are network nodes that act and exchange data as clients and servers likewise. Each node can relay for exchanging data, participate within the network as a super node and assist in discovering other nodes and network addresses. For instance, a recent evaluation of the prominent peer-to-peer infrastructure Skype [18] has shown that 3 relay nodes are required to guarantee a 99.9% success rate of 60 minutes calls [1]. This high reliability requires considerable bandwidth and processing of the peers.

The *Autonomous NAT Traversal method* [12] allows to address clients behind a NAT system by manipulating ICMP packages directed to the public IP-address of the system. It uses the ICMP ECHO RESPONSE messages for embedding payload data. Since some NAT implementations ensure the correspondence of ICMP ECHO REQUEST messages and ICMP ECHO RESPONSE mes-

sages, the described method cannot reliably send arbitrary messages to clients nor pass firewalls.

The *Universal Plug and Play (UPnP) mechanism* [19] is based on the above mentioned port mapping and allows applications running on clients to open a public port through which it is available for incoming connections. On request the client advertises the public port as communication channel and it listens on the internal port for incoming connections. This mechanism is powerful and allows the seamless integration of NAT environment within distributed systems. However, port mapping to multiple clients behind a NAT router requires that the node outside knows the public IP-address and specific port of each client it wants to address.

The *AJAX-Push mechanism* (also known as Server Push or Comet [15]) was the first wide-spread implementation of the above mentioned long polling mechanism. It extends the polling mechanism, which has been used in early Web applications, for pushing data to clients. Long polling uses asynchronous requests from the Web browser to the server. As the Web browser initiates the request, it can pass its enclosing NAT environment and receive new data from the server as soon as it is available. This mechanism generates considerable overhead compared to true push.

2.2 Performance of NAT Traversal

Literature on the performance of NAT traversal is rare. The *AJAX-Push mechanism* was initially proposed to leave incoming connections open and alive. Increasing incoming asynchronous requests (i.e., each collaborating client uses an own request) cause significant load for the server. Improved implementations (e.g., using jetty [3]) introduced the continuation approach, which handles incoming requests as separate threads on the server and puts them to sleep immediately after receiving the incoming connection. On notification the server wakes up the corresponding connection threads, sends the data as a reply, and closes the connection. Evaluations of the *Autonomous NAT Traversal method* have shown that nearly 50% of the clients could be reached using the ICMP ECHO REQUEST messages, while 70% of the hosts were in NAT environments. A possible solution to reach all network hosts, making some hosts relays for others, would increase the processing and management overhead. While this method might have performance advantages due to fast UDP communication, it is limited in its reliability.

2.3 Measuring NAT Performance

Measuring the performance of NAT traversal mechanisms mainly focuses on the loss of messages passed through NAT environments and the latency of messages passed through the traversal mechanism.

The *NAT/Firewall Traversal Cost Model* [2] proposes performance measurements. They are analysed by measuring the performance of three traversal mechanisms: HTTP, Relaying, and Virtual Private Networks. As the individual mechanisms differ in their performance, the model focuses on measuring latency for the easy quantisation and comparability of results. Latency is not linearly proportional to event size, rather the resulting latency from a publisher of events to a subscriber is the sum of all occurring individual point-to-point latencies. Additionally, for measuring NAT traversal performance, it is challenging to setup a NAT simulation. For obtaining reliable results, the authors simulate a NAT infrastructure by combining virtual machines running NAT services. For our performance measurement and optimisation we also use this approach.

2.4 Groupware and Awareness Support

Several groupware and awareness support systems aim to provide users with innovative concepts for social interaction and mutual information. Yet they face challenges with the underlying network infrastructure.

The NESSIE environment [14] as a groupware and awareness service offers a generic distributed infrastructure for sensing events and notifying actuators on the occurrence of events. Events are sent directly to a central server via a HTTP and CGI API. Events get to actuators either via push or pull. The NESSIE environment requires public addressable clients for pushing events to indicators.

The extended GROOVE groupware [5] provides a client application for cooperation, coordination, and collaboration. Specialised events can be generated and distributed to attached clients. The events are first sent to a central Web service and then distributed to desktop and mobile clients via SOAP messages. While this approach leverages on easy publish and subscribe it also lacks mechanisms to distribute SOAP-messages to clients within NAT environments or behind firewalls.

The Web-based Groupware Service-Oriented Architecture (WGWSOA) [11] provides a middleware for groupware applications using a service-oriented architecture (SOA). The middleware services within this distributed system implement specialised business logic and are announced using the SOA mechanisms. The architecture implements standard services for coordination, cooperation, and communication, which can be shared. Groupware applications built with it benefit form standardised APIs, however the services use pull connections and therefore lack scalability due to frequent polling.

The PPPSpace Media Space environment [7] is an awareness support system. It provides concepts and an implementation for permanent capturing, persistent storing, and parallel processing and distributing awareness information as well as video views to distant spaces. Within the PPPSpace sensors capture events and send them to a central server using XML-RPC [20]. The central server notifies interested actuators for presenting awareness information relying on a persistent XML-RPC connection. While the PPPSpace offers parallel notification of clients, it does not provide mechanisms for reaching clients within a NAT environment.

3 The RobustCooperation Suite

The RobustCooperation Suite supports the social interaction of remote teams and consists of a set of distributed client and server software components. It provides functionality for communication, coordination, and data sharing. A sensor-based event processing captures, stores, and communicates events among all components.

3.1 RobustCooperation Suite Architecture

The RobustCooperation Suite is based on a client-server architecture with clients and a central server (cf. Figure 1).

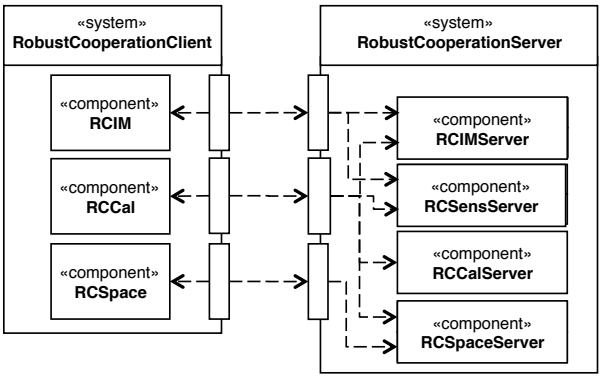

Figure 1. UML component diagram of the Robust-Cooperation Suite.

The *RobustCooperationClient* consists of RCIM, RCCal, and RCSpace).

RCIM uses XMPP [16, 17] to provide users with mutual presence information and online communication. Users can manage a list of contacts and receive their availability and status information. Online users can exchange instant text messages. They can also conveniently transfer files and folders to selected online contacts. Furthermore, RCIM provides advanced concepts for capturing and distributing events about users and their activities and for presenting them to selected online contacts. A variety of sensors for capturing events can be dynamically integrated into RCIM via a plugin mechanism. For privacy protection users have fine-grained control over who receives their events; and for interruptability management users can specify from whom they receive events and at what refresh rate their view updates.

RCCal is a calendaring application that allows sharing calendars among users. RCCal and RCIM use and integrated contact management—changes to the contact list are synchronised between the two applications, sensor notifications are used to coordinate synchronisation of visibility settings and the detail levels of information (calendar events, presence and availability information) that is shared. Calendars are managed with CalDav. RobustCal also has an interface to a shared workspace, where users can manage and share documents organised into workspaces.

RCSpace is a Web-based frontend to the shared workspace. Using the Web-frontend, users can access project documents in a Web-browser, without special client software.

The *RobustCooperationServer* contains four software components. The *RCIMServer* uses XMPP based on Openfire [10], and provides services for instant messaging as well as user authentication and management. The *RCSensServer* is based on the sensor-based platform Sens-ation [8], and aggregates and broadcasts events created in all parts of the suite. The RCSensServer also does the event handling for the RCIMServer. *RCCalServer* is a CalDav [4] server stores and handles calendars and calendar events. *RCSpaceServer* is based on BSCW [13] and stores and handles shared workspaces and documents and manages access rights.

3.2 RobustCooperation Suite and Traversal Challenges

The RobustCooperation Suite provides users with advanced concepts for social interaction, but faces challenges with the underlying network technology. In this section we illustrate typical challenges that arise in the deploy-

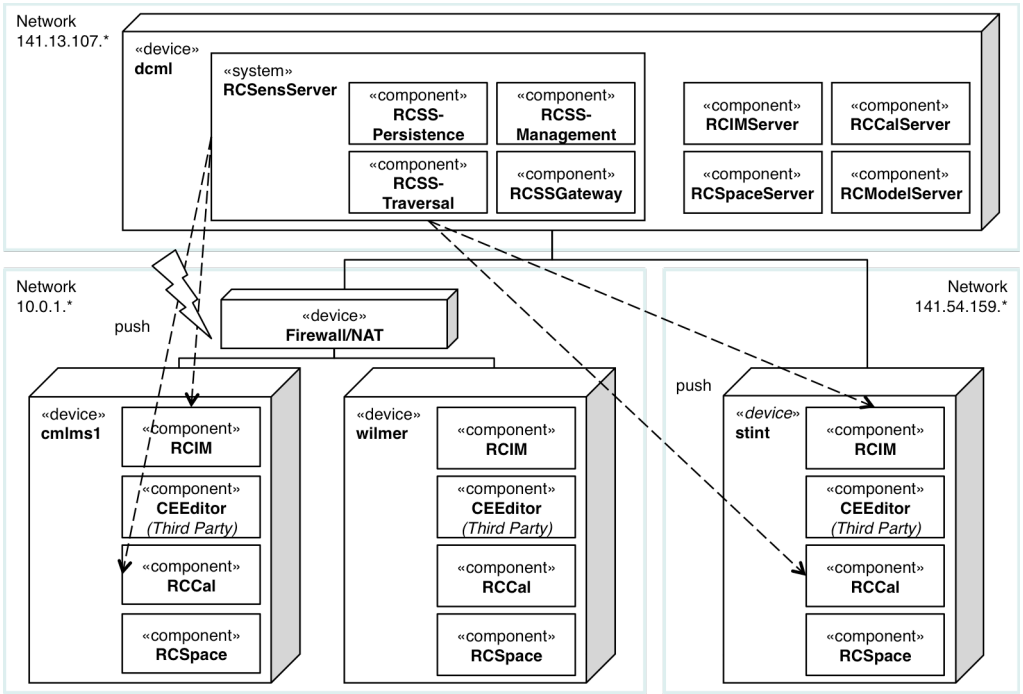

Figure 2. Deployment diagram of the RobustCooperation Suite exemplifying traversal challenges.

ment of the RobustCooperation Suite in a civil engineering deployment case (cf. Figure 2).

In this project the RobustCooperation Suite is used to support an iterative and interactive design process of structural models in distributed teams. The RobustCooperationClients are deployed on client devices with a model editor (CEEditor), which has been developed by civil engineering project partners. The RCSensServer relays events captured in the editor allowing live sharing of changes made in interactive editing session with other users.

One instance of the RobustCooperationServer is running on a server device, the dcml Intel Pentium 4 3.20 GHz, with a direct Internet connection and a publicly routed IP address. It is in the network 141.13.107.*. It consists of five components. The RCSensServer component provides sensor management and event processing services that are used to tie together the functionality of the other server components. It has four subcomponents: RCSSGateway offers an XML-RPC interface for sending control commands and sensor events. RCSSManagement handles the registration of sensors and actuators. RCSSPersistence stores events in a database and a cache allowing state recovery after restarts and analysis of the event history. RCSSTraversal implements the RobustTrav mechanism, which is described below. Further components are the RCIMServer, RCCalServer, RCSpaceServer, and the RCModelServer (the latter stores the structural models created in the editor).

Three instances of the *RobustCooperationClient* are running on distinct physical client devices. Each instance consists of the RCIM, CEEditor, RCCal, and RCSpace client components. The cmlms1 client and the wilmer client devices are a PowerMac G5 Dual 1.8 GHz, and a MacBook Pro 15 Inch 2.4 GHz Intel Core i5 and are in the network 10.0.1.*. The stint client device is a MacBook Pro 15 Inch 2.4 GHz Intel Core 2 Duo in the network 141.54.159.*.

While stint has a public IP address and can connect to the RCSensServer and receive push notifications, the clients cmlms1 and wilmer can only initiate outbound connections to the RCSensServer component (i.e., users can send and receive instant messages, use calendaring and the shared project space). However, cmlms1 and wilmer cannot be reached by the RCSensServer component through push notifications, since they have non-routable IP addresses in network 10.0.1.*. In fact, this network is private and connected to the Internet via an Apple AirPort Express WLAN station that acts as an Internet Gateway Device (IGD) and performs NAT. Without push notifications, advanced features like sharing of live editing sessions do not work (indicated as lightning bolt in Figure 2).

4 The RobustTrav Mechanism

The RobustTrav mechanism connects the server and all clients of the RobustCooperation Suite within and across different network and provides solutions to challenges such as in the deployment above.

4.1 The RobustTrav Process

The RobustTrav mechanism leverages long polling; subsequently we describe a scenario of its processing, where for instance the user makes a change in the personal profile of the RCCal calendar client, which needs to be propagated to the RCIM instant messaging client (cf. Figure 3).

The RCIM component subscribes to the actuator proxy of the RCSSTraversal specifying its interest in the profile change sensor through the RCSSGateway provided by the RCSensServer component. The RCSSTraversal component receives the incoming asynchronous RCIM request, stores it as an individual thread (i.e., `LongPollingActuatorClient extends ActuatorClient`) in an internal `HashTable` and sends the corresponding `LongPollingActuatorClient` to sleep. Sleeping saves processing power and memory on the RCSensServer component. The RCCalProfileChange sensor sends updates to the RCSSManagement component of the RCSensServer, which stores them persistently. Updates matching the RCIM component subscriptions wake up the corresponding `LongPollingActuatorClient`. The RCSSTraversal component collects all current events from the RCSSPersistence and prepares a reply by formatting events as `XML-String` within a `Vector` of the asynchronous request, sends the reply, and closes the connection. The RCSSPersistence component stores all associated events in a dedicated cache for fast delivery and for guaranteeing consistency of events to attached clients.

Also for guaranteeing consistency, the RCIM component can specify the identifier of the last event of the previous notification in order to receive only events since then.

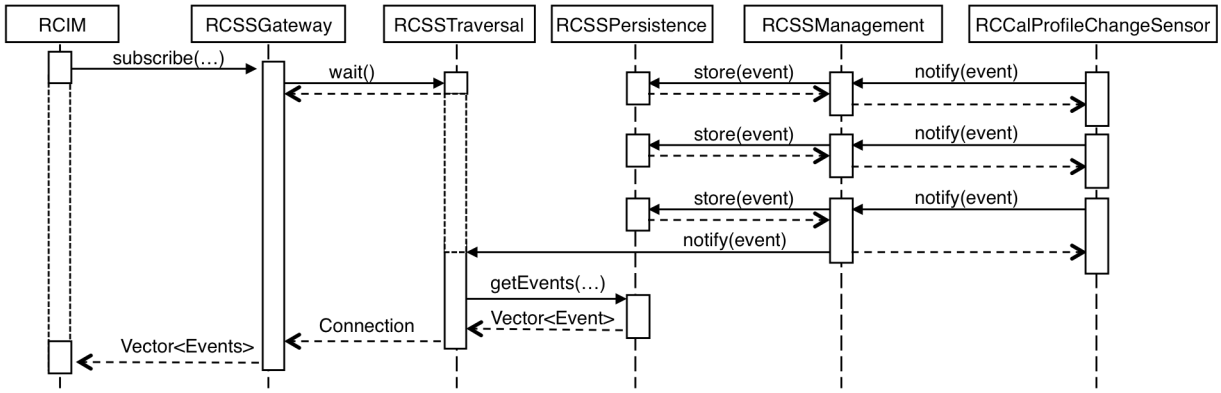

Figure 3. A typical publish/subscribe sequence using the RobustTrav mechanism.

4.2 The RobustTrav Parameters

The RobustTrav mechanism provides runtime and request parameters that influence the communication behaviour between the server and the clients. Based on a thorough literature review and our own experience with developing and deploying event notification systems for over a decade (e.g., [6]) we identified five parameters and corresponding value ranges that are most significant for the quality of the traversal mechanism.

Table 1 summarises the parameters and their value ranges. The server parameters (S) specify the configuration of the server before runtime. The request parameters (R) specify the processing of the respective request in the server and the clients.

Parameter	S	R	Range
Server Timeout	X		0–120,000ms
Client Timeout		X	0–120,000ms
Client Regenerate Time		X	0–500ms
Batch Size		X	1–Cache Size
Batch Timespan		X	0–10,000ms

Table 1. Server and Request parameters.

The *Server Timeout* parameter describes the time the server keeps an incoming client connection open while the corresponding thread is sleeping (i.e., the connection is inactive). The minimum value of 0 means that the server notifies the clients immediately. The maximum time a connection can be sent to sleep is limited to 2 minutes in order to avoid timeouts of the underlying TCP connections (e.g., some firewall implementations close inactive connections after 2 minutes). In general, the shorter the timeout, the more frequent new requests to the server are made.

The *Client Timeout* parameter describes the time the client waits for a reply on an opened server connection before closing it. It has the same minimum, maximum, and general characteristics as the server timeout.

The *Client Regenerate Time* parameter defines the time a clients waits from the point a reply of a previous connection was received until it initiates a new asynchronous request to the server. During this time, for instance, the client can process an incoming batch of events. A minimum value of 0 means the client initiates a new request immediately; the maximum of 500ms guarantees that all events fit into the cache and none are missed.

The *Batch Size* parameter defines the amount of events that the server sends to the client in one bunch. A minimum value of 1 means a bunch size of 1 event. The maximum value is limited through the cache size of the hardware the actuator proxy runs on.

The *Batch Timespan* parameter defines how long the sever stores events in the cache before it sends them in a bunch to the client. A minimum value 0 means, the server sends new events immediately. The maximum value of 10,000ms prevents cache overflow during typical event capturing frequencies.

These five parameters correlate among each other with positive and negative effects. An example of a positive effect is the combination of a moderate timeout value and a low batch size value, which both cause frequent connections guaranteeing open notification channels and immediate notifications on events. An example of a negative effect is a low timeout value and a large batch size value that both cause numerous connections with few new events for the clients.

5 The RobustTravOpt

The optimisation of the overall performance of the RobustTrav mechanism leads to fast delivery of events to clients (i.e., low latency between incoming events and the notification of subscribers), while keeping processing overhead low (i.e., the required amount of processing for receiving, for sending to sleep, and for waking long polling connection threads). A series of performance measurements paved the way to the optimum values for the parameters.

5.1 Setup for the Performance Measurements

The setup is characterised by the hardware, software, and test case arrangements.

The *hardware* setup includes a native network host, a simulated NAT environment, and a virtualised network host. The native network host runs the RCSensServer component of the RobustCooperationServer on a MacBook Pro, Intel Core i5 2.4 GHz with 4 GB of DDR3 RAM with OS X 10.6.7 and Java version 1.6.0_24 installed. The NAT environment is simulated (as recommended by [2]) using Oracle's virtualisation software Virtual Box version 4.0.10. On the virtual network host runs a test scenario virtualised in Ubuntu Linux 10.04 with Java version 1.6.0_20 and 1024 MB RAM assigned. The virtual network host automatically runs and logs all test cases for each parameter.

The *software* setup is based on two applications for the clients: EventGenerator and RobustTravClient. The EventGenerator generates events containing an incremental event number and a sending timestamp in nanoseconds. It pushes them to the RCSensServer using a direct socket connection. It runs on the virtual network host. The RobustTravClient subscribes to the RCSensServer and receives events accordingly. It analyses incoming events with respect to their order based on their event numbers and concerning their latency based on their sending and receiving time stamps. Finally, it logs those events. Running both the EventGenerator and the RobustTravClient within one virtual machine guarantees synchronised timestamps for sending and receiving events.

Each *parameter* is arranged in one test case. Each test case has ten test runs with a specific test value of the respective parameter, in order to have a sufficient amount of data points generated while keeping runtime moderate. For each parameter we chose the ten test values according to their range as described above. Table 2 shows the parameters and their test values. The lower test values are denser, because here short distances contain valuable insights.

A typical event sending frequency is 30ms, as we obtained from analysing events occurred during the development of the RobustCooperation Suite. We choose 10,000 events per test run, in order to ensure a time of 5 minutes per test run. The 5 minutes also allow the occurrence of at least two timeouts within a single test run. 50

RobustTravClient clients are dynamically created, which means 500,000 received events (i.e., each of the 50 clients receives 10,000 events) per test run.

Parameter	Values
Server Timeout	500; 1000; 5000; 10,000; 30,000; 60,000; 75,000; 90,000; 105,000; 120,000 [ms]
Client Timeout	500; 1000; 5000; 10,000; 30,000; 60,000; 75,000; 90,000; 105,000; 120,000 [ms]
Client Regenerate Time	0; 50; 100; 150; 200; 250; 300; 350; 400; 450 [ms]
Batch Size	1; 5; 10; 25; 50; 100; 200; 300; 500; 750 [events]
Batch Timespan	50; 500; 1000; 2000; 3000; 4000; 5000; 7500; 9000; 10,000 [ms]

Table 2. Parameters values used in the measurement.

Consequently, 5 parameters, 10 values per parameters, and 500,000 events per value amount to 25,000,000 events within a total run time of 240 minutes.

5.2 Results of the Parameter Measurement

In order to derive the optimal value for each parameter, we analysed the log files aggregating the minimal, average, and maximal latencies. We identified optimal values by considering local minimal latencies while balancing average and maximal latencies.

In general, the sequence of the events remained consistent over all runs; there was no event loss over all 25,000,000 events; the server's processes generated a CPU load between 90% using one core up to 150% using two cores on the native host. Subsequently, we report on the results of the individual test cases.

For the *Server Timeout* the latencies varied between 1.29ms and 10.7 s. Figure 4 shows the measured latencies. The wide range between minimum and maximum latencies results from the caching behaviour of the RCSensServer.

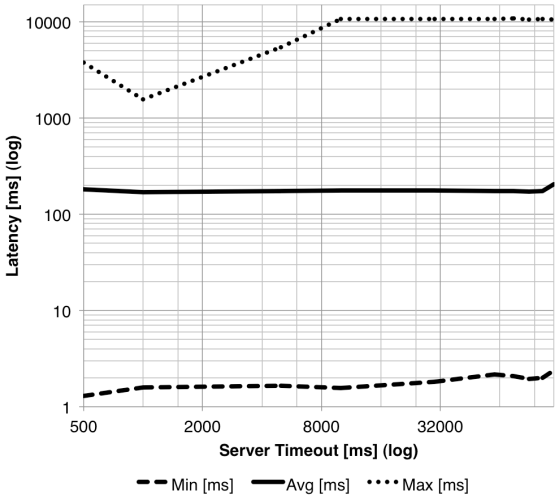

Figure 4. Effect of Server Timeout on latency.

The minimal latency depends on the timespan between two client connects: the longer the time span, the higher the latencies for early events. The maximum latency has a plateau at 10,000ms, which results from the aggregation of the peaks of the 50 clients' processing and scheduling overheads. The average latencies remained nearly constant between 160ms and 190ms. The optimal Server Timeout is 1000ms, due to the local minimum of average and maximum.

For the *Client Timeout* the latencies varied between 1.87ms and 10.7s. Figure 5 shows the measured latencies. The curve shows minimum values and differences to the maximum value that are similar to the server timeout, for the same reasons explained above. The maximal latency values have their plateau at a top boundary of around 10,000ms, and the maximum latency grows rapidly above a client timeout of 5,000ms. Increasing the Client Timeout from 500ms to 1000ms has no significant impact on the minimal (from 2.00ms to 1.88ms), average (from 171.5ms to 169.53ms), or maximal (from 1,459.12ms to 1,825.92ms) latencies. Therefore, the optimal Client Timeout is 1000ms.

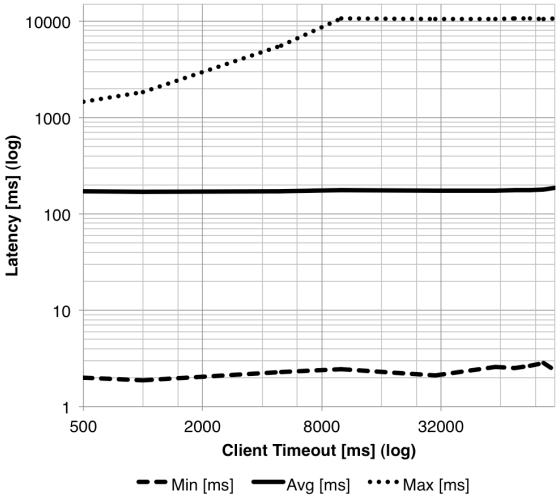

Figure 5. Effect of Client Timeout on latency.

For the *Client Regenerate Time* the latencies varied between 2.86ms and 10.9s. Figure 6 shows the measured latencies. Again we see a wide range between minimum and maximum latencies.

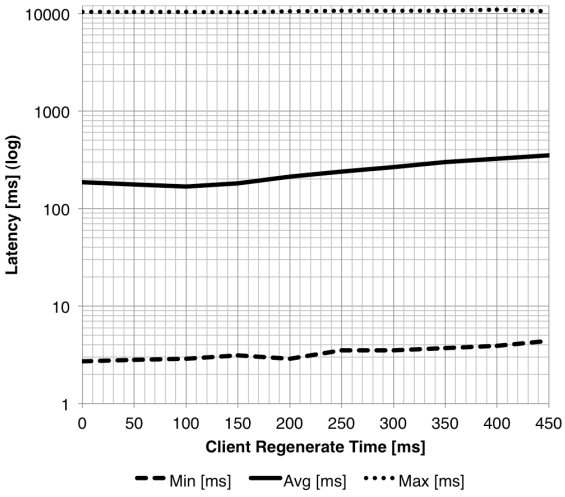

Figure 6. Effect of Client Regenerate Time on latency.

This parameter has no impact on the maximum latencies, because they only depend on the timeout parameters (Client Timeout and Server Timeout are set to 120,000ms). The minimum of the average latency is 100ms; in fact, the average latency is 8% lower than with a time of 0ms. Therefore, the optimal Client Regenerate Time is 100ms.

For the *Batch Size* parameter the latencies varied between 1.67ms and 59.5s. Figure 7 shows the measured latencies. It is obvious that larger batch sizes cause higher latencies as the sending events is delayed until a batch is full or a timeout is reached. This parameter correlates with the average latency with a factor of 20 to 30. There is a wide range between minimal and maximal latencies. The minimal latencies reached up to 115ms. The average latencies are between 169ms and 20s. The maximal latencies do not have a plateau within the measured range. Therefore, the optimal Batch Size is 5 events, at the minimum of the average latency. If an application requires instant notification on a high event frequency (i.e., lower than 30ms) the Batch Size can be reduced to 1 event.

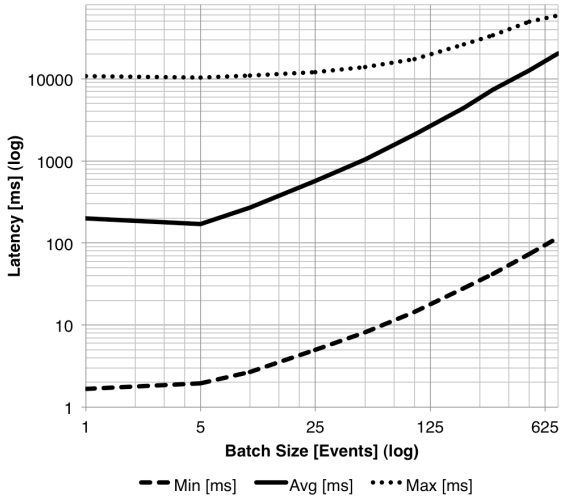

Figure 7. Effect of Batch Size on latency.

For the *Batch Timespan* parameter the latencies varied between 2.1ms and 27.5s. Figure 8 shows the measured latencies.

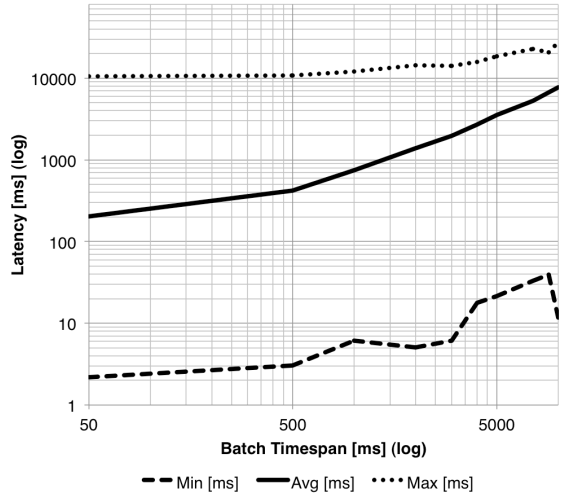

Figure 8. Effect of Batch Timespan on latency.

As with the Batch Size, it is obvious that higher Batch Timespans cause higher latencies: this parameter correlates with latency with a factor of 1.5 (at 30ms event generation frequency). There is a wide range between minimal and maximal latencies. Increasing it has significant impact on the average latency. Although we have an outlier in the minimum latency at 9s Batch Timespan in our measurement, no global minima are visible. Therefore, the optimal Batch Timespan is 0ms. Depending on the application (e.g., less frequently connecting long polling clients, embedded systems), this parameter may have to be set to a higher value.

5.3 The Optimal Configuration of RobustTravOpt

The optimal values for all individual parameters were derived above. In order to confirm that these values also lead to an optimal overall performance of RobustTravOpt, we conducted another test run with all optimal parameter values.

Table 3 shows the optimal configuration of RobustTravOpt—it shows the parameter names and summarises the minimum, average, and maximum latencies of the individual parameter optimisations and in the last row shows the overall performance with the optimal configuration.

Parameter	Min [ms]	Avg [ms]	Max [ms]
Server Timeout (1000ms)	1.29	169.35	1553.72
Client Timeout (1000ms)	1.87	169.53	1825.93
Client Regenerate Time (100ms)	2.88	168.01	10,405.89
Batch Size (5 events)	1.94	169.10	10,349.52
Batch Timespan (0ms)	2.19	203.30	10,523.80
Overall (all parameters values optimised as above)	1.66	167.50	2,677.29

Table 3. Parameters and measured latencies.

As can be seen the overall minimal latency of 1.66ms is close to the best minimal latency of 1.29ms. The overall average latency of 167.50ms is even better than the best average latency of 168.01ms. And the overall maximum latency of 2,677.29ms is a decent value, especially compared to the three high latencies of more than 10,000ms. So, overall the performance of the optimal configuration is fast and efficient in its minimum and average, and balancing and stable in its maximum.

6 Conclusions

In this paper we motivated the need for effective and efficient NAT traversal. We introduced RobustTrav, a long polling mechanism implemented for XML-RPC and demonstrated its feasibility. We identified the five most significant parameters for the performance of RobustTrav and introduced the optimisation of the latency of event delivery RobustTravOpt.

We learned important lessons on the arrangement of measurements for NAT traversal mechanisms. Using a virtual machine to simulate a NAT environment elimi-

nates unwanted delays and interference caused by a physical network connection while at the same time not behaving like a simple loopback device. For measuring latency, it is important that synchronised clocks create timestamps for the generating and receiving events. In order to reach sufficient accuracy (i.e., in the range of nanoseconds) generating and receiving events should run on the same computer. Pushing generated events to the server should create a minimal overhead, which is why we decided to use raw sockets rather than XML-RPC.

Compared to our earlier evaluation [7] of non-NAT-traversing push notifications an increasing amount of clients caused no significant load on the server and the processing overhead.

Furthermore, we introduced the RobustCooperation Suite for supporting end-users in distributed teams. The RCIM application allows end-users to manage their online contacts and exchange instant text messages, as well as share files and workspaces. The RCCal application lets end-users coordinate their tasks—they can create calendars and calendar events, and share them with other users. The RCSpace application provides means for organising and sharing project files.

End-users benefit from the RobustTrav mechanism as they are working in their client application, which can span inhomogeneous networks across different NAT environments without any effort to modify the routing and firewall configuration for receiving events from the central server. At the same time they profit from a balanced average performance as shown in RobustTravOpt.

Additionally to the performance measurement conducted in a lab setting, we are looking forward to analysing how the measured average latencies are met in inhomogeneous real world network spanning across different NAT environments and how our optimal configuration generalises to other network topologies such as rings or trees.

Acknowledgments

The work on the first user-centred prototyping of the concepts and systems was done in the project TransKoop, partly funded by the Federal Ministry of Transport, Building, and Urban Affairs and by the Project Management Juelich (TransKoop FKZ 03WWTH018); and the work on the distributed architecture and its performance optimisation was done in the project Robust Design of Structures, partly funded by the Thuringian Programme on Excellency (ProExzellenz-Projekt B514-09052). We thank all members of the Cooperative Media Lab as well as the project partners of the two projects.

References

[1] Baset, S.A. and Schulzrinne, H. Reliability and Relay Selection in Peer-to-Peer Communication Systems. In Proc. of the Conference on Principles, Systems, and Applications of IP Telecommunications Conference - IPTComm 2010. ACM, N.Y., 2010. pp. 111-121.

[2] Biswas, D., Bean, K. and Kerschbaum, F. NAT/Firewall Traversal Cost Model for Publish-Subscribe Systems. In Proc. of the Second Joint WOSP/SIPEW International Conference on Performance Engineering - ICPE 2011. ACM, N.Y., 2011. pp. 487-492.

[3] Codehaus Foundation. jetty - Jetty WebServer. http://jetty.codehaus.org/jetty/, 2011. (Accessed 20/7/2011).

[4] Daboo, C., Desruisseaux, B. and Dusseault, L. Calendaring Extensions to WebDAV (CalDAV) The Internet Engineering Task Force, http://www.ietf.org/rfc/rfc4791.txt, 2007. (Accessed 20/7/2011).

[5] Dustdar, S., Gall, H. and Schmidt, R. Web Services for Groupware in Distributed and Mobile Collaboration. In Proc. of the Twelfth Euromicro Conference on Parallel, Distributed, and Network-Based Processing - PDP 2004. IEEE Computer Society Press, Los Alamitos, 2004. pp. 241-248.

[6] Gross, T. and Beckmann, C. Advanced Publish and Subscribe for Distributed Sensor-Based Infrastructures: The CoLocScribe Cooperative Media Space. In Proc. of the Seventeenth Euromicro Conference on Parallel, Distributed, and Network-Based Processing - PDP 2009. IEEE Computer Society Press, Los Alamitos, 2009. pp. 333-340.

[7] Gross, T., Beckmann, C. and Schirmer, M. The PPPSpace: Innovative Concepts for Permanent Capturing, Persistent Storing, and Parallel Processing and Distributing Events. In Proc. of the Eighteenth Euromicro Conference on Parallel, Distributed, and Network-Based Processing - PDP 2010. IEEE Computer Society Press, Los Alamitos, 2010. pp. 359-366.

[8] Gross, T., Egla, T. and Marquardt, N. Sens-ation: A Service-Oriented Platform for Developing Sensor-Based Infrastructures. International Journal of Internet Protocol Technology (IJIPT) 1, 3 (2006). pp. 159-167.

[9] Huston, G. Anatomy: A Look Inside Network Address Translators. The Internet Protocol Journal 7, 3 (Sept. 2004). pp. 2-32.

[10] Ignite Realtime. Ignite Realtime: Openfire Server. http://www.igniterealtime.org/projects/openfire/, 2011. (Accessed 20/7/2011).

[11] Maciel, R.S.P., David, J.M.N., Oei, M.R., Bastos, A.A.d.O. and Menezes, L.d.O. Supporting Awareness in Groupware Through an Aspect-Oriented Middleware Service. Journal of Universal Computer Science 15, 9 (May 2009). pp. 1945-1969.

[12] Mueller, A., Evans, N., Grothoff, C. and Kamkar, S. Autonomous NAT Traversal. In Proc. of the Tenth International Conference on Peer-to-Peer Computing - P2P. IEEE Computer Society Press, Los Alamitos, 2010. pp. 1-4.

[13] OrbiTeam. OrbiTeam | BSCW : BSCW Groupware for Efficient Team Collaboration and Document Management. OrbiTeam Software GmbH & Co. KG, http://www.bscw.de/english/index.html, 2011. (Accessed 20/7/2011).

[14] Prinz, W. NESSIE: An Awareness Environment for Cooperative Settings. In Proc. of the Sixth European Conference on Computer-Supported Cooperative Work - ECSCW'99. Kluwer Academic Publishers, Dordrecht, 1999. pp. 391-410.

[15] Russel, A. Comet: Low Latency Data for the Browser. http://www.infrequently.org/2006/03/comet-low-latency-data-for-the-browser/, 2006. (Accessed 26/10/2011).

[16] Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Core - RFC 6120. The Internet Engineering Task Force, http://www.ietf.org/rfc/rfc6120.txt, 2011. (Accessed 20/7/2011).

[17] Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence - RFC 6121. The Internet Engineering Task Force, http://www.ietf.org/rfc/rfc6121.txt, 2011. (Accessed 20/7/2011).

[18] Skype. Free Skype Calls and Cheap Calls to Phones - Skype. http://www.skype.com/, 2011. (Accessed 20/7/2011).

[19] UPnP Forum. Device Control Protocols - UPnP Forum. http://www.upnp.org/sdcps-and-certification/standards/sdcps/, 2011. (Accessed 26/10/2011).

[20] Winer, D. XML-RPC Specification. http://www.xmlrpc.com/spec, 1999. (Accessed 20/7/2011).