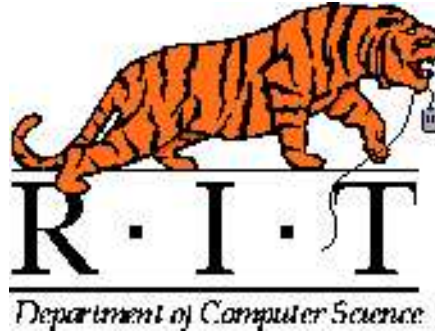


1. Many-to-Many Aufrufe:

Ein objekt orientiertes Paradigma für ein Ad-Hoc/Server-Loses Netzwerk



Hans-Peter Bischof  
hpb@cs.rit.edu

Alan Kaminsky  
ark@cs.rit.edu

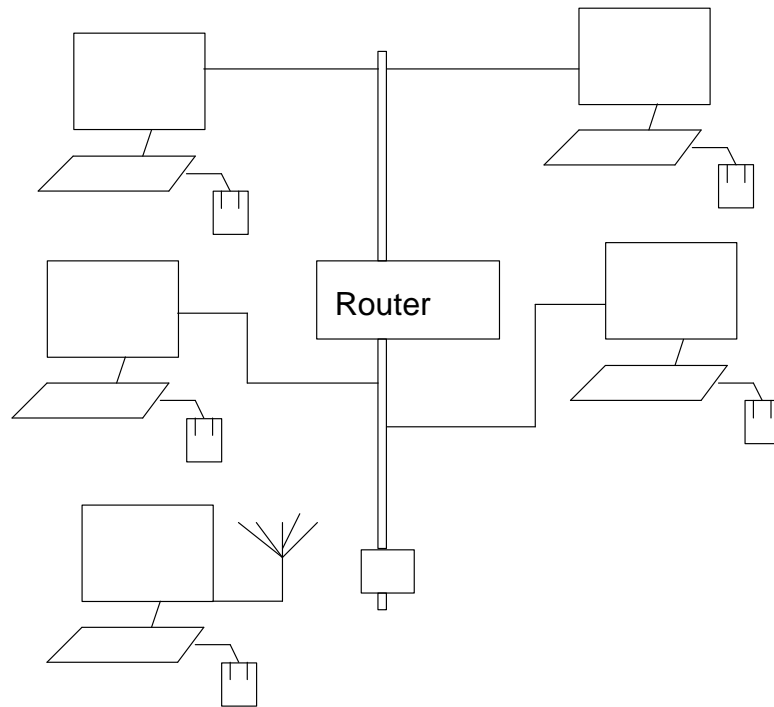
°RIT

Eine Kopie des Vortrags kann hier gefunden werden:

°[http://www.cs.rit.edu/~hpb/Talks/M2mic\\_OS/index.html](http://www.cs.rit.edu/~hpb/Talks/M2mic_OS/index.html)

## 2. Aktuelle Infrastruktur

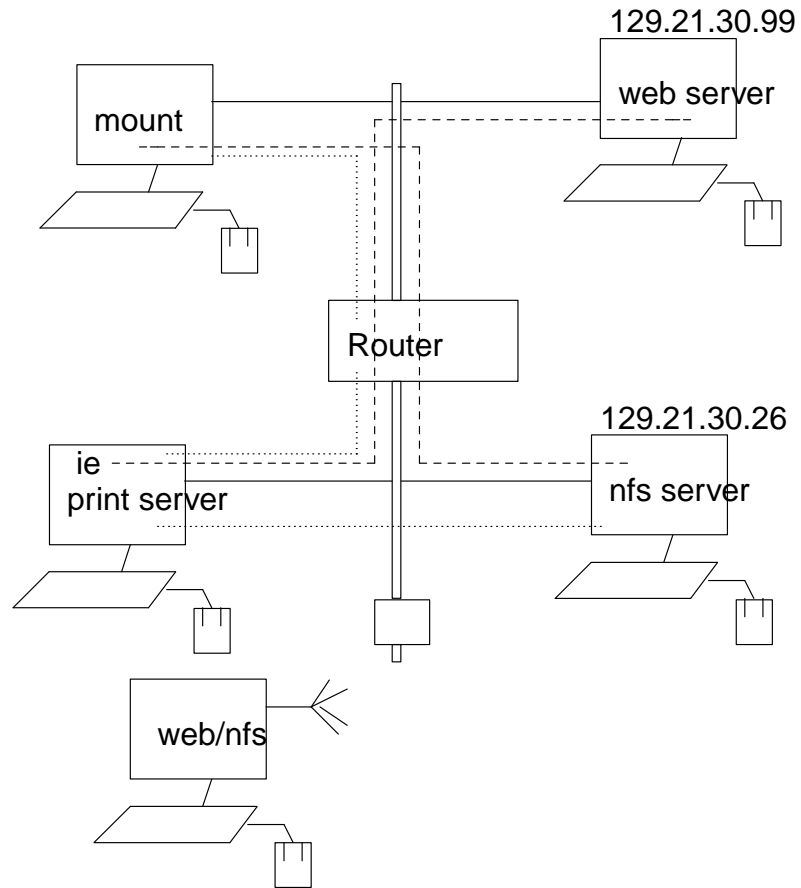
### 2.1. Aktuelle Hardware Infrastruktur



## 2.2. Verwaltung

- ip Adressen
- Router
- Router Tabellen
- ... und mehr

### 2.3. Aktuelle Software Infrastruktur



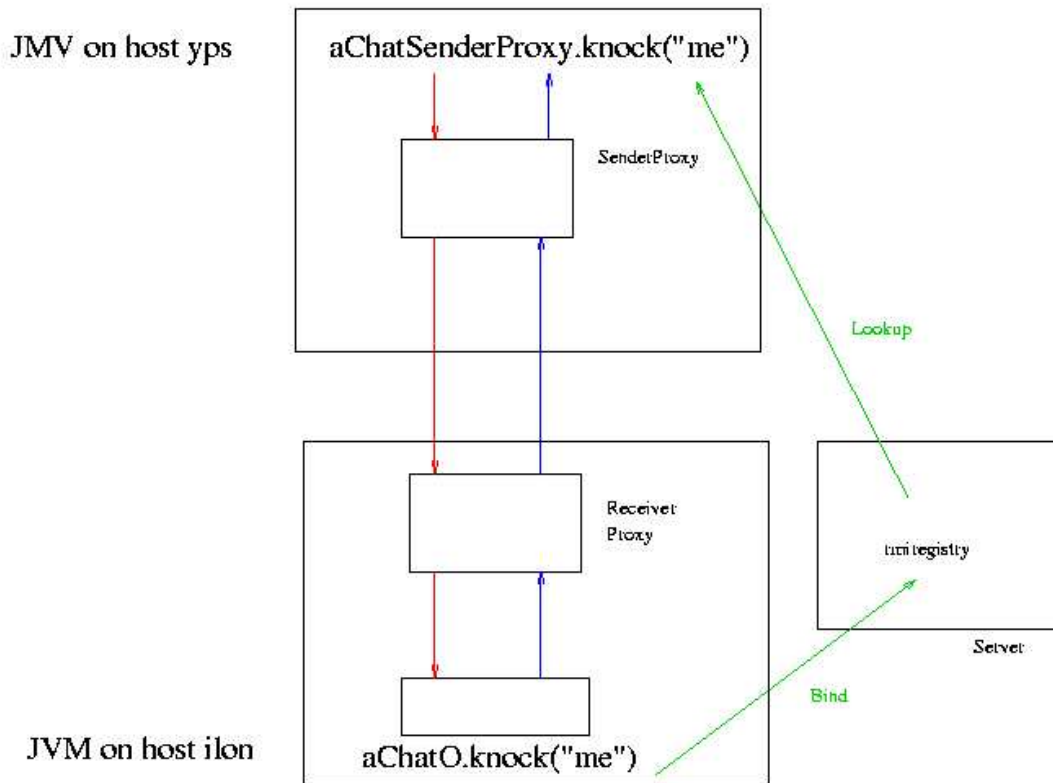
#### 2.4. Typische Kommunikations Muster

- eins zu eins: ip Adressen und Ports
- eins zu vielen: broadcast/multicast
- viele zu vielen: multicast und channels

## 2.5. Typische Kommunikationswege

- Senden von Daten
- Aufruf von Methoden

## 2.6. Remote Method Invocation: Die Idea



## 2.7. SenderProxySource Code

```
....
    private static final long INTERFACE_ID = 32;
    private final long String__METHOD_1 = 0;

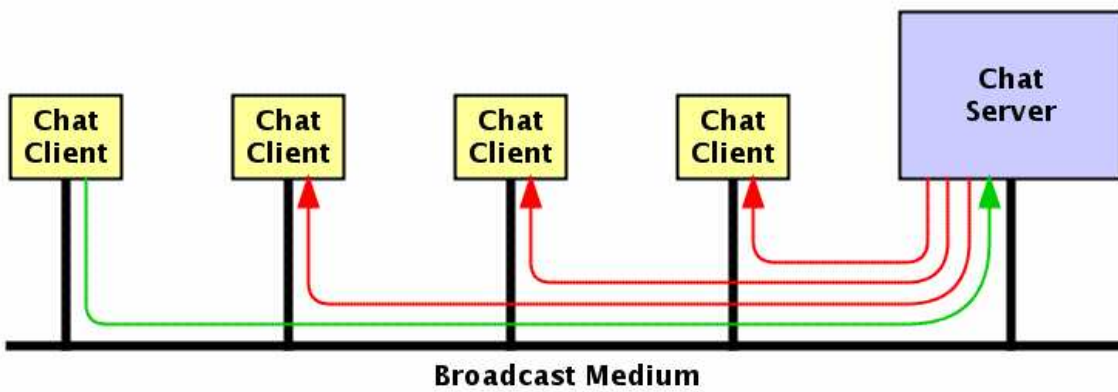
...
public void knock( String _0) {
    try {
        ByteArrayOutputStream aS = new ByteArrayOutputStream(1024);
        ObjectOutputStream p = new ObjectOutputStream(aS);
        p.writeObject(_0);
        p.close();
        RmiPacket aRmiPacket = new RmiPacket( INTERFACE_ID,
                                                String__METHOD_1,
                                                aS.toByteArray() );
        netWorkCommEndPoint.sendDataToMany(aRmiPacket);
        e.printStackTrace();
        System.exit(1);
    }
}
```

## 2.8. ReceiverProxySource Code

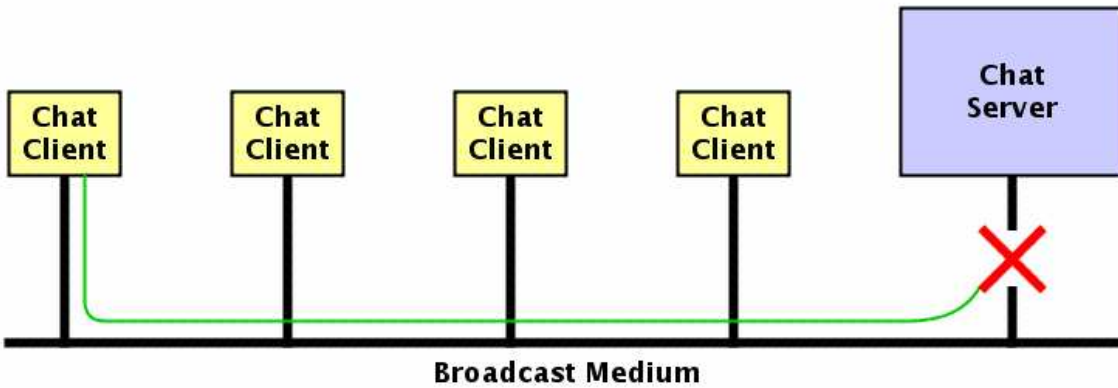
```
....  
private static final long INTERFACE_ID = 32;  
private final long String__METHOD_1 = 0;  
  
aRmiPacket = netWorkCommEndPoint.getData();  
ByteArrayInputStream iS = null;  
ObjectInputStream ip = null;  
iS = new ByteArrayInputStream(aRmiPacket);  
ip = new ObjectInputStream(iS);  
  
if ( aRmiPacket.getMethodId() == String__METHOD_1) {  
    String __0 = (String)ip.readObject();  
    aThisClass.knock(__0);  
}  
...
```

- rmic generiert die Sender und Receiver Proxys

## 2.9. Eine typische Klienten-Server-Architektur



## 2.10. Eine typisches Klienten-Server-Architektur Problem



- Falls der Server ausfällt, funktioniert die Applikation nicht mehr, auch wenn die Klienten direkt kommunizieren könnten.
- Konsequenz: Keine zentralen Server

### 3. Eine etwas andere Art von Infrastruktur

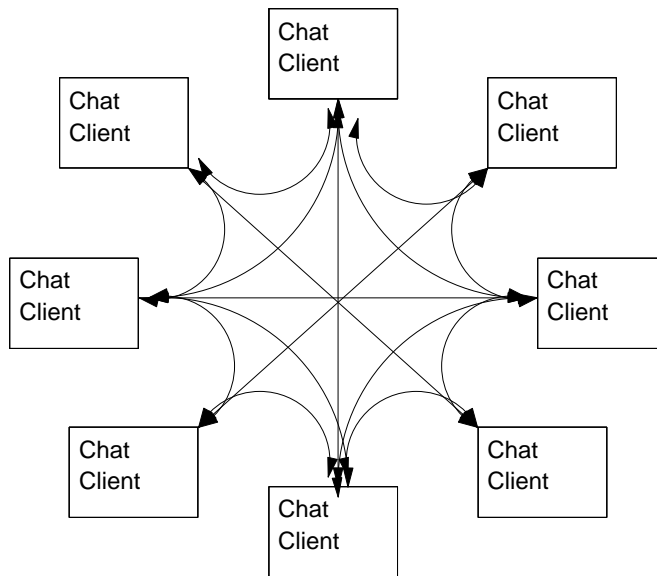
#### 3.1. Zukünftige Hardware Infrastruktur



### 3.2. Eine etwas andere Umgebung: Ad-hoc Netzwerk

- Das Theaterstück ist nicht gerade berauschend
- Meetings: Zeit um Backgammon/Poker/Chatten
- NFL Superbowl XXXVIII

### 3.3. Eine Typische Architektur ohne Server



### 3.4. Eine andere Art von Software Infrastruktur

- Programmieren zu vereinfachen
  - Objekt orientierte abstraktion einer viele-zu-vielen Kommunikation
  - Broadcast Methoden Aufruf: M2MI
- Vereinfachung der Verbreitung der Software
  - Keine Proxy Compiler, Codebase Server, aktive server ...
  - Automatische Proxy Generierung
- Vereinfachung von Administration
  - Keine Netzwerkadressen, kein ad-hoc Router Protokolle

4. M2MI: Ein neues Paradigma für Ad Hoc Kollaborative Systeme
- M2MI stellt einen objekt orientierten Methodenaufruf auf der Basis von Broadcasting zur Verfügung.
  - M2MI-basierte Systeme brauchen keinen Server
  - M2MI-basierte Systeme bedarfren keiner Netzwerk Administration
  - M2MI ist sehr gut geeignet für ad-hoc Netzwerke

#### 4.1. Handles

- Omnihandle: verweist zu allen Objekte die ein Interface implementieren
- Multihandle: verweist zu einer Gruppe von Objekte die ein Interface implementieren
- verweist zu einem Objekte das ein Interface implementieren

## 4.2. Nutzung von Omnihandles

- Exportieren entfernte Objekte

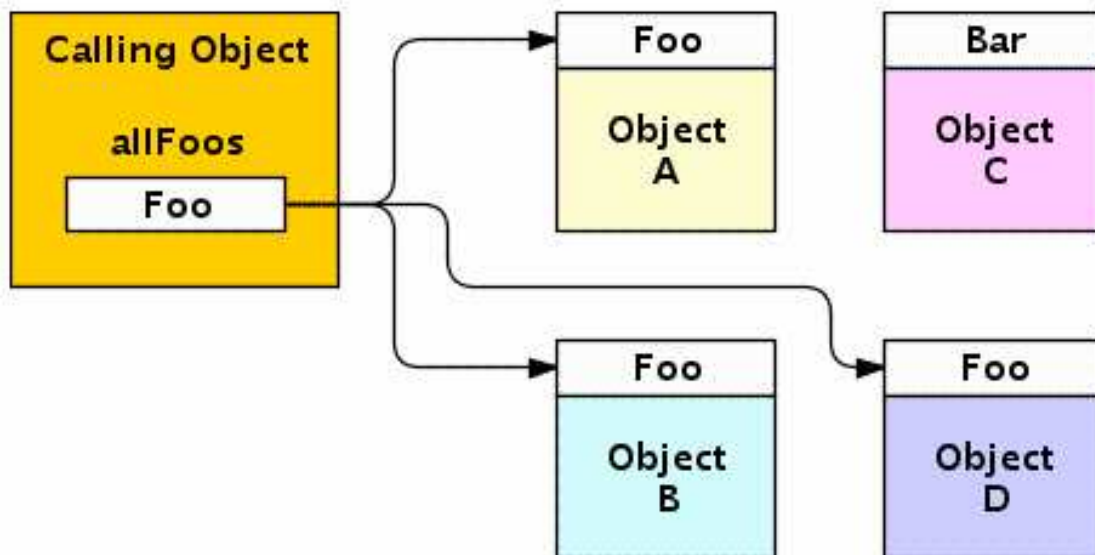
```
M2MI.export (a, Foo.class);
```

- Besorgen einer omnihandle

```
Foo allFoos = (Foo) M2MI.getOmnihandle  
(Foo.class);
```

- Aufruf einer Methode

```
allFoos.y();
```



### 4.3. Nutzung von Multihandles

- Besorgen einer multihandle

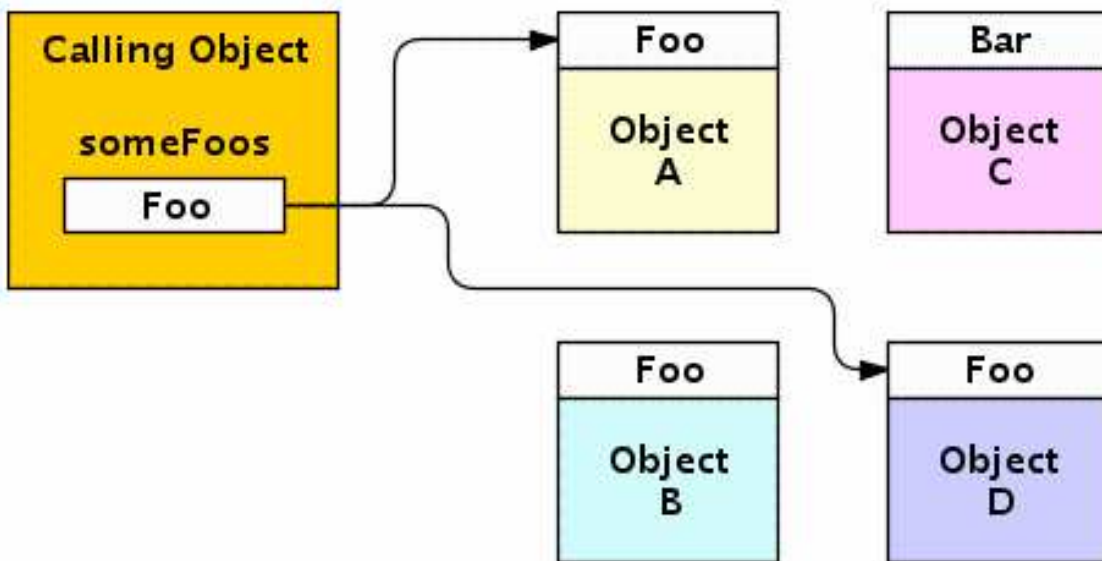
```
Foo someFoos = (Foo)M2MI.getMultihandle(Foo.class);
```

- Ein Objekt zu einer Gruppe hinzufügen

```
someFoos.attach(a);
```

- Aufruf einer Methode

```
someFoos.y();
```



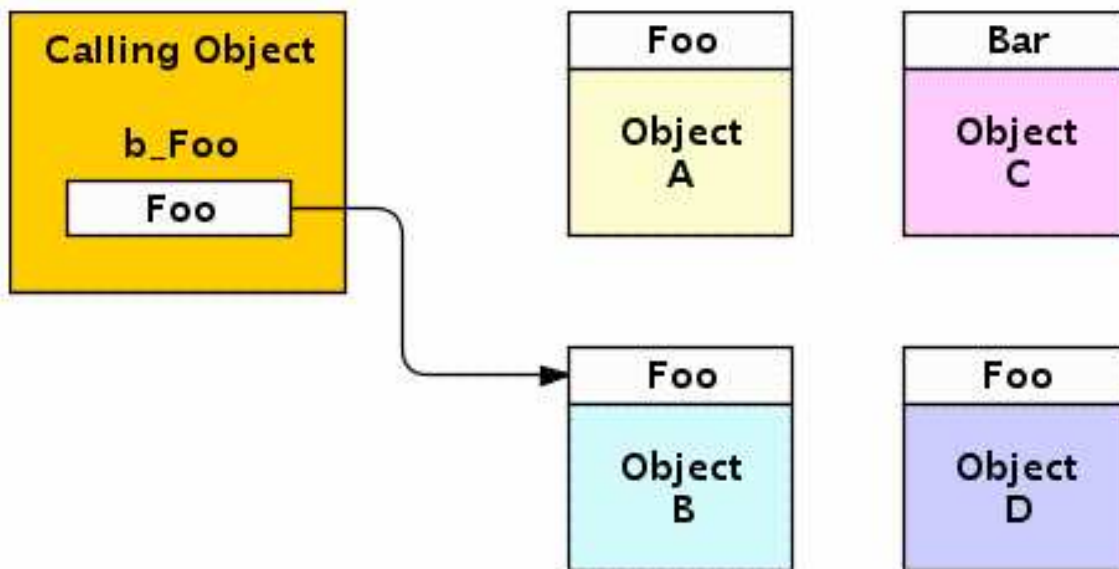
#### 4.4. Nutzung von Unihandles

- Exportieren eines Objekts und besorgen einer unihandle

```
Foo b_Foo = (Foo)M2MI.getUnihandle(b,  
Foo.class);
```

- Aufruf einer Methode

```
b_Foo.y();
```

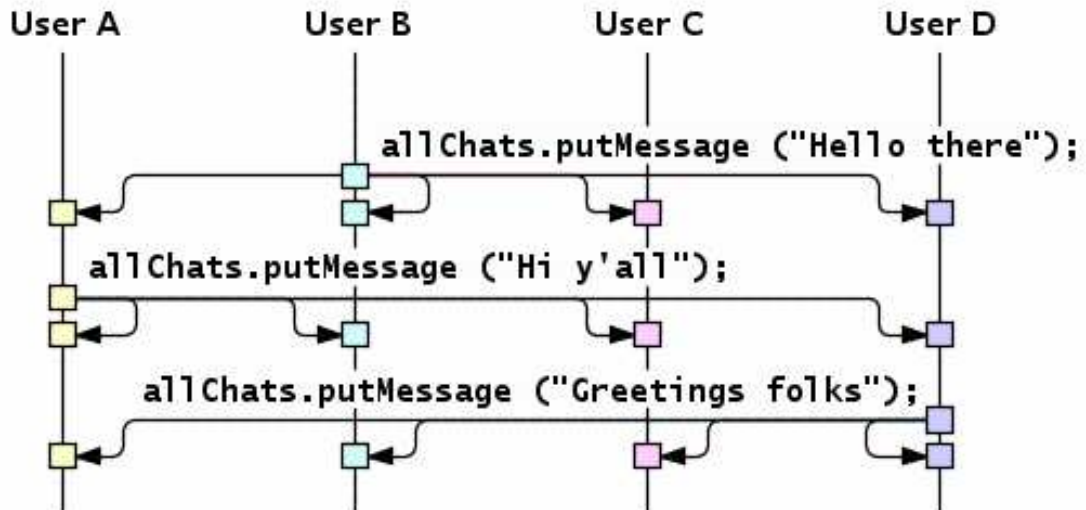


#### 4.5. Eigenschaften von M2MI Aufrufen

- M2MI ist eine objekt orientierte Abstraktion einer viele-zu-vielen Kommunikation
- Semantik von M2MI:
  - Methoden können Argumente haben
  - Objekte werden als Kopie übergeben (object serialization)
  - Handles werden als Reference übergeben
  - M2mi Methoden haben keinen Return Wert
  - M2mi Methoden können keinen Exception werfen
  - Method Aufrufe sind nicht blockierend

## 5. M2MI basierte Applikation

### 5.1. Die Idee für eine Chat Applikation



## 5.2. Das Chat Interface

```
public interface Chat {  
    public void putMessage(String line);  
}
```

### 5.3. Chat Objekt Code

```
public class ChatObject implements Chat {
    private String myUserName;
    private Chat allChats;

    public ChatObject (String theUserName) {
        myUserName = theUserName;
        M2MI.export(this, Chat.class);
        allChats = (Chat)M2MI.getOmnihandle(Chat.class);
    }
    public void send(String line) {
        allChats.putMessage(myUserName + "> " + line);
    }

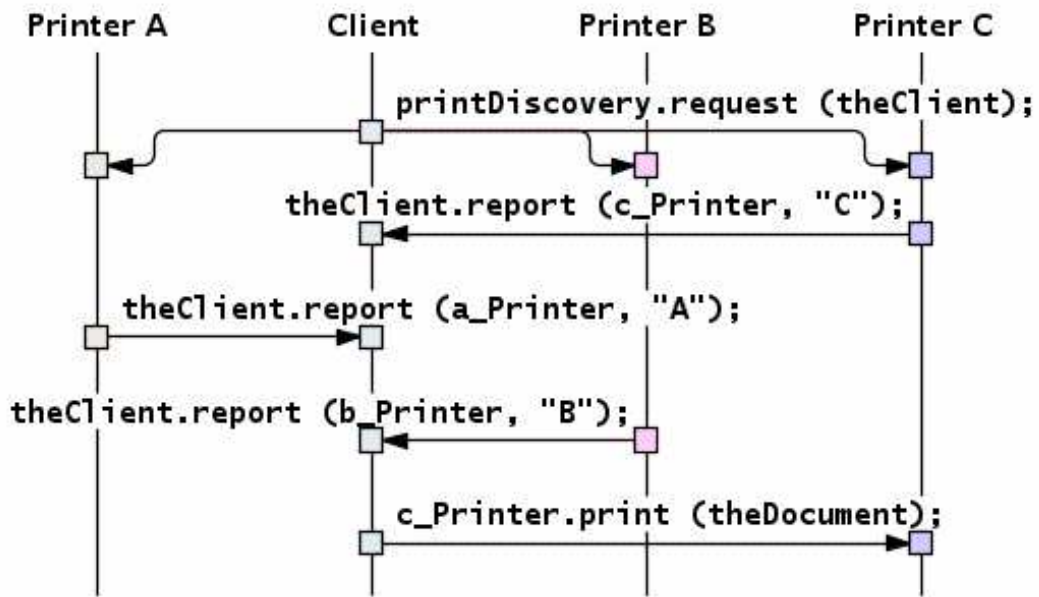
    public void putMessage(String line)
    {
        myChatFrame.addLineToLog (line);
    }

    public static void main (String args []) {
        String input;
        ChatObject aChatObject = new ChatObject( args[0] );
        while ( ( input = readFromTerminal() ) != null )
            aChatObject.send(input);
    }
}
```

## 6. Service Discovery

- Service discovery kann mit JINI erreicht werden
- Benötigt eine Server
- Wie kann ein Service Discovery in einem ad-hoc Netzwerk funktionieren?
- Es gibt keine Server in einem Ad-Hoc Netzwerk!

### 6.1. Die Idee



## 6.2. Die Interfaces

```
public interface PrintDiscovery {  
    public void request(PrintClient client);  
}
```

```
public interface PrintClient {  
    public void report(PrintService printer, String name);  
}
```

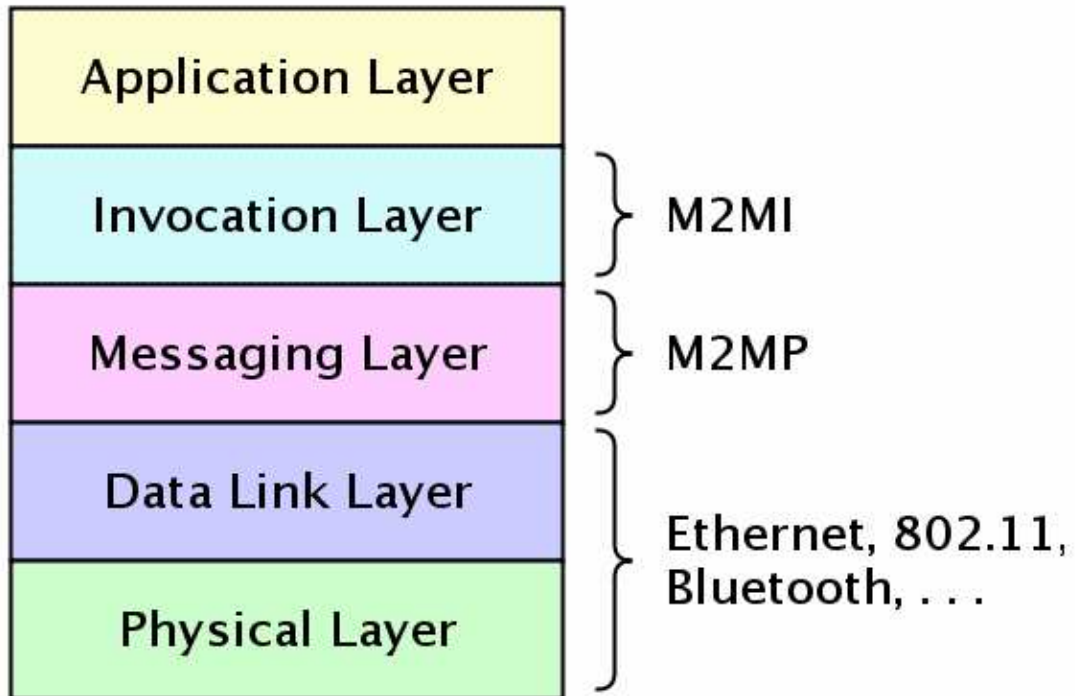
```
public interface PrintService {  
    public void print(Document doc);  
}
```

### 6.3. Andere M2MI Applikationen

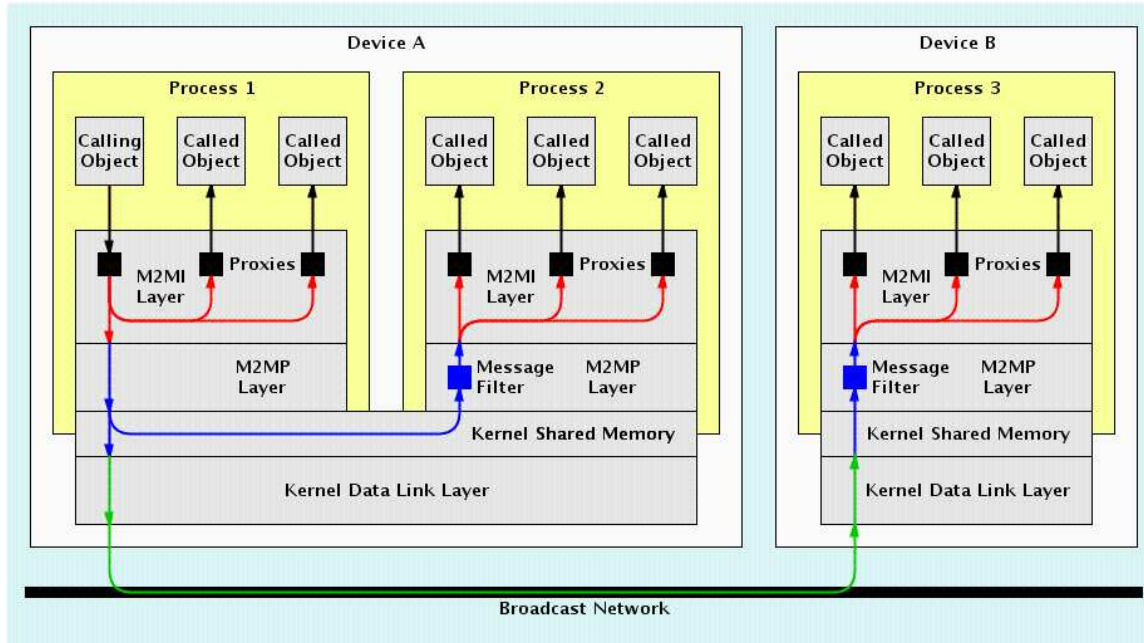
- Kommunikation  
    Kommunikationen in “quiet spaces”, “noisy spaces”, . . .
- Groupware  
    Präsentationen, Whiteboard, Kalender, etc.
- Sensor networks  
    Video Überwachung, Patienten Überwachung ...
- Middleware frameworks
  - Shared tuple spaces, (Java Spaces) ...
- Spiele!

## 7. M2MI Architektur

### 7.1. Layers



## 7.2. Software Architektur



## 8. Status

- Erste Version von M2MI ist in Java entwickelt worden
- Getestet auf Workstations
- Einige Performance und Throughput Messungen wurden vorgenommen
- Benutzt UDP/IP für den Transport
- Eine andere Version benutzt JRMS/Sockets
- Einige M2MI-basierte Applikationen wurden entwickelt

## 8.1. In Arbeit

- M2MI monitoring API
  - Beobachtungen von M2MI-Aufrufen und Debuggen
- M2MI Sicherheit
  - Confidentiality, participant authentication, service authentication
  - Zero knowledge proofs, . . .

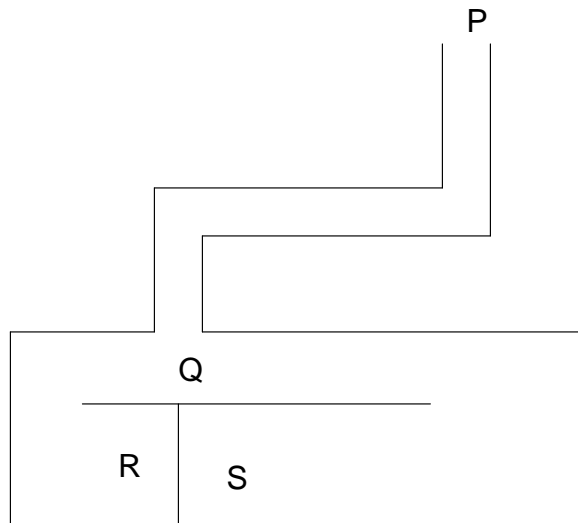
## 8.2. Security in Ad-Hoc Networks

- How do you achieve security in an ad-hoc network?
- Major Challenge: no servers.
  
- Problems to solve:
  - \* Confidentiality
  - \* Data Integrity
  - \* Participant Authentication

### 8.3. Interactive Zero Knowledge Proofs in General

- Interactive Zero Knowledge Proofs allow one party to prove its knowledge of a secret to another party without ever revealing the secret itself.
- It is useful for interactive proofs to have the following properties:
  - \* Completeness. The verifier always accepts the proof if the fact is true and both the prover and the verifier follow the protocol.
  - \* Soundness. The verifier always rejects the proof if the fact is false, as long as the verifier follows the protocol.
  - \* The verifier learns nothing about the fact being proved (except that it is correct) from the prover.
  - \* The verifier cannot even later prove the fact to anyone else.

#### 8.4. Ali Baba's Cave



- Alice wants to prove to Bob that she knows the secret words that will open the portal at R-S in the cave,
- but she does not wish to reveal the secret to Bob.

## 8.5. Zero Knowledge Proofs in Detail I

The general protocol works as follows:

- There is a prover who knows the secret and a verifier who wishes to know if the prover knows the secret.
- The prover proposes a "hard" problem to the verifier.
- The verifier asks one or more questions about a hard problem until she or he is convinced that the prover really knows the answer to the problem.
- Classic examples of the "hard" problem include factoring the product of large primes, graph isomorphism, and discrete logarithms.
- Problems that are well-suited for ZKPs are typically NP-complete, yet verifiable in polynomial time.
- However, a close analysis of specific ZKP algorithms suggests that many ZKPs are not well-suited for an ad hoc environment, because of CPU/memory limitations.
- see also: Uriel Feige and Amos Fiat and Adi Shamir, "Zero Knowledge Proofs of Identity", Proceedings of the 19th ACM Symp. on Theory of Computing, 1987.
- °RSA

## 8.6. Algorithms

- ZKP's for Quadratic Residues

Given:

$$p, q \in \mathbb{P}.$$

$$n = p * q$$

$x$  is a quadratic residue of  $n \leftrightarrow$  an  $y$  exists so such:

$$x = y^2 \pmod n.$$

$$Z_n^* = \{ i \mid i \in \mathbb{N} \text{ and } i < n \text{ and } \gcd(n, i) = 1 \}$$

1. Repeat:

Repeat the following steps  $\log_2 n$  times

- a) Peggy chooses  $v \in Z_n^*$  at random and then she computes:  $y = v^2 \pmod n$ .  
Peggy send  $y$  to Victor.
- b) Victor chooses  $i \in \{ 0, 1 \}$  at random and sends  $i$  to Peggy.
- c) Peggy computes:  $z = u^i * v \pmod n$ , where  $u \in Z_n^*$  and is a solution to  $u^2 \equiv x \pmod n$ .  
Peggy send  $z$  to Victor.
- d) Vic checks that:

$$z^2 \equiv x^i y \pmod n$$

2. Accept:

Victor accepts Peggy's proof, that  $x$  is a quadratic residue of  $n$ , if step (d) is verified  $\log_2 n$  rounds.

## 8.7. Other Algorithms

- ZKPs for Graph Isomorphism
- ZKP's for Graph 3-coloring

8.8.














































































8.9. Zukunft

- Klein und Wireless
  - Portierung von M2MI und M2MP zu kleine Geräten
  - Test in einem wireless Netzwerk
  
- M2MP in das Betriebssystem
  
- Entwicklung von mehr M2MI-basierten Applikationen in verschiedenen Umgebungen
  
- Entwicklung von M2MI-Design-Pattern

## 9. Danke

Jim Waldo inspired the idea for M2MI when he said: "Everyone that's out there, call this method" during a discussion about M2MP.

### 10. Fragen

*	+	,	-	.	/	0	1	2	3
									
4	5	6	7	8	9	:	;	<	=
									
>	?	@	A	B	C	D	E	F	G
									
H	I	J	K	L	M	N	O	P	Q
									
R	S	T	U	V	W	X	Y	Z	[
									
\	]	^	_	`	a	b	c	d	e
									
f	g	h	i	j	k	l	m	n	o
									
p	q	r	s	t	u	v	w	x	y
									

From: °<http://www.myfonts.com>

## 11. Publikationen

- 2002 Alan Kaminsky and Hans-Peter Bischof  
"Many-to-Many Invocation: A New Object Oriented  
Paradigm for Ad Hoc Collaborative Systems"  
OOPSLA 2002, Seattle, Washington, USA, November 2002
- 2002 Hans-Peter Bischof and Alan Kaminsky.  
"Many-to-Many Invocation: A new framework for building  
collaborative applications in ad hoc networks."  
CSCW 2002 Workshop on Ad Hoc Communication and  
Collaboration in Ubiquitous Computing Environments,  
New Orleans, Louisiana, USA, November 2002.

## 12. Java Reliable Multicast System

This research was supported by a grant from Sun Microsystems.

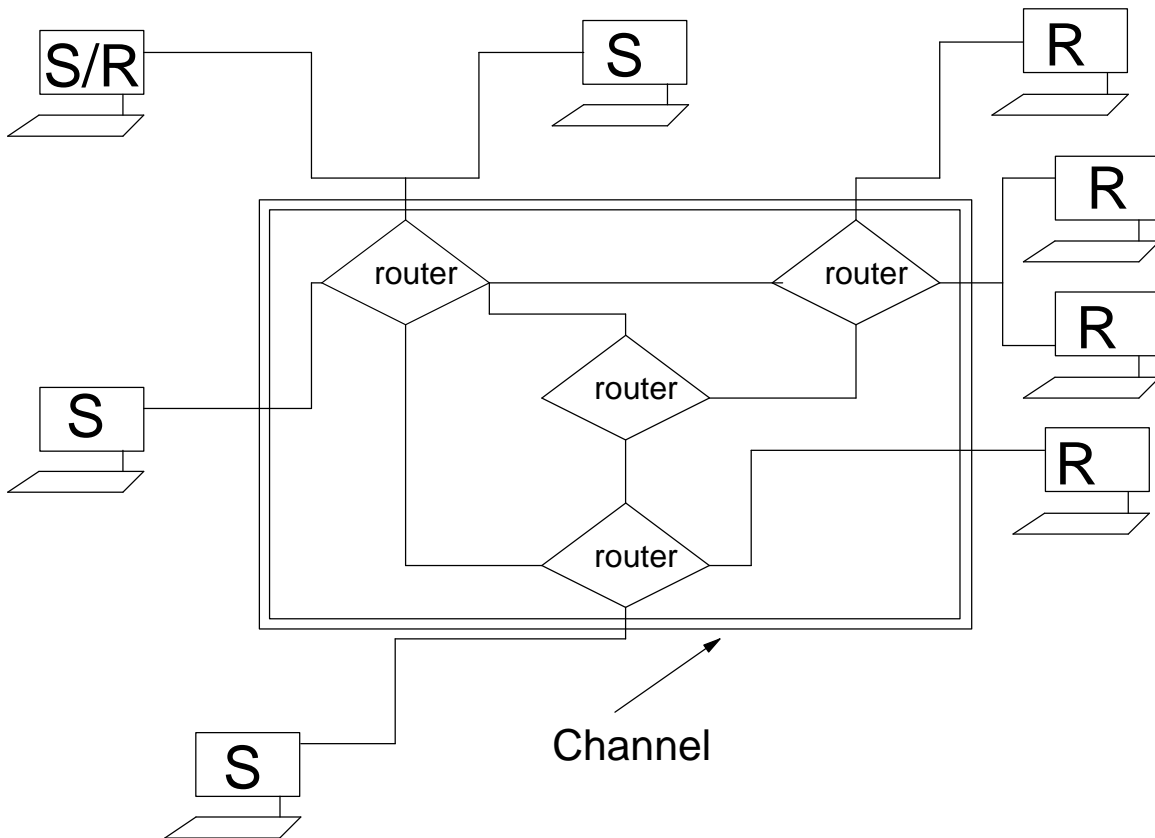
The JRMS project is completed.

## 12.1. Overview of the Problem

See also ° [www.experimentalstuff.com](http://www.experimentalstuff.com)

and ° [http://www.cs.rit.edu/~hpb/Lectures/20002/JRMS\\_590](http://www.cs.rit.edu/~hpb/Lectures/20002/JRMS_590)

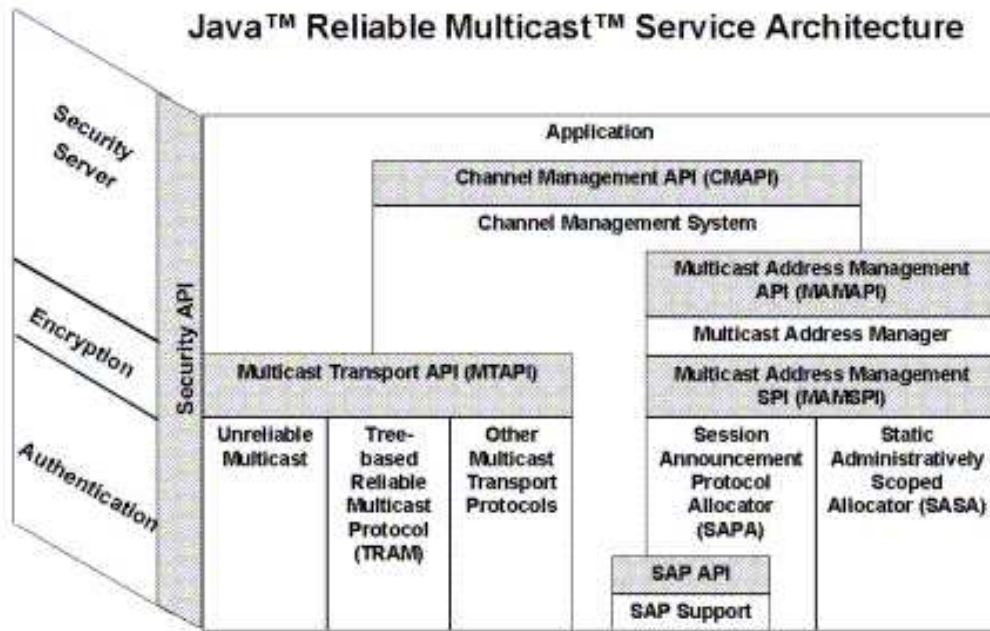
- Create a network service which enables building multicast applications that distribute information over IP networks.



## 12.2. Aspects of Reliable

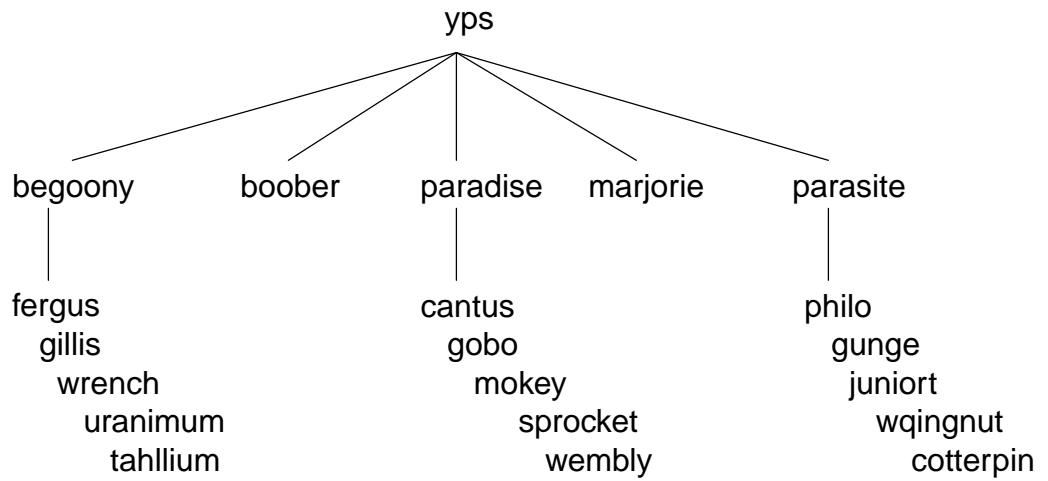
- What are the limits for reliability?
  - \* What happens if a receiver joins a channel late?
  - \* Is it possible to recover under all circumstances (pruning a receiver)?
  - \* What happens if a receiver can not keep up with the load?

### 12.3. JRMS Architecture



## 12.4. Repair Heads

- What is happening, if a receiver misses a packet?
- Does the TCP/IP idea work?
- A repair tree helps.

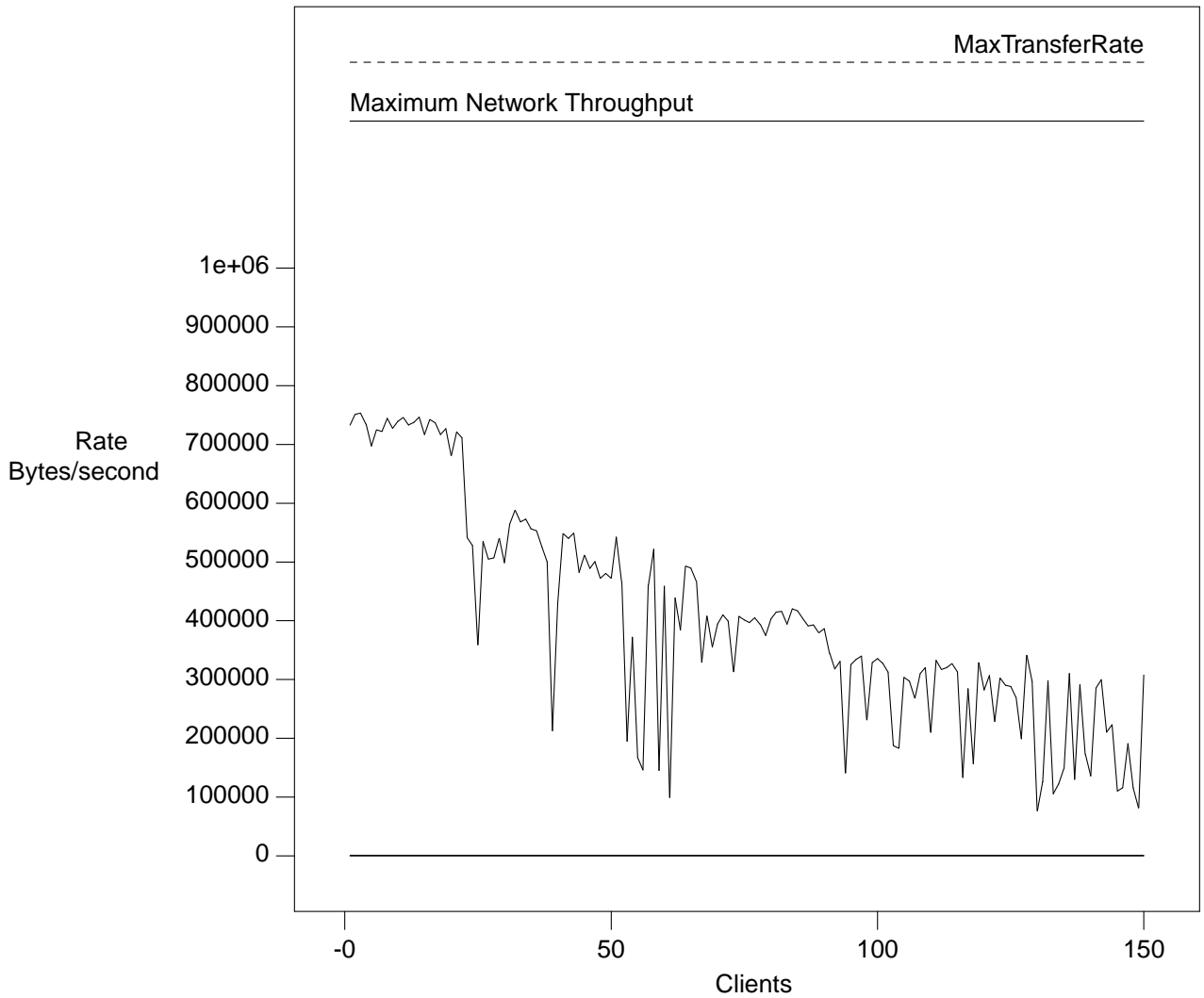


## 12.5. Throughput Problem

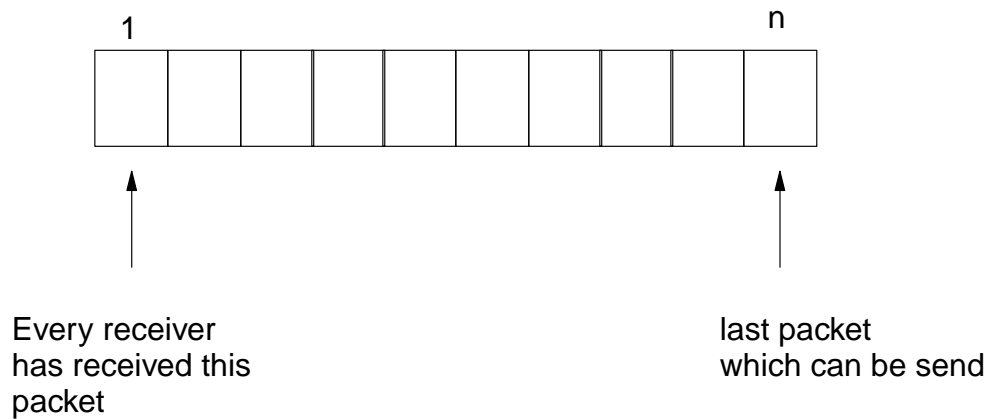
Graphical Representation of the Throughput

Graphic generated at: Sun Oct 15 18:46:01 EDT 2000.

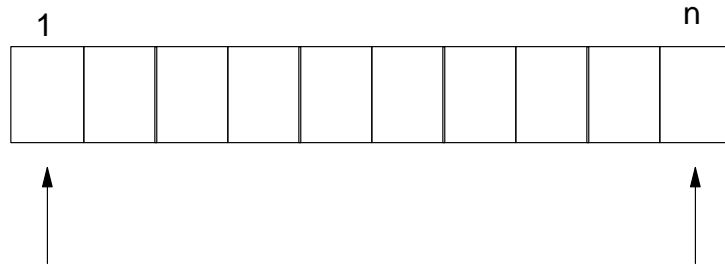
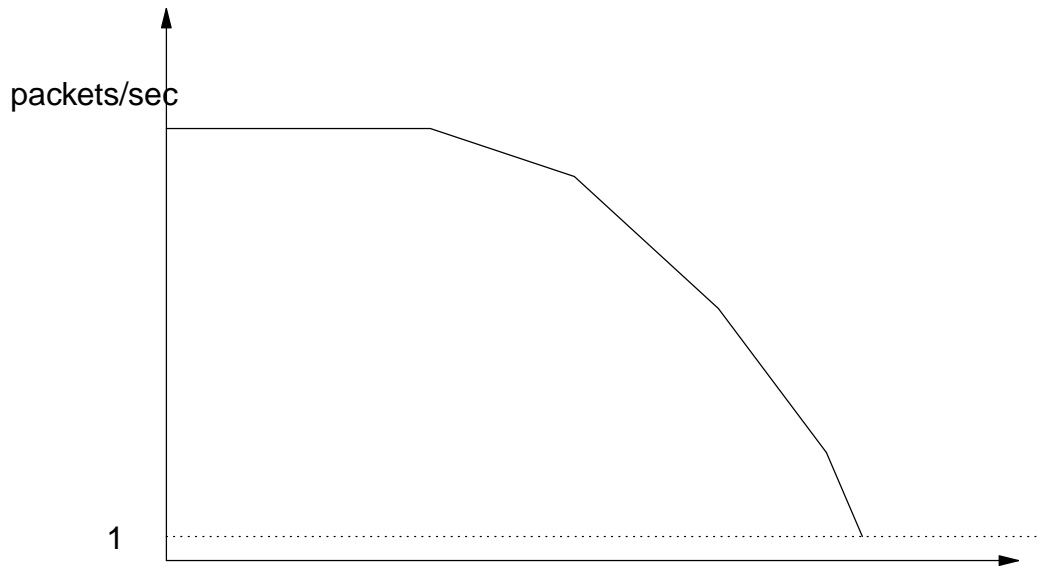
Data sent	Bytes	140000000
	KBytes	136718.750
	MBytes	133.514
	GBytes	.130385



### 12.6. Congestion Control Algorithm: Window Size



### 12.7. Congestion Control Algorithm: Sending Speed

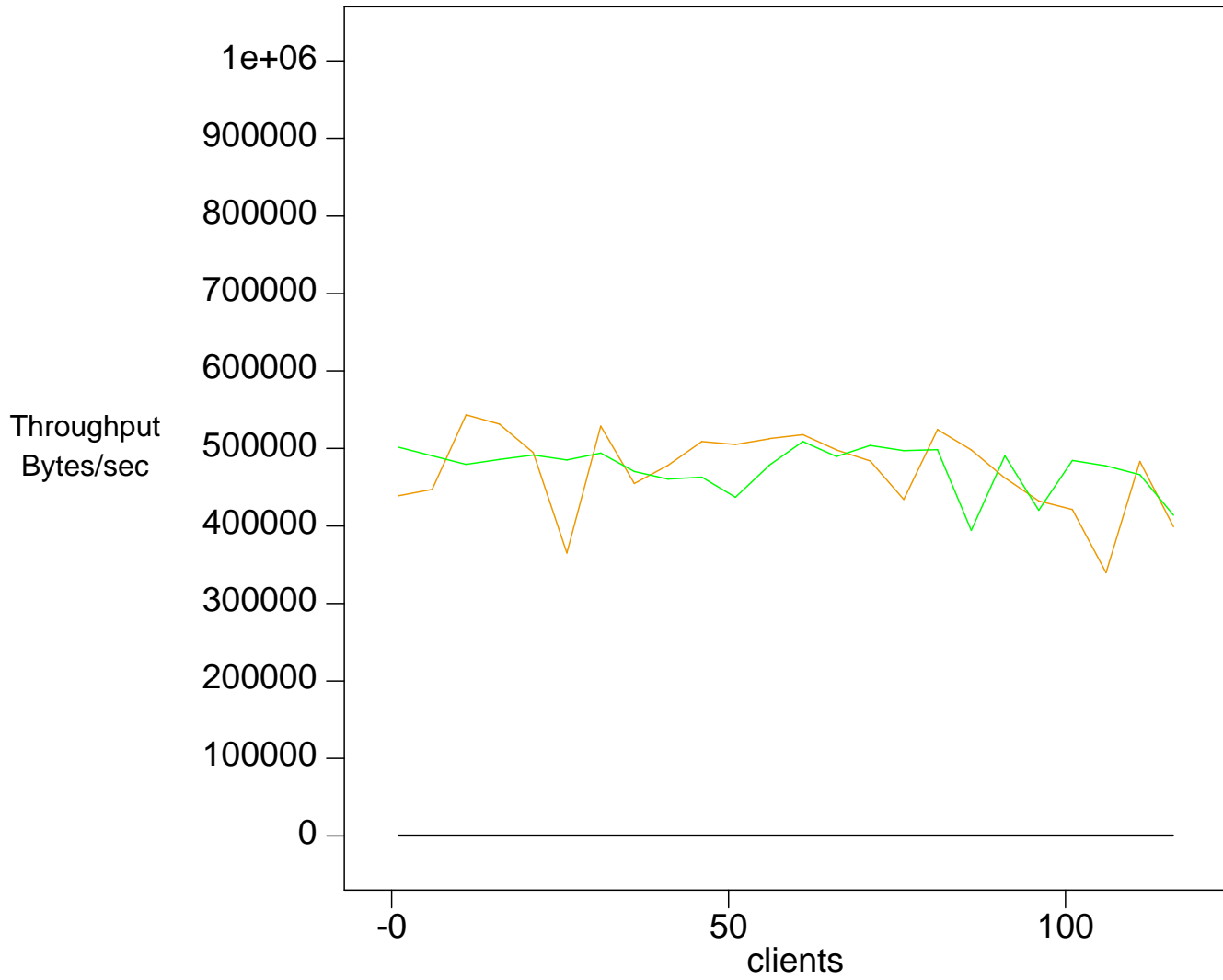


Every receiver has received this packet

last packet which can be send

### 12.8. The Result

Graphic generated at: Mon Apr 23 08:15:50 EDT 2001.



### 12.9. What was Done at RIT

- Stress test environment
- I was 3 times in Boston for a longer period of time to work with the team on JRMS.
- All tests have been done at RIT
- Two different lectures have been given in this area.
- Two researcher from SunLabs gave a talk in my lecture.

## 12.10. Publications

- 2002 "A Congestion Control Algorithm for Tree-based Reliable Multicast Protocols", INFOCOM 2002  
Authors: Dah Ming Chiu, Miriam Kadansky, Joe Provino, Joseph Wesley, Hans-Peter Bischof and Haifeng Zhu  
The paper has been accepted.
- 2001 "A Congestion Control Algorithm for Tree-based Reliable Multicast Protocols",  
Technical Report, Sun Microsystems, Report Number: TR-2001-97,  
<http://research.sun.com/technical-reports/2001/>,  
Authors: Dah Ming Chiu, Miriam Kadansky, Joe Provino, Joseph Wesley, Hans-Peter Bischof and Haifeng Zhu
- 2001 "JRMS Stress Test Environment ", only published on the web,  
download from  
<http://www.experimentalstuff.com/Technologies/JRMS/>  
Author: Hans-Peter Bischof
- 2001 "JRMS Tutorial", only published on the web, download from  
<http://www.experimentalstuff.com/Technologies/JRMS/>  
Authors: Joe Binder, Hans-Peter Bischof, Jonathan Coles, Damian Eads, Collin Fagan, Jeffrey Myers

### 13. Fragen

*	+	,	-	.	/	0	1	2	3
4	5	6	7	8	9	:	;	<	=
>	?	@	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O	P	Q
R	S	T	U	V	W	X	Y	Z	[
\	]	^	_	`	a	b	c	d	e
f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y

From: ° <http://www.myfonts.com>

#### 14. Kontakt Information

Das Anhinga Projekt

<http://www.cs.rit.edu/~anhinga/>

Hans-Peter Bischof

<http://www.cs.rit.edu/~hpb/>

[hpb@cs.rit.edu](mailto:hpb@cs.rit.edu)

Alan Kaminsky

<http://www.cs.rit.edu/~ark/>

[ark@cs.rit.edu](mailto:ark@cs.rit.edu)

