Berner Fachhochschule
Technik und Informatik

Secure Distributed Bulletin Board

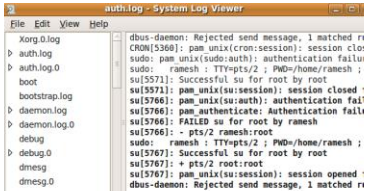
Beuchat José, 26.01.2012

What Is a Bulletin Board?

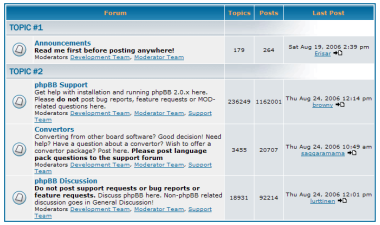
- Computer system
- Share messages
- Append only
- Public (for reading)
- Can be on the Internet

Typical Applications

- E-voting
- Auctions
- Online Petitions
- Auditable Discussion Boards
- System Logs



```
dbus-daemon: Rejected send message, 1 matched r
CRON[5360]: pam_unix(cron:session): session clo
sudo: pam_unix(sudo:auth): authentication failu
sudo:   ramesh : TTY=pts/2 ; PWD=/home/ramesh ;
su[5571]: Successful su for root by root
su[5571]: pam_unix(su:session): session closed
su[5766]: pam_unix(su:auth): authentication fai
su[5766]: pam_authenticate: Authentication fail
su[5766]: FAILED su for root by ramesh
su[5766]: - pts/2 ramesh:root
sudo:   ramesh : TTY=pts/2 ; PWD=/home/ramesh ;
su[5767]: Successful su for root by root
su[5767]: + pts/2 root:root
su[5767]: pam_unix(su:session): session opened
dbus-daemon: Rejected send message, 1 matched r
```



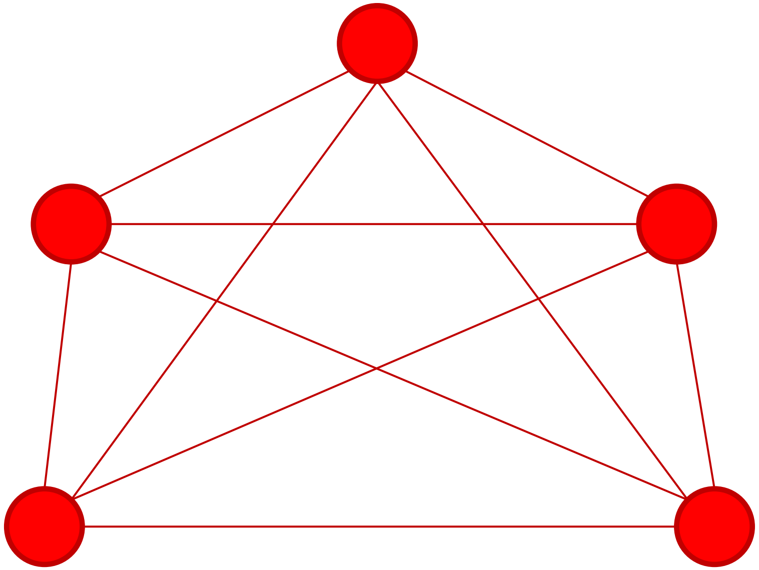
Forum	Topics	Posts	Last Post
TOPIC #1			
Announcements **Read me first before posting anywhere!** Moderators Development Team, Moderator Team	179	264	Sat Aug 19, 2006 2:39 pm Erisar
TOPIC #2			
phpBB Support Get help with installation and running phpBB 2.0.x here. Please **do not** post bug reports, feature requests or MOD-related questions here. Moderators Development Team, Moderator Team, Support Team	236249	1162001	Thu Aug 24, 2006 12:14 pm browny
Convertors Converting from other board software? Good decision! Need help? Have a question about a convertor? Wish to offer a convertor package? Post here. **Please post language pack questions to the support forum** Moderators Development Team, Moderator Team, Support Team	3455	20707	Thu Aug 24, 2006 10:49 am saggaramama
phpBB Discussion **Do not post support requests or bug reports or feature requests.** Discuss phpBB here. Non-phpBB related discussion goes in General Discussion! Moderators Development Team, Moderator Team, Support Team	18931	92214	Thu Aug 24, 2006 12:01 pm lurttinen

Requirements

- Availability
- Append only
- Failure detection
- No single point of failure

A Distributed Solution

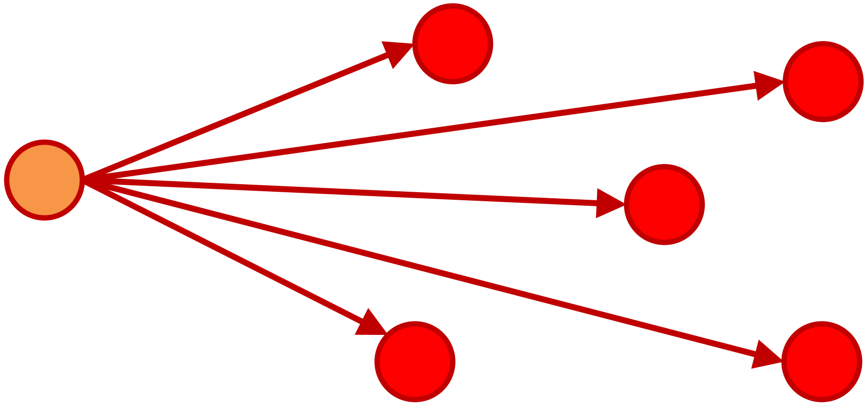
- Fully connected network with *n* parties.
- Replicated content.



Byzantine Generals Problem

- Using a secure broadcast channel ensures that every party receives the same message.

  – Condition: more than 2/3 of the parties must be correct.

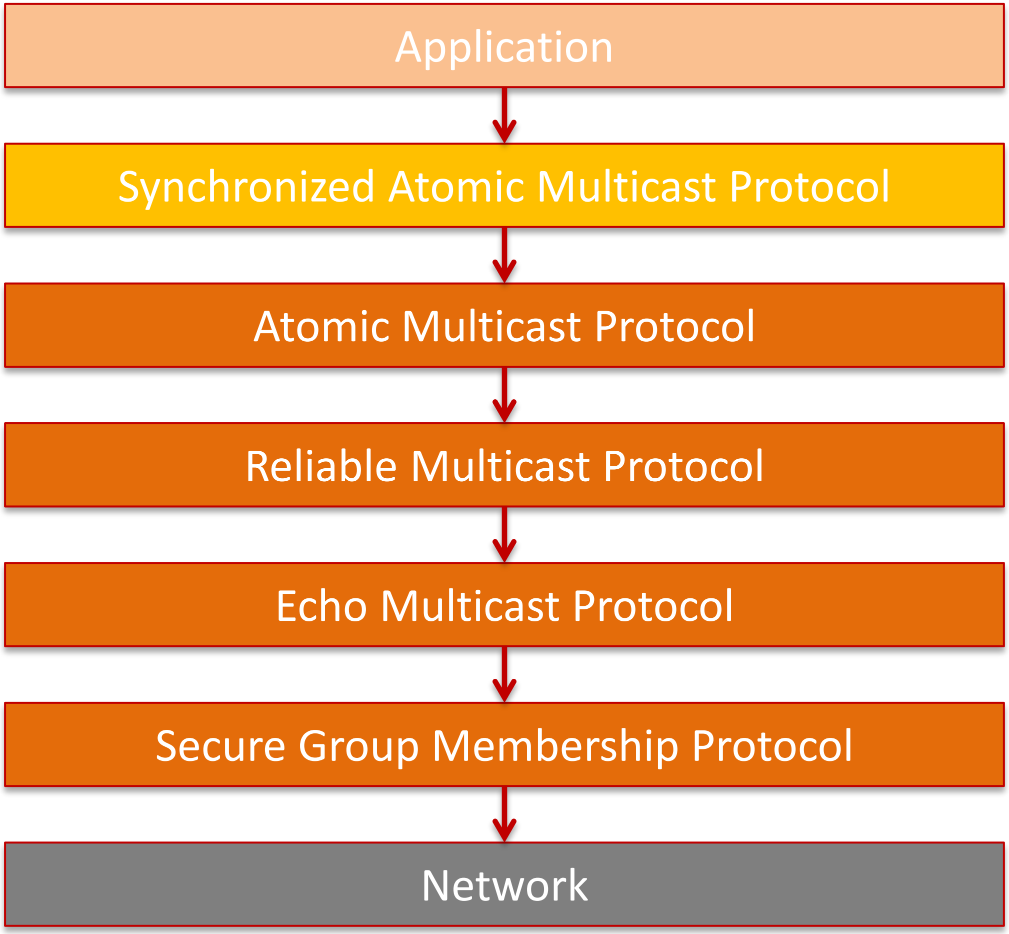


Rampart

“Rampart is a toolkit of protocols to facilitate the development of high-integrity services, i.e., ***distributed*** *services that retain their* ***availability*** *and* ***correctness*** *despite the malicious penetration of some component servers by an* ***attacker****.”*

(*Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart* by Michael K. Reiter)

The Protocols Stack



Application

Synchronized Atomic Multicast Protocol

Atomic Multicast Protocol

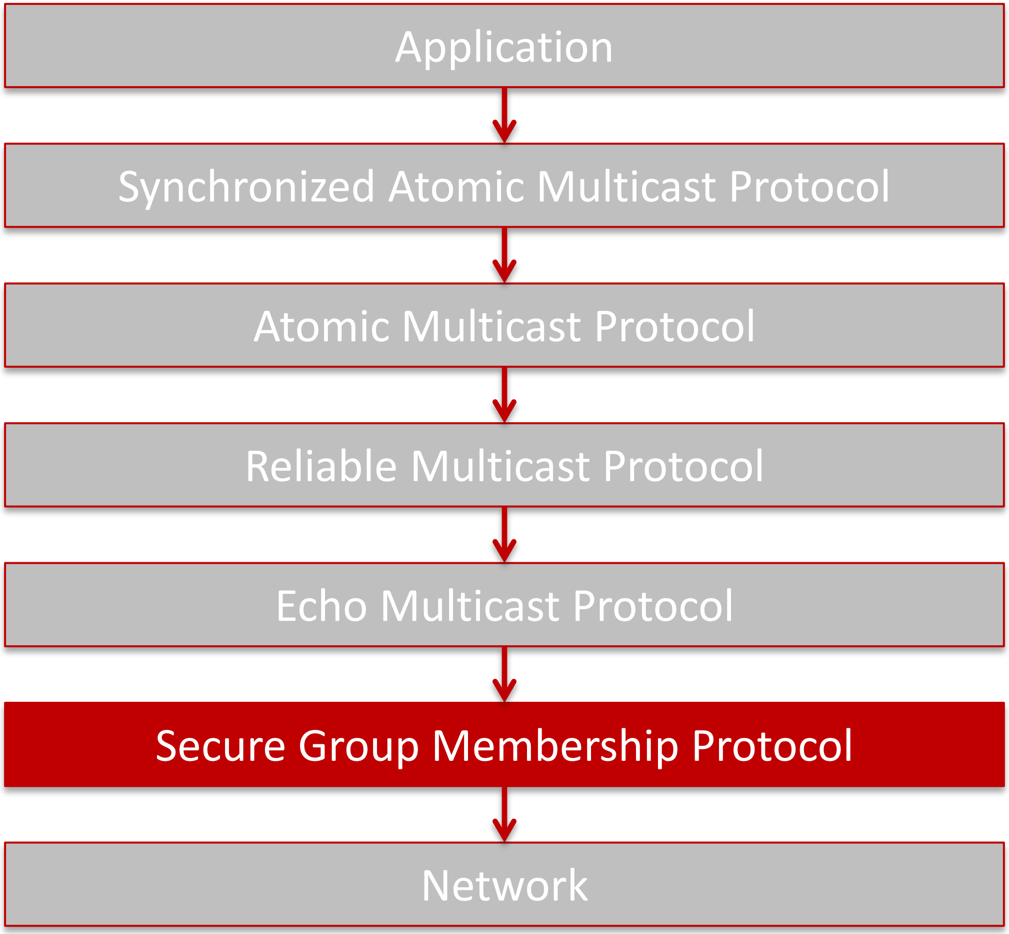
Reliable Multicast Protocol

Echo Multicast Protocol

Secure Group Membership Protocol

Network

Secure Group Membership Protocol



Application

Synchronized Atomic Multicast Protocol

Atomic Multicast Protocol

Reliable Multicast Protocol

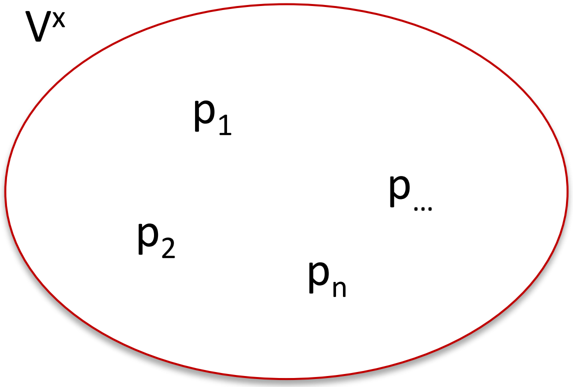
Echo Multicast Protocol

Secure Group Membership Protocol

Network

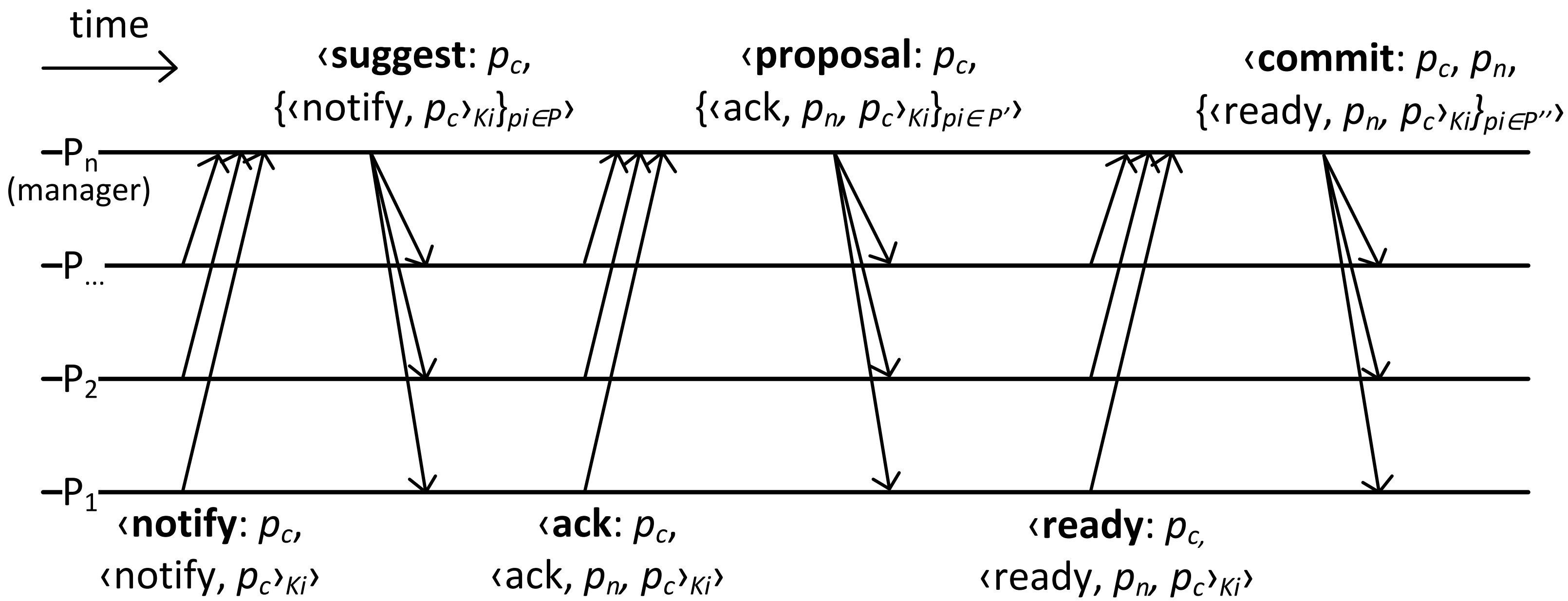
Secure Group Membership Protocol (1/2)

- Maintains a set of correct parties (the view *V*).
- Parties can be added or removed.
- Views are the same at each correct party.
- Parties are totally ordered (rank 1 to *n*).
- Party with rank *n* is the manager (p_m).



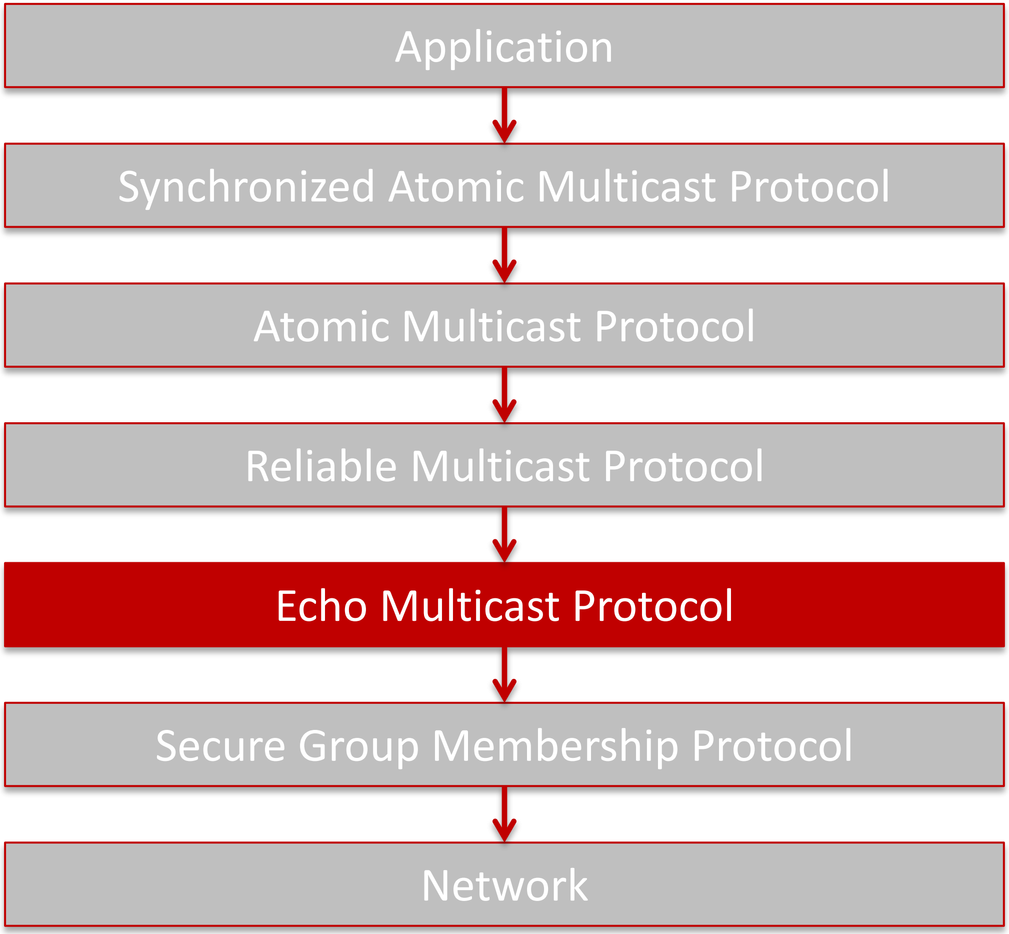
Secure Group Membership Protocol (2/2)

Remove or add a party p_c:



time
‹**suggest**: p_c,
$\{$‹notify, p_c›$_{Ki}\}_{pi \in P}$›
‹**proposal**: p_c,
$\{$‹ack, p_n, p_c›$_{Ki}\}_{pi \in P'}$›
‹**commit**: p_c, p_n,
$\{$‹ready, p_n, p_c›$_{Ki}\}_{pi \in P''}$›
P_n
(manager)
$P_{...}$
P_2
P_1
‹**notify**: p_c,
‹notify, p_c›$_{Ki}$›
‹**ack**: p_c,
‹ack, p_n, p_c›$_{Ki}$›
‹**ready**: p_c,
‹ready, p_n, p_c›$_{Ki}$›

Echo Multicast Protocol



Application

Synchronized Atomic Multicast Protocol

Atomic Multicast Protocol

Reliable Multicast Protocol

Echo Multicast Protocol

Secure Group Membership Protocol

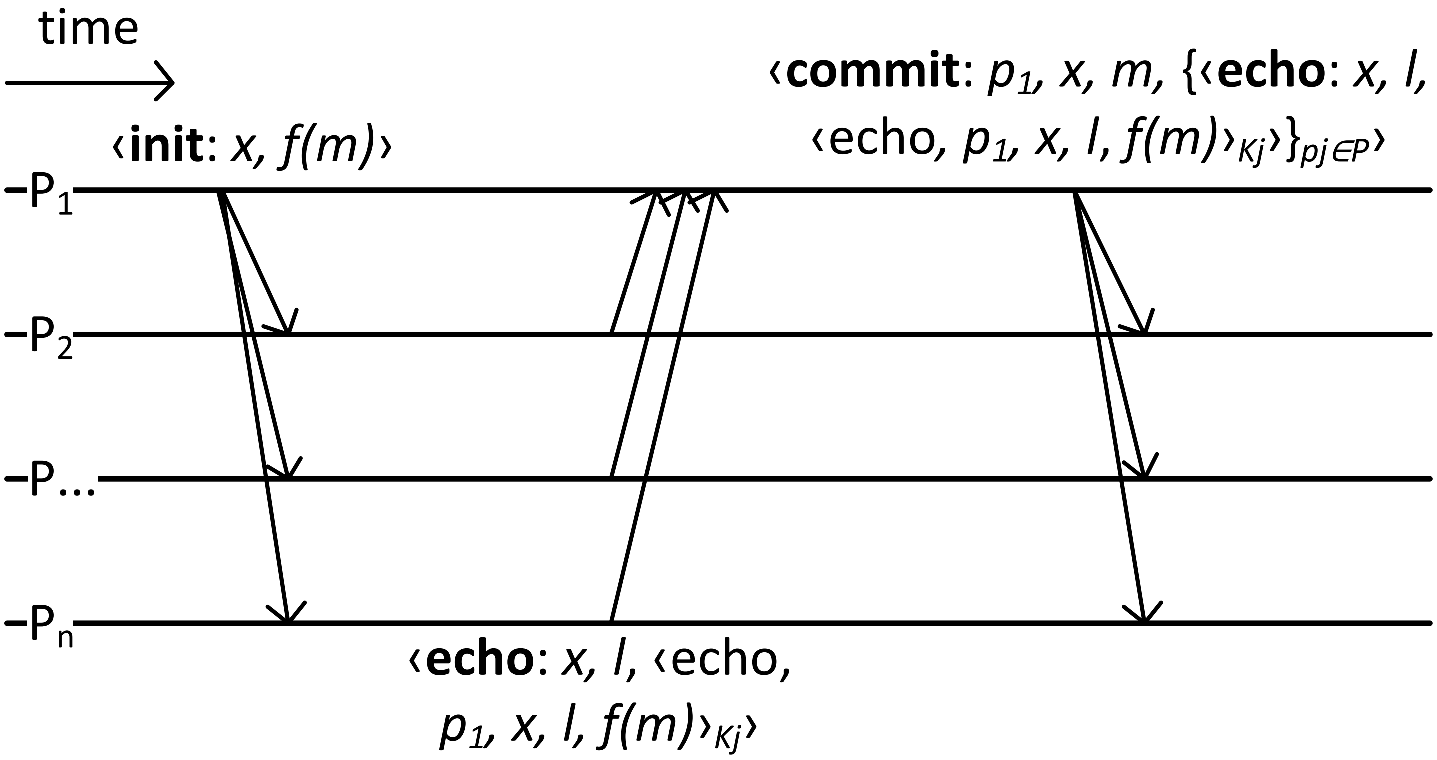
Network

Echo Multicast Protocol (1/2)

- Ensures that each message sent by a party *p* in view *x* is the same at each correct party.
- Delivers messages to the upper layer once they are stable (received by every party).
- Each party periodically sends counters messages, indicating the messages it has received.

Echo Multicast Protocol (2/2)

Party p_1 multicast a message m in view x:



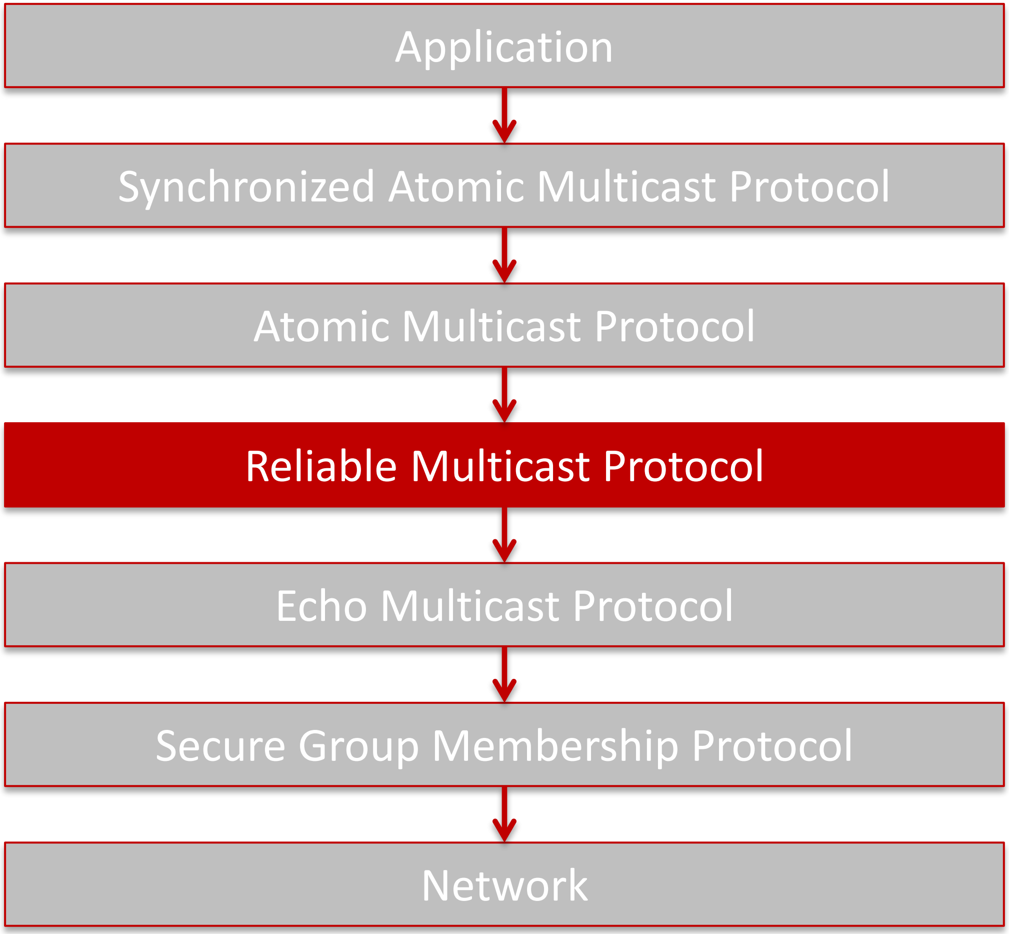
$\langle \textbf{init}: x, f(m) \rangle$

$\langle \textbf{echo}: x, l, \langle \text{echo}, p_1, x, l, f(m) \rangle_{K_j} \rangle$

$\langle \textbf{commit}: p_1, x, m, \{ \langle \textbf{echo}: x, l, \langle \text{echo}, p_1, x, l, f(m) \rangle_{K_j} \rangle \}_{p_j \in P} \rangle$

l: amount of messages received by a certain party in view x

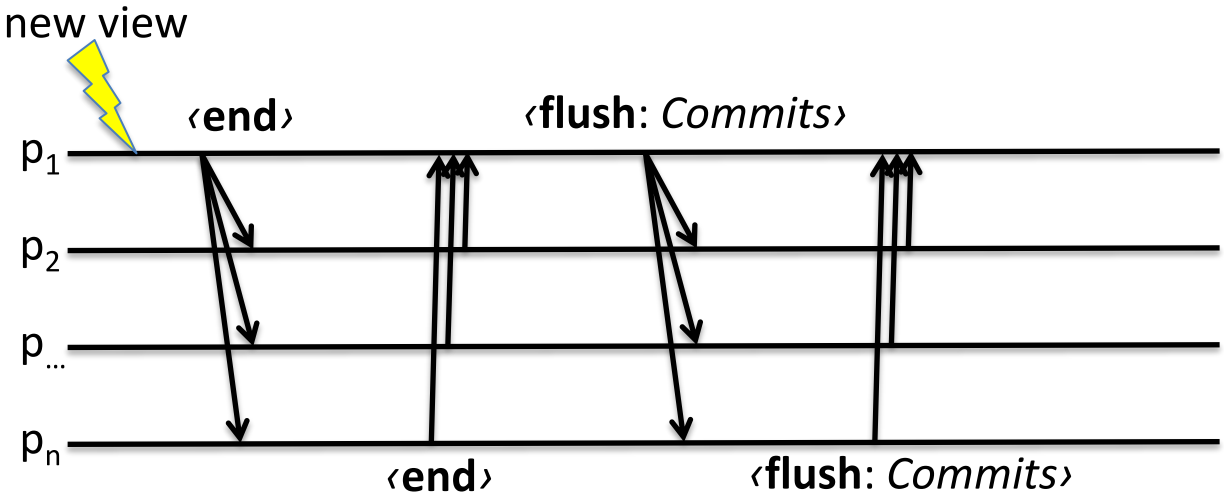
Reliable Multicast Protocol

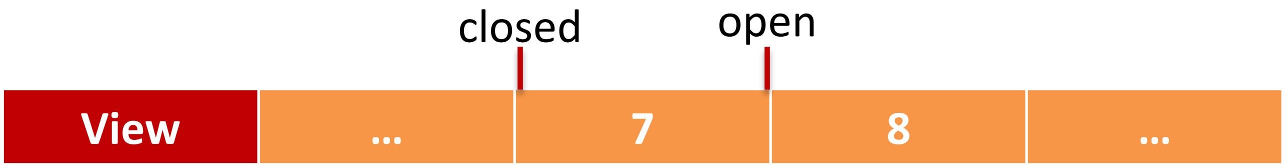

Application
Synchronized Atomic Multicast Protocol
Atomic Multicast Protocol
Reliable Multicast Protocol
Echo Multicast Protocol
Secure Group Membership Protocol
Network

Reliable Multicast Protocol (1/2)

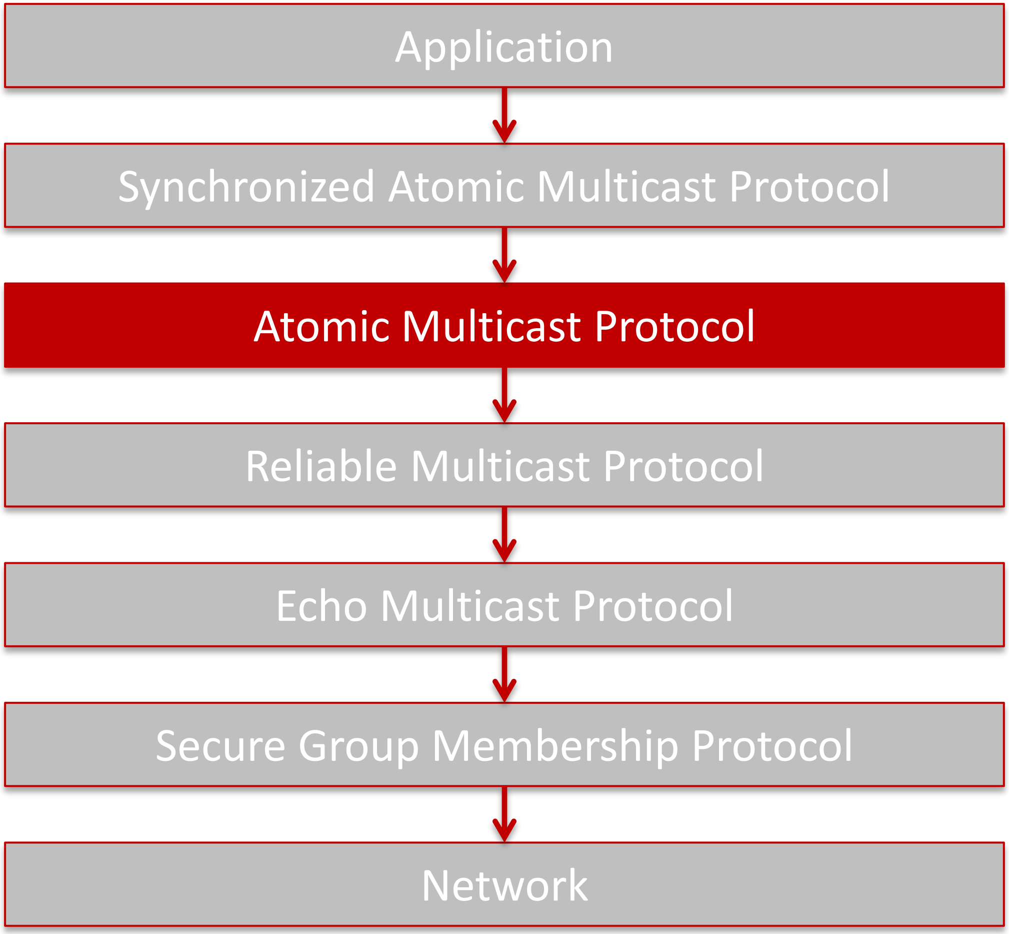
- Uses the Echo Multicast Protocol.
- Makes messages stable if a new view is created.
- Refuses messages for closed views.
- Buffers messages for future views.

Reliable Multicast Protocol (2/2)





Atomic Multicast Protocol



Application

Synchronized Atomic Multicast Protocol

Atomic Multicast Protocol

Reliable Multicast Protocol

Echo Multicast Protocol

Secure Group Membership Protocol

Network

Atomic Multicast Protocol (1/2)

- Uses the Reliable Multicast Protocol.
- Ensures that correct parties deliver messages in the same order.
- Each view has a sequencer, which periodically multicast the order in which it received the messages.

Atomic Multicast Protocol (2/2)

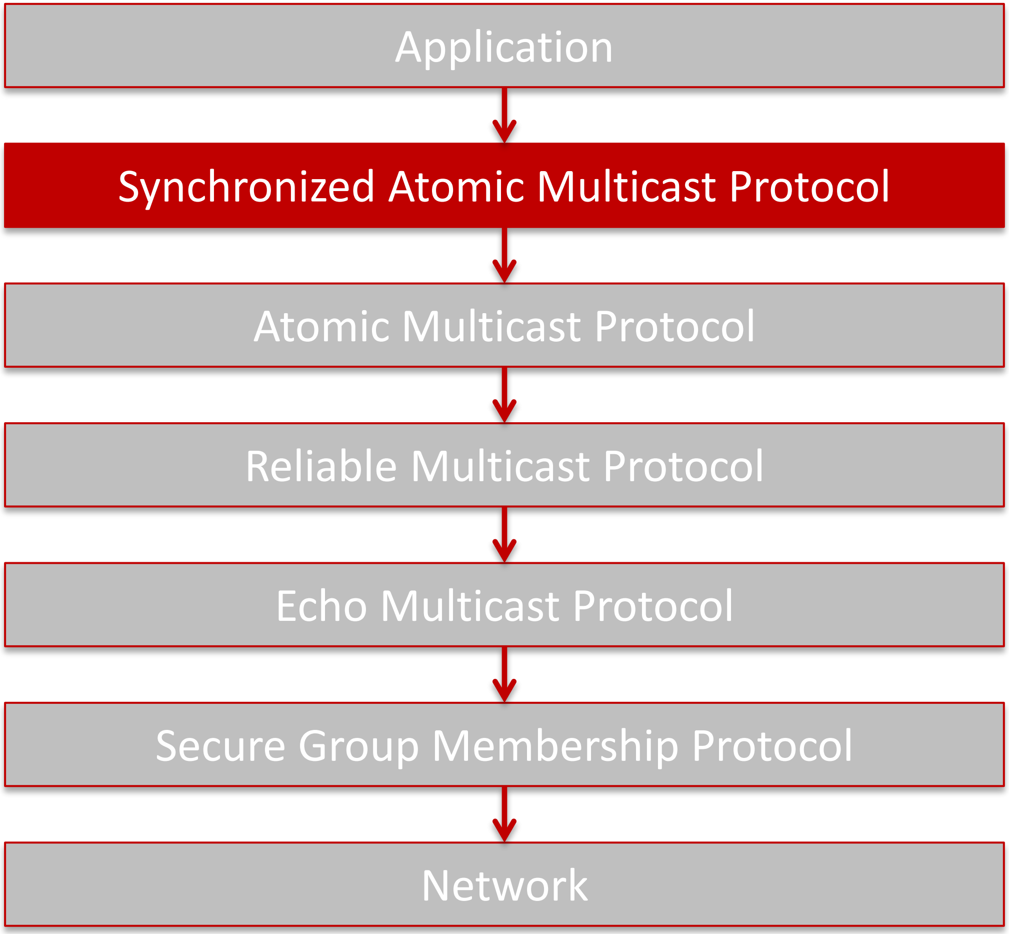
Messages at party p_1:

Messages	$\mathit{Pending}_{p1}$	$\mathit{Pending}_{p\ldots}$	$\mathit{Pending}_{pn}$
m_1 from p_1	m_1		
m_3 from p_1	m_1, m_3		
m_2 from p_n	m_1 , m_3		m_2
‹Order: *Senders*› from p_n			

Messages at party p_n (the sequencer):

Messages	$\mathit{Pending}_{p1}$	$\mathit{Pending}_{p\ldots}$	$\mathit{Pending}_{pn}$	*Senders*
m_1 from p_1	m_1			p_1
m_2 from p_n	m_1		m_2	p_1, p_n
m_3 from p_1	m_1, m_3		m_2	p_1, p_n, p_1
‹Order*: Senders*› from p_n				

Synchronized Atomic Multicast Protocol



Application

Synchronized Atomic Multicast Protocol

Atomic Multicast Protocol

Reliable Multicast Protocol

Echo Multicast Protocol

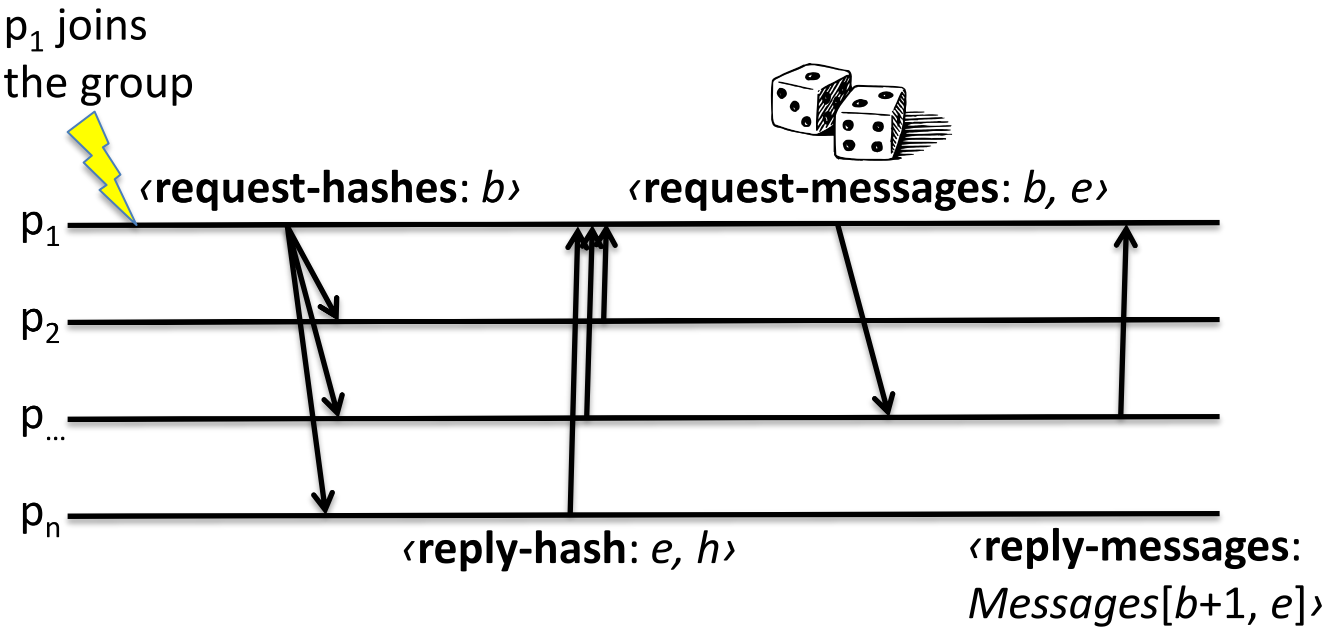
Secure Group Membership Protocol

Network

Synchronized Atomic Multicast Protocol (1/2)

- Uses the Atomic Multicast Protocol.
- Synchronizes messages at parties (re)joining the group.

Synchronized Atomic Multicast Protocol (2/2)



p_1 joins the group

‹**request-hashes**: *b*›

‹**request-messages**: *b, e*›

p_1

p_2

$p_{\dots}$

p_n

‹**reply-hash**: *e, h*›

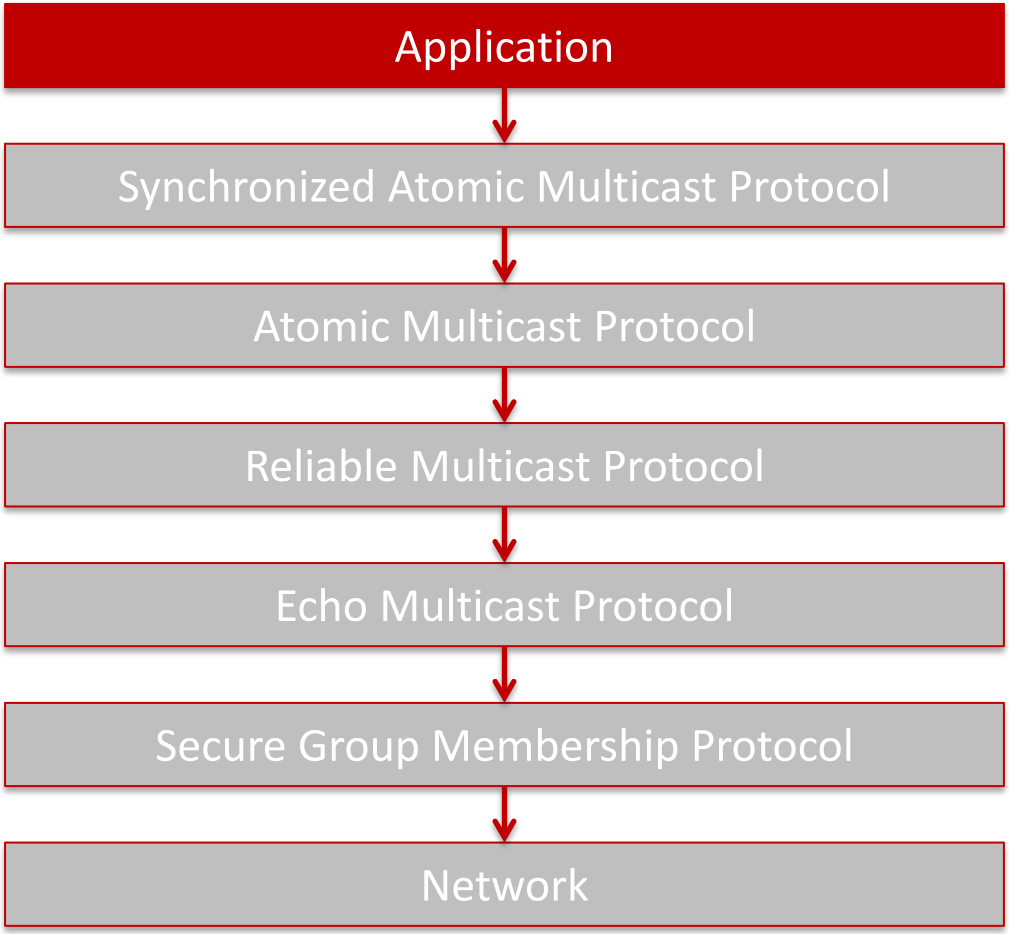
‹**reply-messages**: *Messages*[*b*+1, *e*]›

b: amount of already received messages

e: amount of missing messages

h: hash of the missing messages

Application


Application
Synchronized Atomic Multicast Protocol
Atomic Multicast Protocol
Reliable Multicast Protocol
Echo Multicast Protocol
Secure Group Membership Protocol
Network

Application

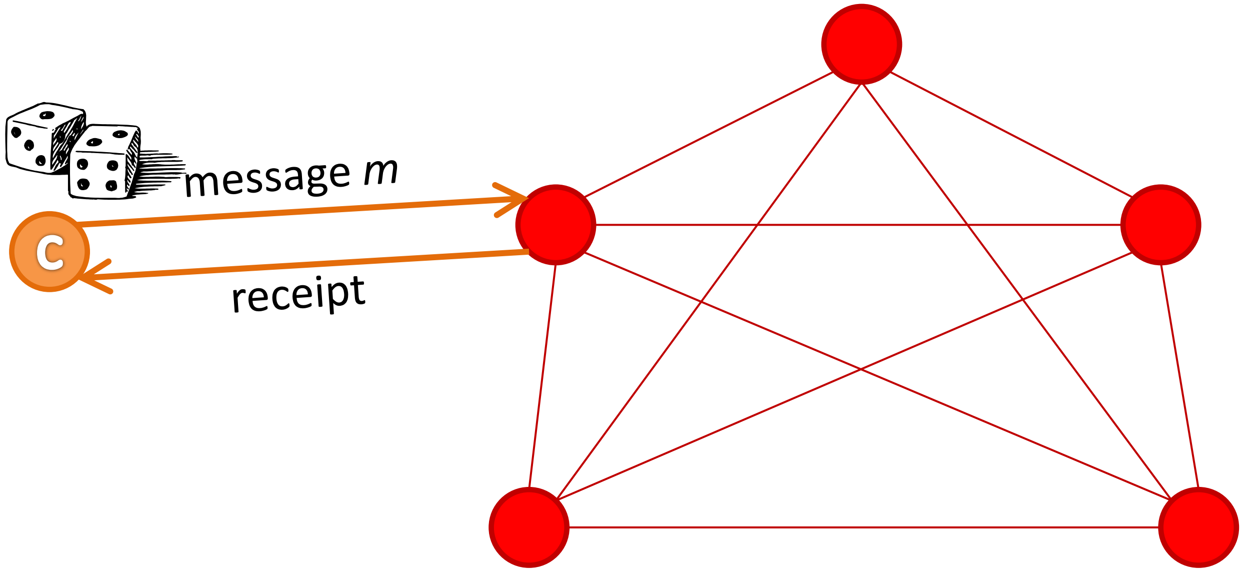
- Depends of the context.
- Consists of three protocols:
  - The Server Protocol
  - The Client Writing Protocol
  - The Client Reading Protocol

The Server Protocol (1/3)

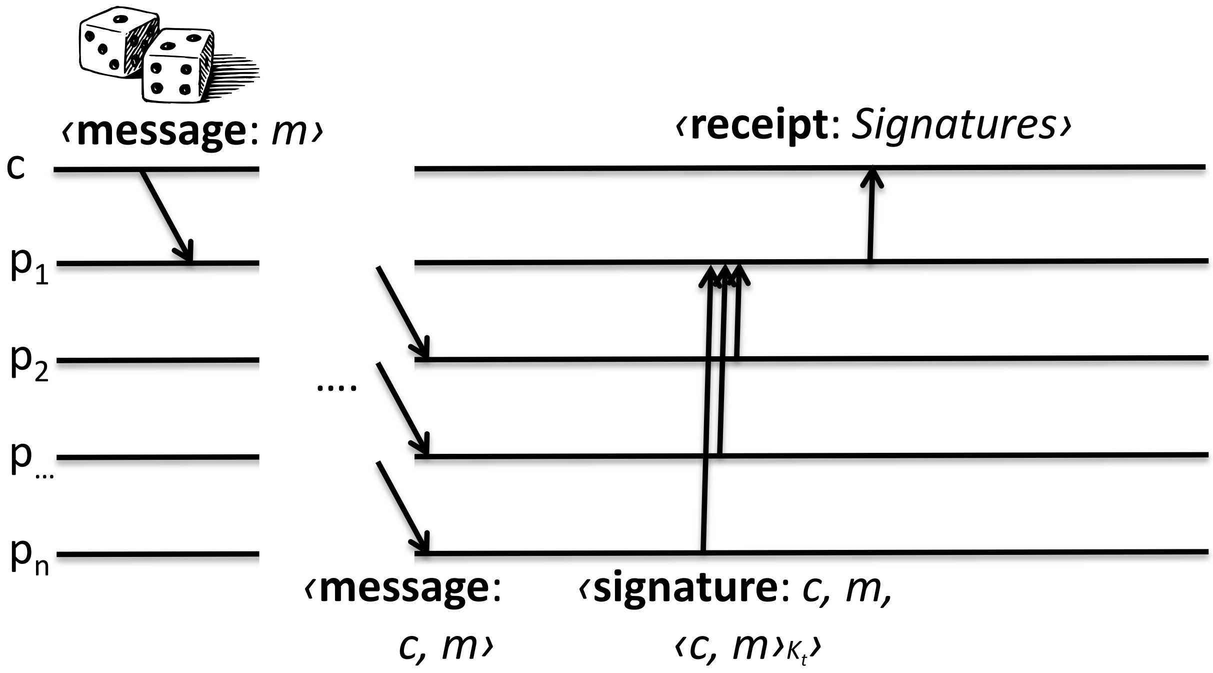
- Receives messages from clients and multicast them to the other parties using the Synchronized Atomic Multicast Protocol.
- Persists the messages.
- Sends receipts to the clients.
- Answers read requests.

The Client Writing Protocol

- A client *c* randomly selects one of the parties to post his message *m*.
- If the client does not receive a correct receipt in time, it selects another party.



The Server Protocol (2/3)



‹**message**: *m*›

‹**receipt**: *Signatures*›

c

p_1

p_2

$p_{...}$

p_n

....

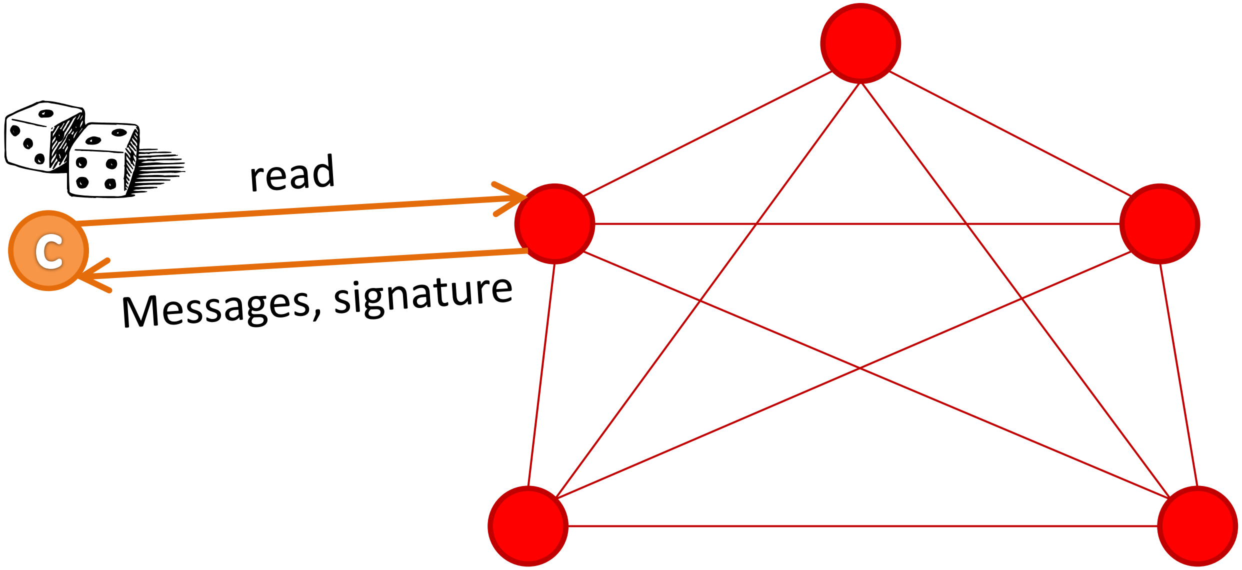
‹**message**: *c, m*›

‹**signature**: *c, m,* *‹c, m›*$_{K_t}$›

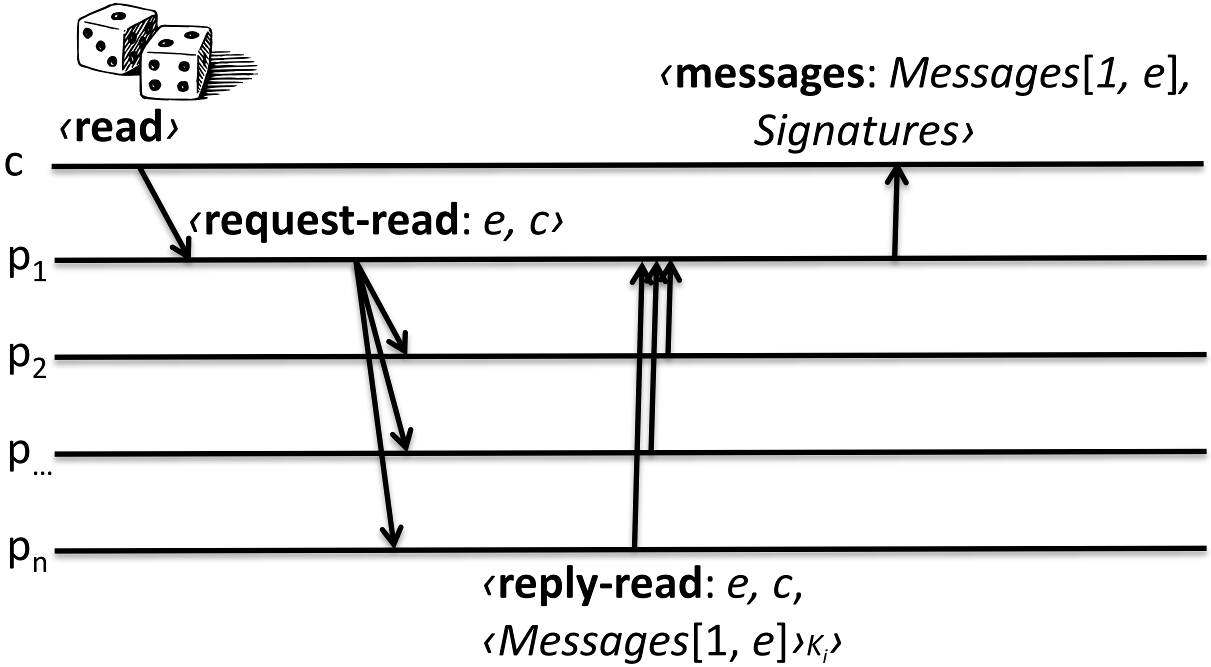
m: the message

The Client Reading Protocol

- A client *c* randomly selects one of the parties and sends a read request.
- If the client does not receive the messages and a correct signature in time, it selects another party.



The Server Protocol (3/3)



‹**read**›

‹**request-read**: *e*, *c*›

‹**reply-read**: *e*, *c*,
‹*Messages*[1, *e*]›$_{K_i}$›

‹**messages**: *Messages*[*1*, *e*],
Signatures›

e : the amount of messages at p_1

Conclusion

- Using the presented protocols, we can build a secure distributed bulletin board which satisfies the requirements.
- Implementation status.
- Future works:

  - group threshold signatures

  - persistence service

  - application (e-voting?)

Sources

- [1] Secure agreement protocols: Reliable and Atomic Group Multicast in Rampart. Michael K. Reiter, 1994.
- [2] A Secure Group Membership Protocol. Michael K. Reiter, 1996.
- [3] A Secure Bulletin Board. Richard A. Peters, 2005.