
Verteilte und Parallele Programmierung

Mit Virtuellen Maschinen

Prof. Dr. Stefan Bosse

Universität Koblenz - FB Informatik - FG Praktische Informatik

Verteilte Programmierung (Web)

Wie können Web Browser für die verteilte Datenverarbeitung genutzt werden?

Zentrales Thema Kommunikation: Wie können Web Browser (und Web Seiten) vernetzt werden?

Prozesskommunikation mit Channels über Web Sockets und Web Real-time Communication

Gruppenkommunikation

Verteilte Systeme

- Parallele Systeme sind eng gekoppelt. Kommunikation zwischen Prozessen nutzt:
 - Physisch geteilten Speicher (Unsynchronisierter Datenaustausch);
 - Semaphore (Synchronisation, Kooperation, Schutz), häufig durch Betriebssystem implementiert;
 - Datenkanäle (**Channels**, Synchronisierter Datenaustausch).
- Verteilte Systeme sind lose gekoppelt und können für die Synchronisation nicht auf das Betriebssystem und die technischen Eigenschaften des Hostrechners zurückgreifen!
 - Aber Channels können auch zwischen getrennten Rechnern mit Nachrichtenkommunikation implementiert werden!
 - Channels können dann genutzt werden um andere Synchronisationsobjekte (z.B. Semaphore) zu implementieren

Verteilte Systeme

Parallele Systeme sind i.A. synchrone Prozesssysteme

Verteilte Systeme sind inhärent asynchron und können nicht grundsätzlich und zuverlässig synchronisiert werden!

- Kommunikation mit Nachrichten erfordert ein Kommunikationsprotokoll (siehe ISO Netzwerkschichten)
 - HTTP/HTTPS (Hyper Text Transfer Protocol + Secure, temporär verbindungsorientiert) →
 - TCP (Transmission Control Protocol, verbindungsorientiert, zuverlässig) |
 - UDP (User Datagram Protocol, verbindungslos, nicht zuverlässig)
 - Web Sockets (aufbauend auf HTTP/HTTPS/TCP)

Verteilte Systeme

- Parallele Systeme arbeiten häufig nach dem Prinzip der Produzenten-Konsumenten Interaktion und Partitionierung
- Es gibt häufig einen ausgewiesenen Master und eine Vielzahl von Worker Prozessen
- In verteilten Systemen gibt es häufig ein Leader Prozess (der selbst aber auch Worker sein kann)
- Leader und Worker bilden eine (temporäre) **Gruppe**
 - Der Leader kann von vornherein fest stehen oder von der Gruppe gewählt werden → Leader Election Algorithmen
 - Der Leader vermittelt Worker und Kommunikation
 - Kommunikation kann zwischen Worker und Leader und zwischen Workern stattfinden
 - Gruppenkommunikation (Broad- und Multicast)

Verteilte Systeme

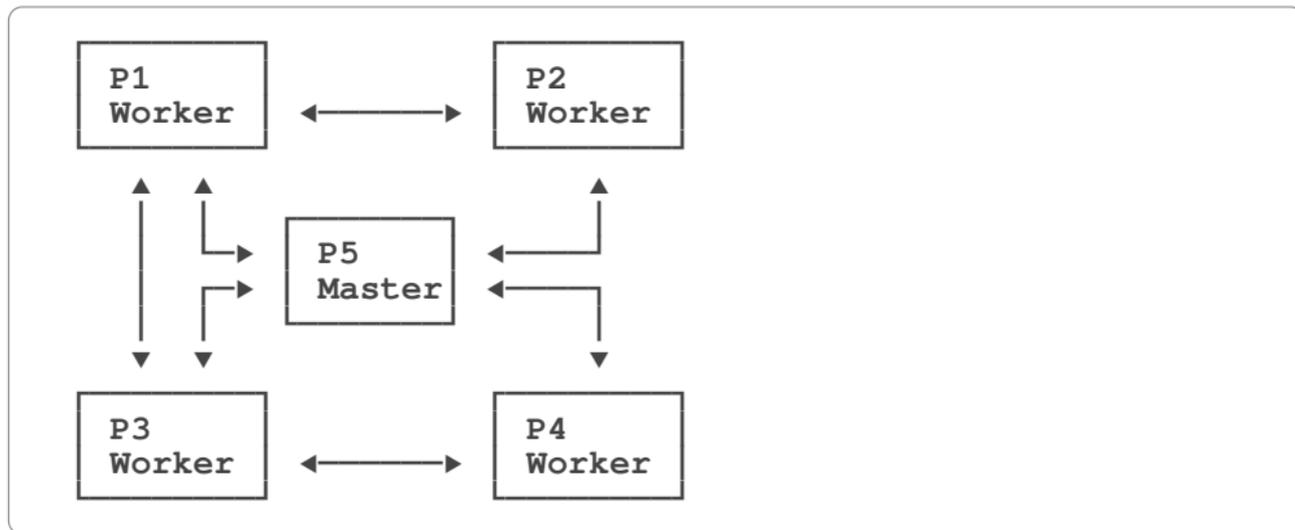
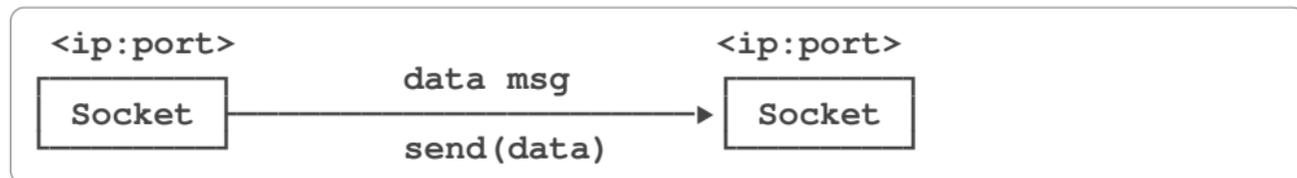


Abb. 1. Verteiltes Prozesssystem mit Leader(Master) und Workern und Kommunikationskanälen (1:1 oder 1:N)

Sockets

- Ein Socket ist sowohl ein Kommunikationsendpunkt (zum Empfang von Nachrichten) als auch ein Sender von Nachrichten
- Sockets (die i.A. zu Prozessen gehören) können miteinander verbunden werden (verbindungsorientierte Kommunikation mit Sessions) oder verbindungslos Datenpakete sich gegenseitig zusenden
- Ein Socket wird durch eine lokale oder netzwerkweite Identifikation (Pfad, Nummer, Nummer und Netzwerkadresse des Gerätes, Namen usw.) erreicht



Beispiel: TCP Socket in Lua/LVM

Server

```
local chaneServer = luv.net.tcp()
chanServer:bind(host, port)
function on_connction(chanClient)
  -- handle client request
  data1=client:read()
  client:write(data2)
end
chanServer:listen(128, function(err)
  -- Accept the client
  local chanClient = luv.net.tcp()
  server:accept(chanClient)
  on_connection(chanClient)
end)
```

Klient

```
local chanClient = luv.net.tcp()
chanClient:connect(host, port,
  function (err)
    -- check error and carry on.
    chanClient:write(data1)
    data2=chanClient:read()
  end)
```

Bsp. 1. Lua Socket mit bidirektionaler TCP
Kanalverbindung

TCP versa UDP

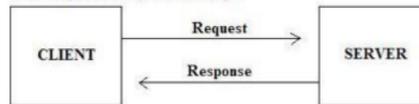
TCP

Transmission Control Protocol → verbindungsorientiertes Protokoll

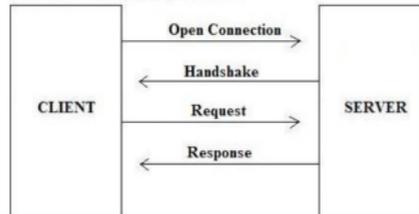
UDP

User Datagram Protocol → verbindungsloses Protokoll

UDP Request / Response Paradigm



TCP Handshake Paradigm



<https://www.cstack.org/difference-connection-oriented-connectionless-services-protocol/>

TCP versa UDP

Faktor	TCP	UDP
Verbindungstyp	Erfordert eine bestehende Verbindung, bevor Daten übertragen werden; Nur Unicast	Zum Starten und Beenden einer Datenübertragung ist keine Verbindung erforderlich. Uni-, Multi- und Braodcast Adressierung möglich
Adressierungstyp	Nur Unicast	Uni-, Multi- und Broadcast Adressierung möglich
Reihenfolge der Daten	Kann Daten sequenzieren (in einer bestimmten Reihenfolge senden)	Keine Reihenfolge oder Anordnung der Daten möglich
Neuübertragung von Daten	Kann Daten erneut übermitteln, falls Pakete nicht ankommen	Keine erneute Datenübermittlung. Verlorene Daten können nicht wiederhergestellt werden
Zuverlässigkeit	Die Zustellung ist garantiert	Die Zustellung ist nicht garantiert
Fehlerprüfung	Gründliche Fehlerkontrolle garantiert, dass die Daten in ihrem vorgesehenen Zustand ankommen	Die minimale Fehlerprüfung deckt die Grundlagen ab, kann aber nicht alle Fehler verhindern
Nachrichtengröße	Theoretisch unbegrenzt	Auf Datenpaketgröße begrent (z.B. 4kB)
Datenfragmentierung	Automatisch	Manuell
Geschwindigkeit	Höhere Latenz, niedrigere Bandbreite	Schneller, aber mit der Gefahr einer unvollständigen Datenzustellung

Tab. 1. Vergleich TCP/UDP

<https://www.avast.com/de-de/c-tcp-vs-udp-difference>

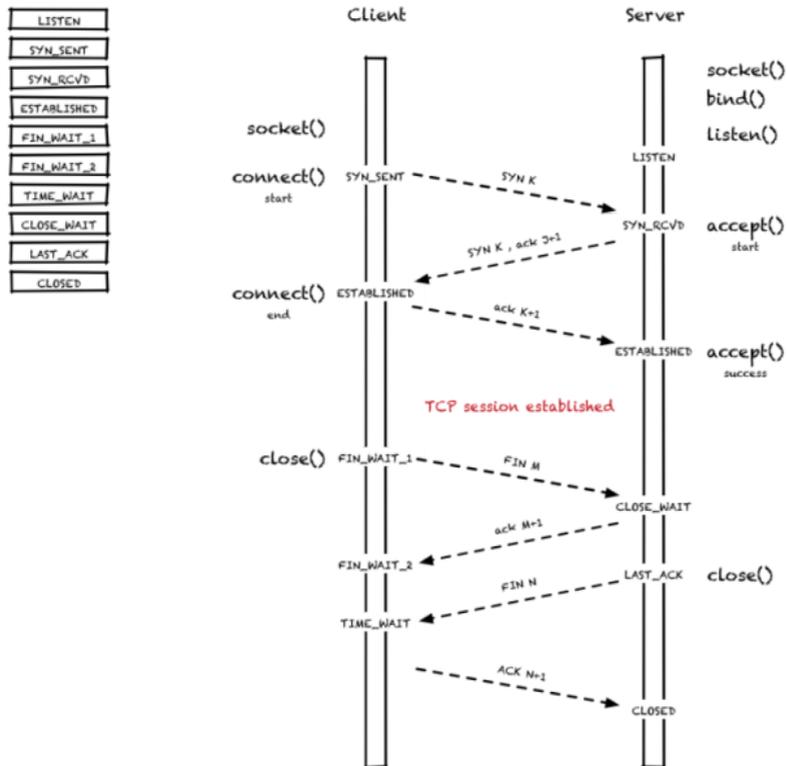
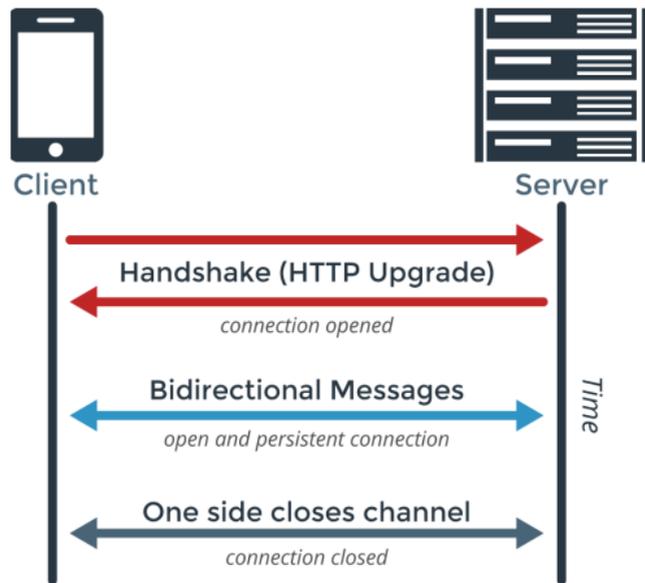


Abb. 2. TCP Kommunikation ist zustandsbasiert und hat ein striktes Ablaufprotokoll

Web Sockets

- HTTP bietet sogenannte "pull" Anfragen (GET/POST), auf die es eine Antwort gibt (response).
- Web Sockets bieten eine Zweiwege (bidirektionale) Kommunikation zwischen ursprünglich einem Server und einem Klienten (d.h. auch die andere Seite kann direkt Daten senden)
- Web Sockets werden über das HTTP(S) Protokoll verhandelt und eine Socket Verbindung aufgebaut
- Auch für die bidirektionale Kommunikation zwischen Web Browsern geeignet
- Web Sockets nutzen (zwei) TCP Verbindungen um Datenströme zu übertragen. Die Übertragung ist als zuverlässig anzusehen (anders als bei UDP)

Web Sockets



[www.pubnub.com]

Abb. 3. Aufbau eines Web Sockets über eine HTTP(S) Verbindung

Channels

- Ein Kommunikationskanal zwischen zwei Prozessen kann
 - unidirektional (ein Sender, ein Empfänger);
 - einseitig bidirektional (ein Sender und Empfänger einer Rückantwort, request-reply);
 - beidseitig bidirektional sein (zwei getrennte Sender und Empfänger).
- Ein Kommunikationskanal basiert auf einer FIFO Warteschlange (Queue)

Channels

- Es gibt zwei wesentliche Operation auf Kanälen:

read → *data*

Liest ein Datenelement aus der Warteschlange; wenn die Queue leer ist blockiert die Operation

write(*data*)

Schreibt ein Datenelement in die Warteschlange; wenn die Queue "voll" ist blockiert die Operation

Channels

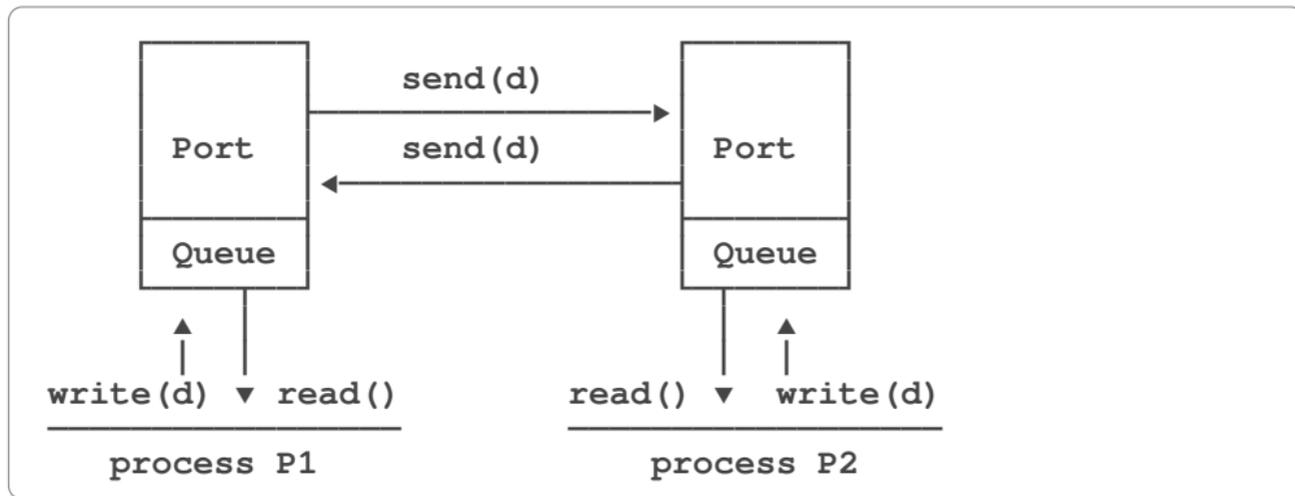


Abb. 4. Kanalkommunikation zwischen zwei (entfernten) Prozessen via Socket Ports

Netzwerkkommunikation

- Netzwerkkommunikation findet i.A. zwischen Prozessen und nicht Geräten statt
- Dazu werden Kommunikationsports auf einem Gerät verwendet (Sockets)
- Ein Kommunikationsport wird in IP (Internet Protocol) Netzen durch das Tupel $\langle ipaddr, ipport \rangle$ referenziert
- Genauer ist es ein Tripel $\langle ipaddr, ipport, ipproto \rangle$ um das Protokoll mit einzubeziehen (z.B. tcp)
- Ein Gerät (Hostrechner) hat mindestens eine wenigstens lokal eindeutige IP(4) Adresse ip , im Format "XX:XX:XX:XX" (X: hexadezimal Ziffer), oder "DDD:DDD:DDD:DDD".

Öffentliche und Private Netzwerke

- Ein Kommunikationsport muss mit seiner Adresse (*ipaddr,ipport*) öffentlich im Internet sichtbar sein (man spricht von einem Server)
- Aber: Schon allein aufgrund der hohen Anzahl von Geräten reicht der IP4 Adressraum nicht mehr aus, und jeder öffentlich erreichbare Rechner ist ein Angriffspunkt
- Es werden private/virtuelle Netzwerke aufgespannt mit zwei Adressräumen:
 - Öffentlich → Das ganze virtuelle Netzwerk hat eine einzige einmalige IP4 Adresse!
 - Virtuell → Jedes Gerät hat innerhalb des Subnetzwerkes eine eindeutige nicht öffentlich sichtbare Geräteadresse
- Problem: Kommunikation nach außen erfordert **Network Address Translation (NAT)**

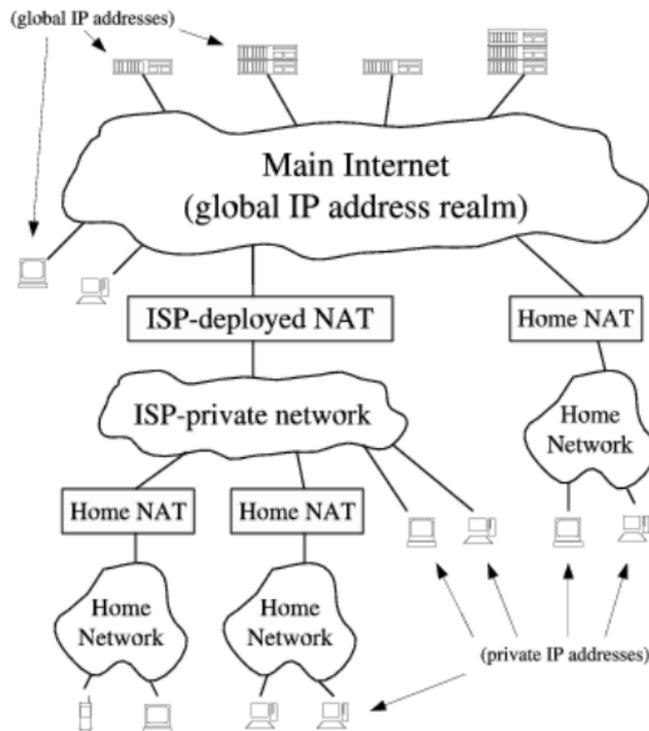


Abb. 5. Öffentliche und private Netzwerke mit unterschiedlichen Adressräumen und Adressumrechnung (NAT)

Network Address Translation

- Immer wenn eine Kommunikation zwischen zwei Kommunikationsports zweier Prozesse auf zwei Geräten in unterschiedlichen Netzwerken stattfinden soll muss an der Grenze Öffentliches Internet ↔ Virtuelles Netzwerk die private Adresse $\langle ipaddr, ipport \rangle$ in eine öffentliche $\langle V2P(ipaddr), V2P(ipport) \rangle$ umgerechnet werden
- Man unterscheidet:
 - Symmetrisches NAT: IP Adresse $ipaddr$ und auch die Portnummer ($ipport$) werden transformiert
 - Asymmetrisches NAT: Nur die IP Adresse $ipaddr$ wird umgerechnet, die Portnummer bleibt unverändert.
- Auch durch NAT werden aber Ports in virtuellen Netzwerken öffentlich nicht sichtbar.
 - *Hier fangen jetzt die Probleme für Kanalkommunikation via UDP/TCP an!!!*

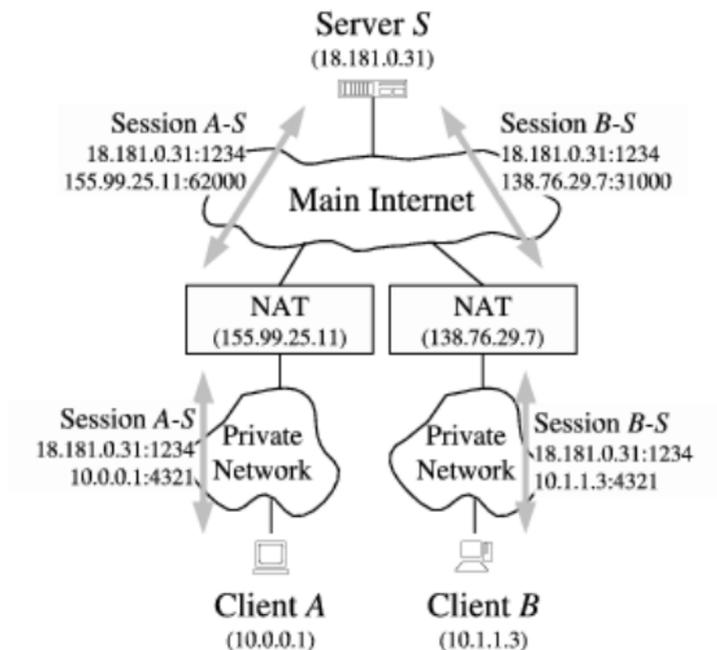


Abb. 6. Kanalkommunikation zwischen zwei (entfernten) Prozessen via NAT

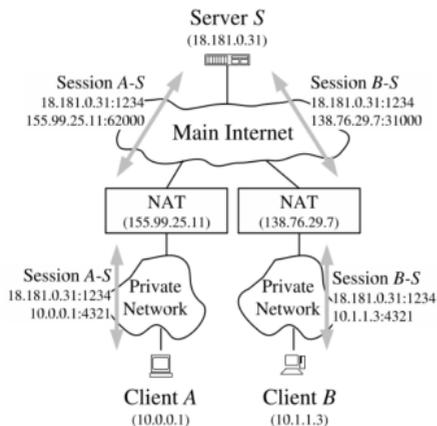
NAT Durchdringung



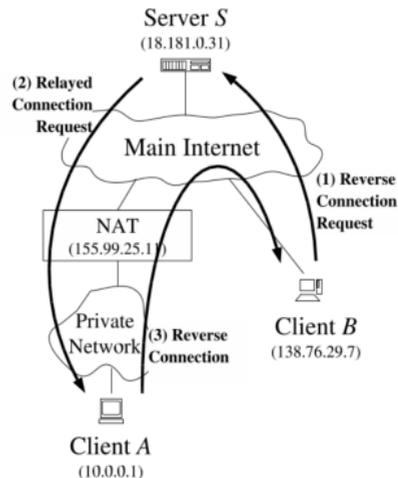
Wie können Prozesse über NAT kommunizieren?

[Peer-to-Peer Communication Across Network Address Translators, Bryan Ford, Pyda Srisuresh]

Relais



Verbindungsumkehr



NAT und UDP Hole Punching

<https://bford.info/pub/net/p2pnat/>

- Das Problem: Das öffentlich sichtbare Tupel $\langle ipaddr, ipport \rangle$ eines Peer Ports muss auch für eingehende Datenströme verwendet werden.
- Wenn ein Klient Daten über einen Port mit NAT sendet merkt sich der NAT das Mapping und eine Rücksendung an den Sendeport wird an das richtige Gerät und den Prozess vermittelt.
- Aber wie soll Klient A mit Klient B Kontakt aufnehmen wenn zuvor Klient B keinen Kontakt mit A aufgenommen hat? Es muss versucht werden den NAT von B "durchzuschalten"



Dazu werden (leere Ping) UDP Pakete zwischen den Klienten gegenseitig zugesendet.

NAT und UDP Hole Punching

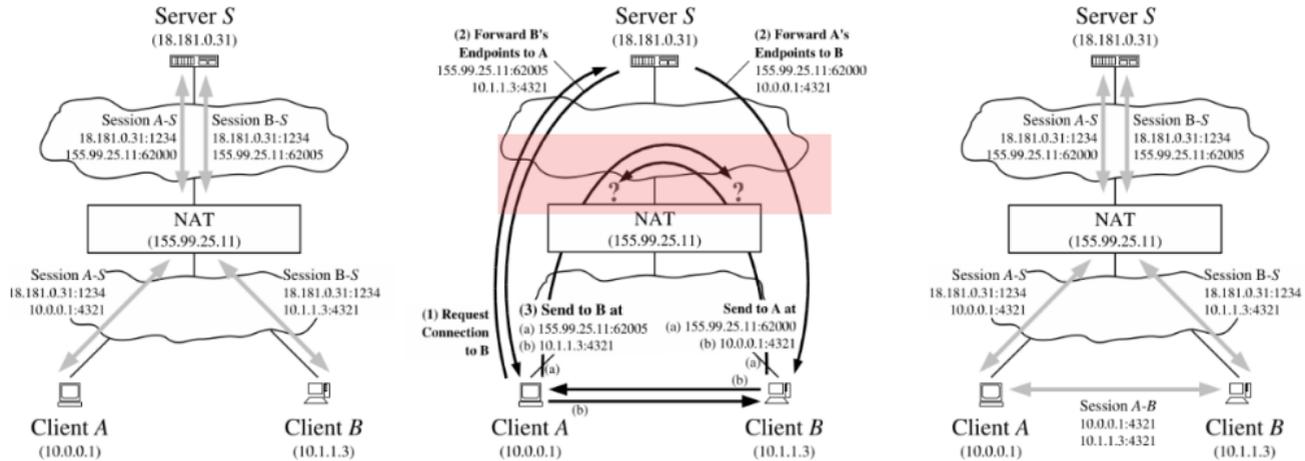


Abb. 7. UDP Hole Punching, Peers hinter gemeinsamen NAT

NAT und UDP Hole Punching

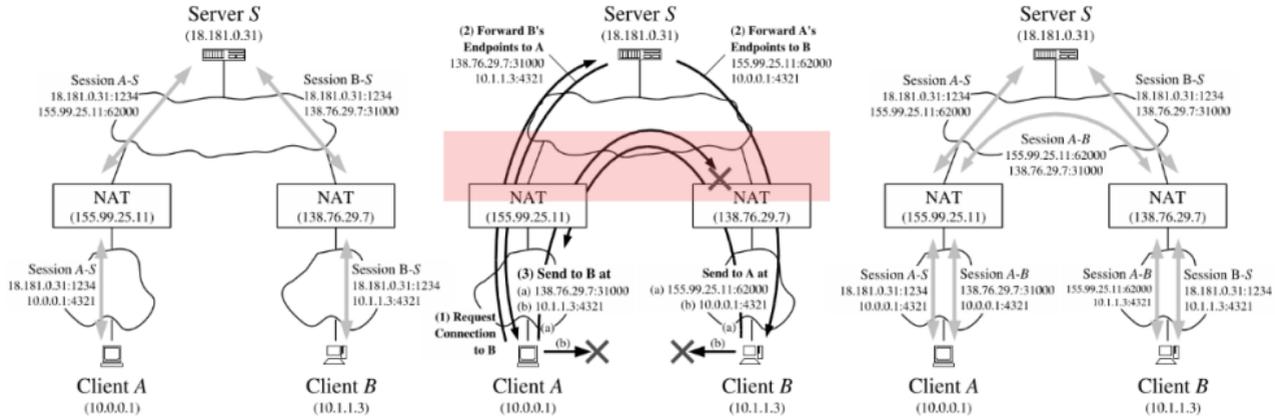


Abb. 8. UDP Hole Punching, Peers hinter getrennten NAT

Signaling Server

<https://support.medialooks.com/hc/en-us/articles/360000213312-%D0%95nvironment-signaling-STUN-and-TURN-servers>

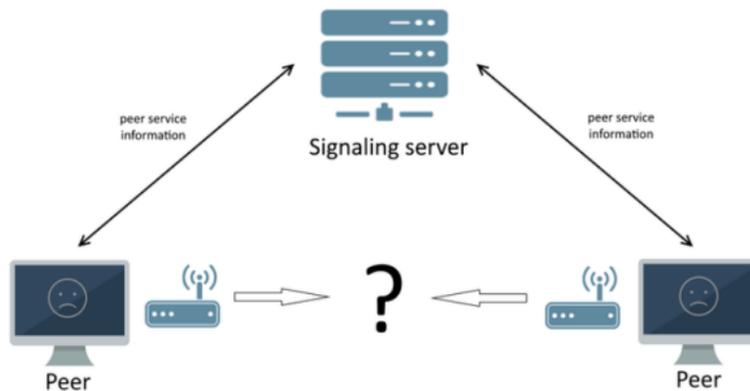


Abb. 9. Signaling server werden benutzt damit Peers Service Informationen teilen können, um eine direkte Peer-to-Peer-Übertragung zu starten.

STUN, TURN

https://help.estos.com/help/de-DE/procall/7/erestunservice/dokumentation/htm/IDD_FUNCTIONALITY.htm

Signaling Server

[help.estos.com]

Signaling Server dienen zum indirekten Austausch von Daten zwischen zwei Klienten. Dies kann ein Dienst sein, der von beiden Klienten erreichbar ist oder auch mehrere Dienste die mittels Zusammenschluss miteinander verbunden sind

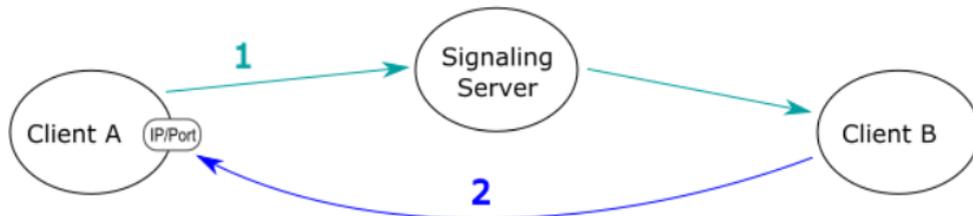


Abb. 10. Klient A ist direkt erreichbar. Klient B kann Daten direkt an Klient A senden

STUN, TURN

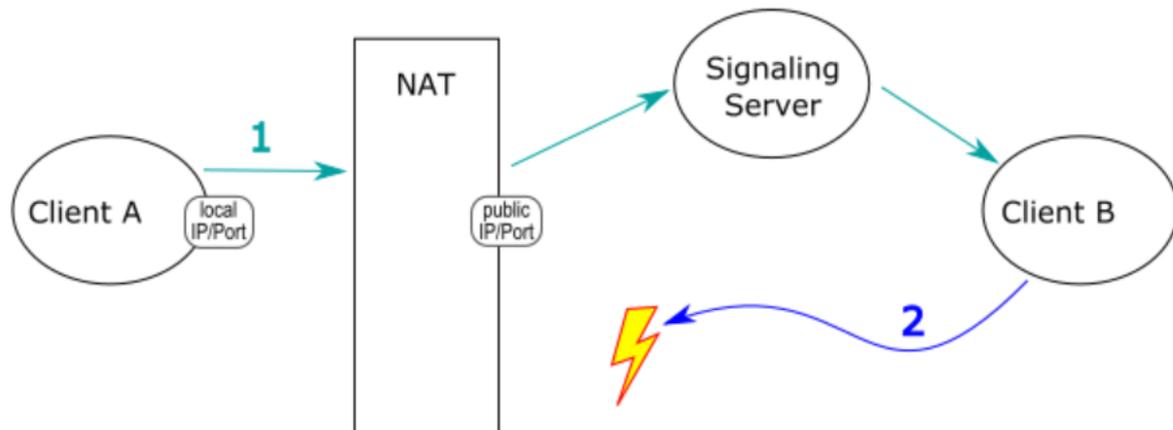


Abb. 11. Erfolgreicher Verbindungsaufbau über einen NAT-Router hinweg.

STUN - Session Traversal Utilities for NAT (RFC5389)

Dieses Protokoll ermöglicht es einem Klienten in einem lokalen Netzwerk (LAN), seine eigene, öffentliche IPv4-Adresse zu ermitteln. Der rufende Client im LAN kann auf diese Weise dem angerufenen Klienten außerhalb des LAN mitteilen, welche IPv4-Adresse (und Portnummer) verwendet werden kann um eine direkte Kommunikation mit ihm zu ermöglichen ("Peer-to-Peer" Verbindung).

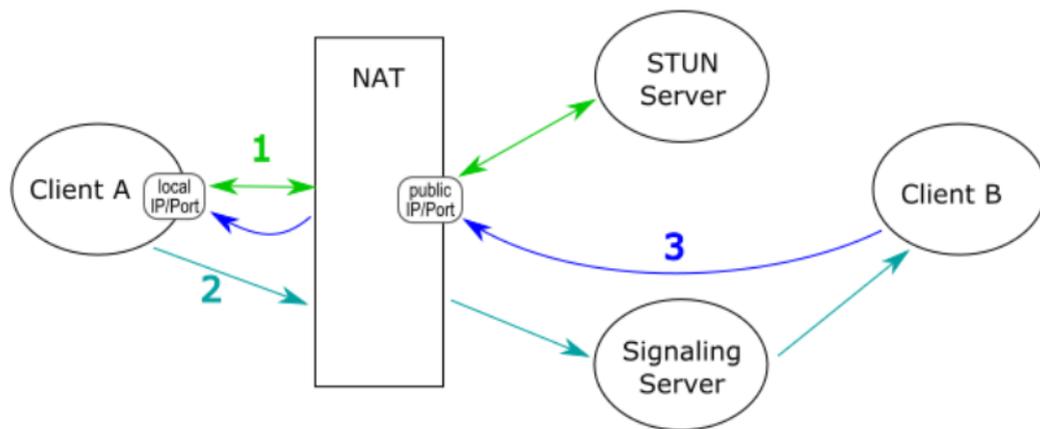


Abb. 12. Erfolgreiche Kommunikation unter Zuhilfenahme eines STUN-Servers.

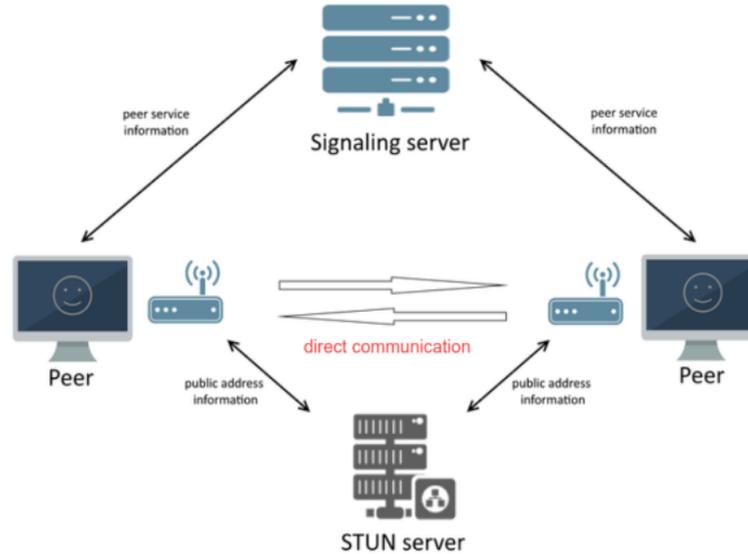


Abb. 13. Verwendung eines STUN Servers um öffentliche IP Adressen der Rechner zu bestimmen.

STUN, TURN

TURN - Traversal Using Relays around NAT (RFC5766)

Ein Server im Internet, der das TURN-Protokoll implementiert, ermöglicht es zwei Klienten, Daten ohne eine direkte Verbindung auszutauschen ("Relais Server"). Dies wird notwendig, wenn es keine Möglichkeit gibt, eine direkte Klient-zu-Klient-Verbindung aufzubauen.

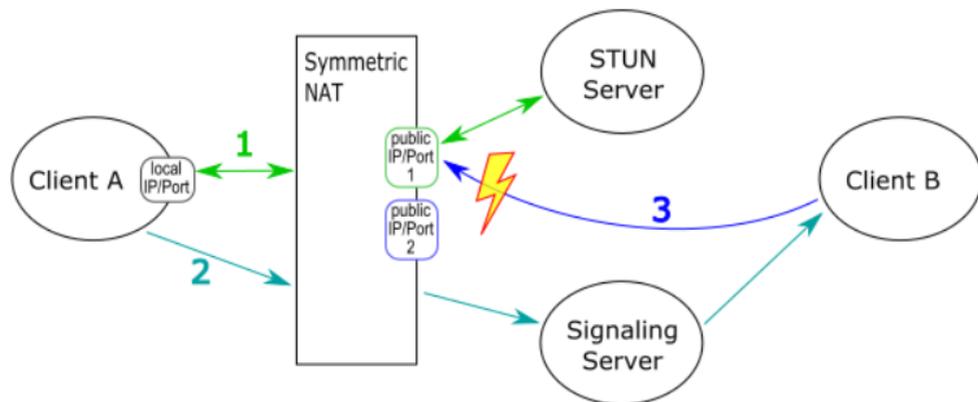


Abb. 14. Erfolgreicher Kommunikationsversuch über ein "Symmetric NAT".

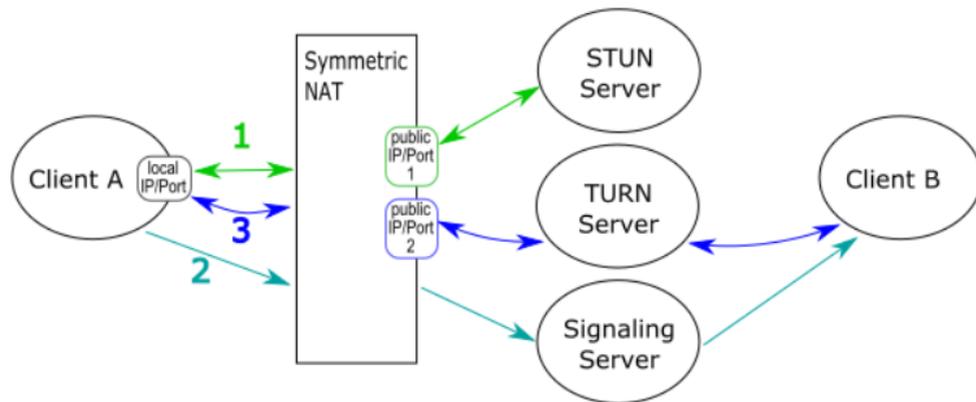


Abb. 15. Erfolgreicher Kommunikationsversuch über ein "Symmetric NAT" durch Nutzung eines TURN-Servers.

ICE - Interactive Connectivity Establishment (RFC5245)

Zwei Klienten können die mit Hilfe von STUN und TURN ermittelten Verbindungsinformationen (und andere Daten) mit Hilfe des ICE Protokolls austauschen. Die Übermittlung der Informationen muss dabei über einen eigenen Dienst erfolgen, einen sog. "Signaling Server". Dieser Dienst muss von beiden Klienten erreichbar sein.

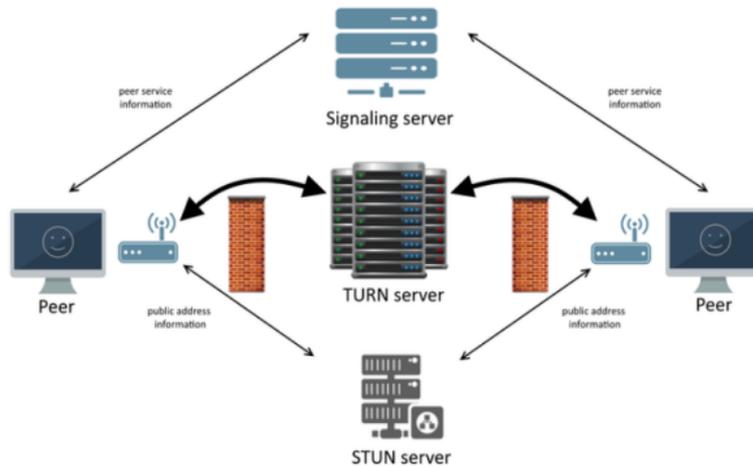


Abb. 16. TURN Server als Relais um (sym.) NAT zu überwinden. Jeder Klient muss mit dem TURN Server verbunden sein.

STUN, TURN

Zusammenfassung

- Der Signalisierungsserver wird verwendet, um Peers die Teilung von Dienstinformationen zu ermöglichen, um die Peer-to-Peer-Stream-Übertragung zu starten.
- Der STUN-Server wird verwendet, um Peers die externe Adresse des anderen mitzuteilen, um die Peer-to-Peer-Stream-Übertragung hinter dem NAT zu starten.
- TURN Server wird verwendet, um Peers Streams hinter einer Firewall miteinander übertragen zu lassen. In diesem Fall findet keine Peer-to-Peer-Stream-Übertragung statt. Der gesamte Medienverkehr läuft über einen TURN-Server.

Symmetrisches NAT

- Kaum für direkte Peer-to-Peer Verbindungen einzusetzen (wenn beide Prozesse hinter NATs sitzen)
 - Typisch für mobile Netzwerkverbindungen (4G/LT/UMTS)
- Die lokalen und öffentlichen Portnummern unterscheiden sich
 - Häufig ein Paar für jede ausgehende und eingehende Datenübertragung (bei gleichen lokaler Portnummer!)
- Einige UDP Hole Punching Algorithmen versuchen das Transformationsschema zu untersuchen. Randomisierte Zuordnungen sind kaum durchdringbar, inkrementelle mittels Probing eventuell!

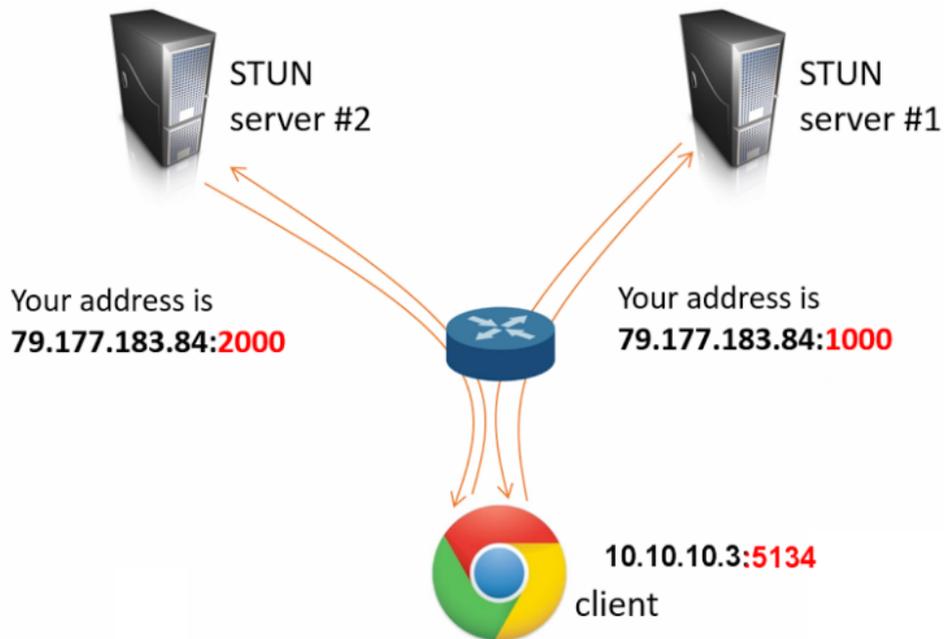


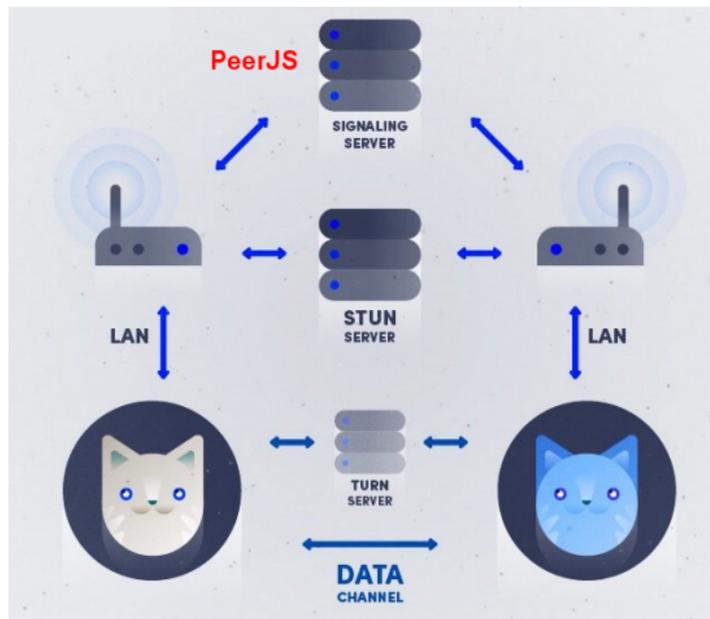
Abb. 17. Hier ist das Problem bei einem sym. NAT gezeigt: Der Klient fragt zwei STUN Server nach seinem öffentlichen Port Tupel und bekommt zwei verschiedene Antworten. Und von außen kann über dieses Mapping auch keine Nachricht an den Port gesendet werden!

PeerJS

<https://github.com/peers/peerjs>

- PeerJS besteht aus einer Klienten API und einem Server, dem Signaling Server
- PeerJS kapselt Web Sockets mit Web RTC (Realtime Comm.)
- Ein Peer Port in PeerJS ist mit einer (eindeutigen) ID Nummer verknüpft, z.B. `a53d9c76-4119-4d82-be5c-88fc11ffc943`
 - Ein Peer Port ist (unter Verwendung von STUN und TURN Servern) im ganzen Internet erreichbar
 - Beliebig viele Verbindungen von anderen Klienten können aufgebaut werden
 - Damit die ID eindeutig ist wird ein PeerJS Server für die Anfrage und Verwaltung verwendet.
 - Jedoch kann auch eine feste ID verwendet werden, so z.B..

PeerJS



[toptal.com]

Abb. 18. Alle Server zusammen: STUN+TURN+PeerJS

Web Browser-Browser Cluster

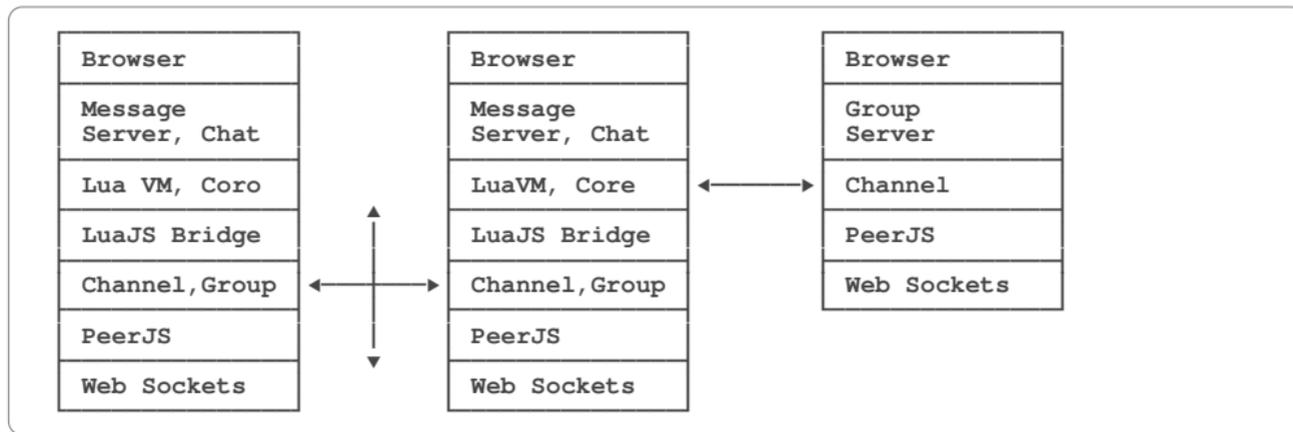


Abb. 19. Browser Cluster mit Gruppenserver (Master und Koordinator)

Lua Web Gruppenkommunikation

- Es wird folgend aufgebauter Kommunikationsstack im Web Browser verwendet:
 - Group (Lua/JS)
 - Channel (Lua/JS)
 - Socket (JS)
 - PeerJS (JS)
 - Web Sockets (JS, Browser API)
 - HTTP/HTTPS
- Es werden folgende Server verwendet:
 - PeerJS (Signaling Server)
 - STUN (IP Adressen und Port Resolver)
 - TURN (IP Relais bei symmetrischen NATs)
- Es wird die fengari Lua VM für Programmausführung verwendet

Lua Group API

```
iceServers = [
  { url: 'stun:stun.l.google.com:19302' },
  { url: 'turn:numb.viagenie.ca',
    credential: 'muazkh', username: 'webrtc@live.com'
  }
]
group = Group(mygroup) -- Gruppe erzeugen bzw. bekanntgeben
status = group:join() -- Der Gruppe beitreten
status = group:unjoin() -- Die Gruppe verlassen
members = group:members() -- Liefert alle momentanen Gruppen
-- Mitglieder
chan = group:connect(member) -- Kanal zu member ∈ members aufbauen
chan:write(data) -- In den Kanal schreiben
data=chan:read(timeoutInMs?) -- Aus dem Kanäle lesen
```

Def. 1. Lua Web Socket Gruppenkommunikation

- Alle Operationen sind synchron und blockierend bis entweder die Operation erfolgreich ausgeführt wurde oder abgebrochen wurde (Rückgabe i.A. nil).

Zusammenfassung

Synchronisiert Kommunikationskanäle (Channels) stellen wichtigstes Interprozesskommunikation in parallelen und verteilten Systemen dar

Channels in lokalen parallelen Systemen sind "einfach" zu implementieren

Channels über nachrichtenbasierte Netzwerkverbindungen können einen hohen Aufwand und Kosten verursachen

Verbindung von Netzwerkprozessen in unterschiedlichen privaten/virtuellen Netzwerken erfordern Hilfsserver (Signaling, STUN, TURN)