Verteilte und Parallele Programmierung

Mit Virtuellen Maschinen

PD Stefan Bosse

Universität Bremen - FB Mathematik und 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 Datenauschtausch);
  - Semaphoren (Synchronisation, Kooperation, Schutz), häugfig 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 Rechner mit Nachrichtenkommunikation implementiert werden!
  - Channels können dann genutzt werden um andere Synchronisationsobjekte (z.B. Semopahore) 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 zuverläassig)
  - 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 vorher ein 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

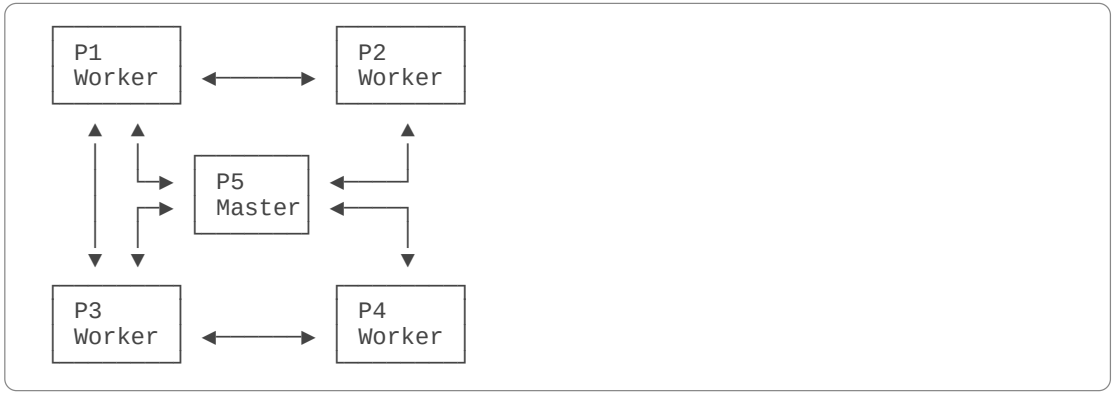


Abb. 1. Verteiltes Prozesssystem mit Leader(Master) und Workern und Kommunikationskanälen

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 server = luv.net.tcp()
server:bind(host, port)
 function on_conncetion(chanClient)
  -- handle client request
  client:write(data)
end
server:listen(128, function(err)
  -- Accept the client
  local chanClient = luv.net.tcp()
  server:accept(chanClient)
  on_connection(chanClient)
end)
```

Klient

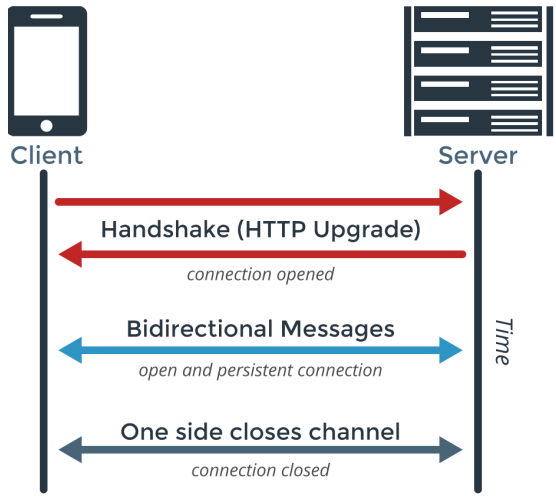
```
local chanClient = luv.net.tcp()
chanClient:connect(host, port,
  function (err)
    -- check error and carry on.
    chanClient:write(data)
    data=chanClient:read()
end)
```

Bsp. 1. Lua Socket mit bidirektionaler TCP Kanalverbindung

Web Sockets

- HTTP bietet sogenannte "pull" Anfragen (GET/POST), auf die es eine Antwort gibt (response).
- WEB Sockets bieten eine Zweiweg (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 zuverläassig anzusehen (anders als bei UDP)

Web Sockets



[www.pubnub.com]

Abb. 2. 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 Qeuee "voll" ist blockiert die Operation

Channels

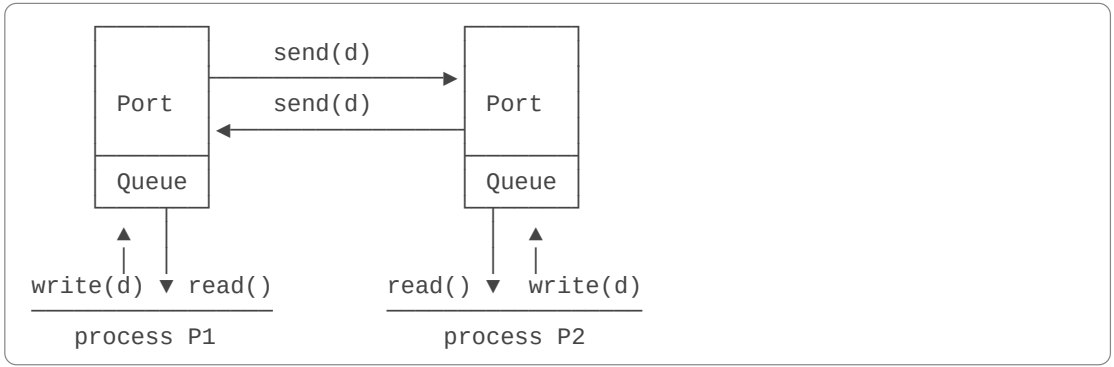


Abb. 3. Kanalkommunikation zwischen zwei (entfernten) Prozessen

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 ⟨*ipaddr*,*ipport*⟩ referenziert
- 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 eintige IP4 Adresse!
  - Virtuell → Jedes Gerät hat innerhalb des Subnetzwerkes eine eindeutige nicht öffentlich sichtbare Geräteadresse
- Problem: Kommunikation nach außen erfordert ***Network Address Translation*** (**NAT**)

[bford.info/pub/net/p2pnat]

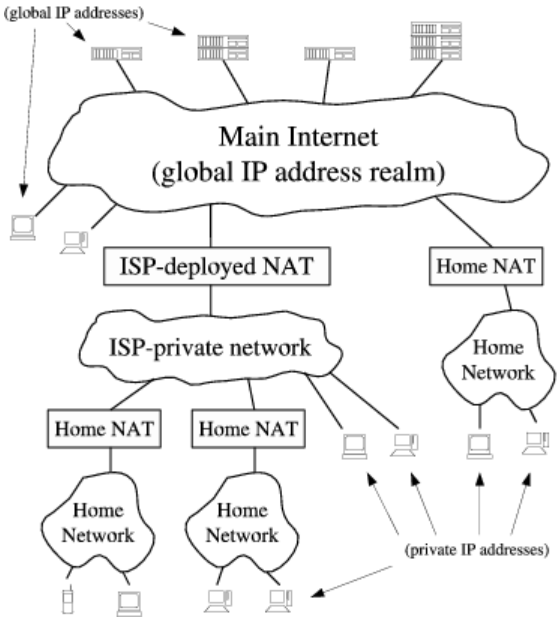


Abb. 4. Öffentliche und private Netzwerke mit unterschiedlichen Adressräumen und Adressumrechnung (NAT)

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 $\leftrightarrow$ 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 Addresse *ipaddr* und auch die Portnummer (*ipport*) werden transformiert
  - Asymmetrisches NAT: Nur die IP Addresse *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*!!!

[bford.info/pub/net/p2pnat]

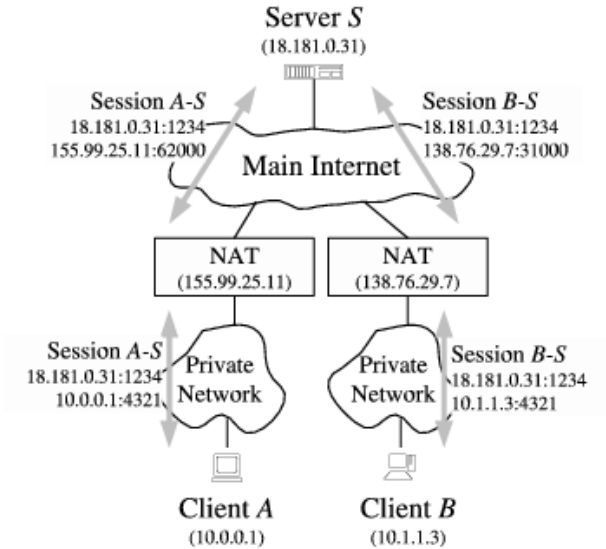


Abb. 5. Kanalkommunikation zwischen zwei (entfernten) Prozessen

NAT und UDP Hole Punching

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

- Das Problem: Das öffentlich sichtbare Tupel ⟨*ipaddr*,*ipport*⟩ 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.

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

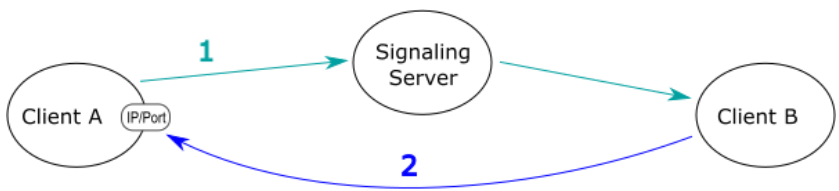


Abb. 6. Klient A ist direkt erreichbar. Klient B kann Daten direkt an Klient A senden

STUN, TURN

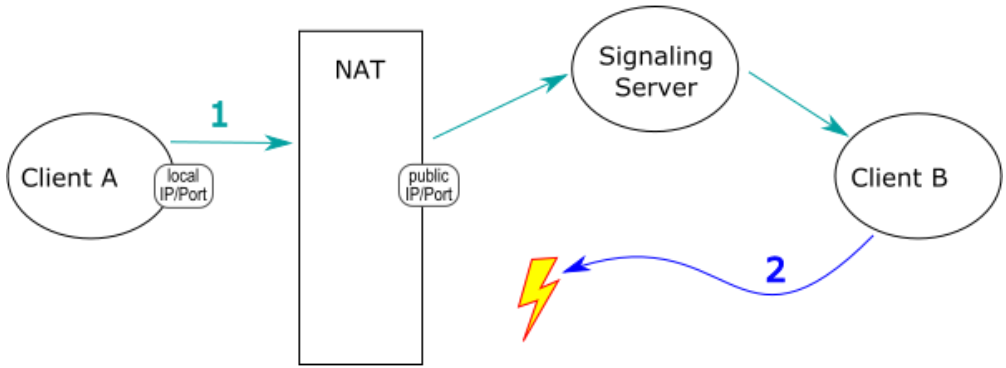


Abb. 7. Erfolgloser Verbindungsaufbau über einen NAT-Router hinweg.

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).

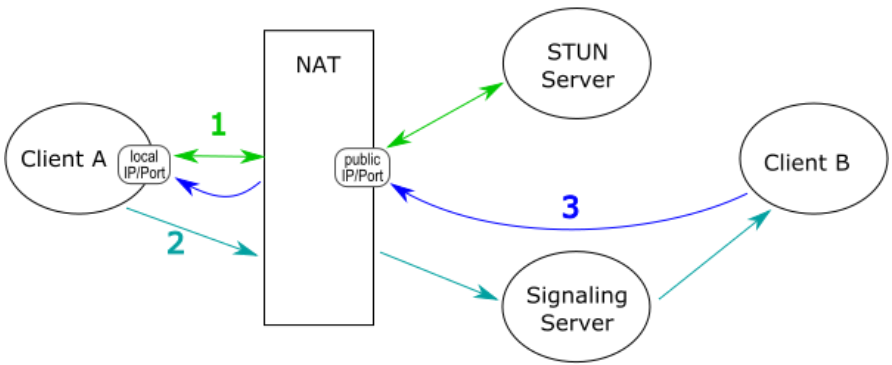


Abb. 8. Erfolgreiche Kommunikation unter Zuhilfenahme eines STUN-Servers.

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.

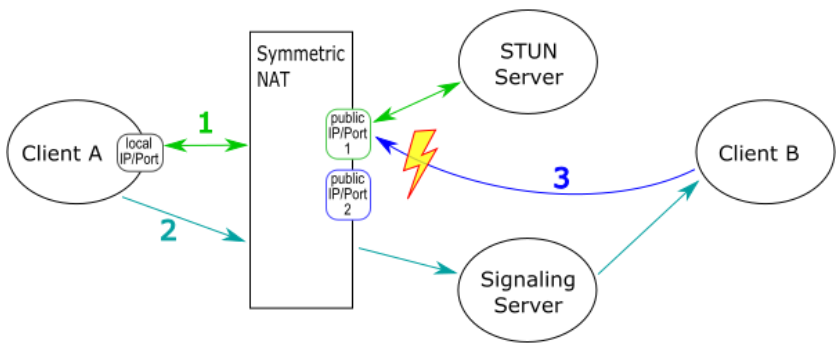


Abb. 9. Erfolgloser Kommunikationsversuch über ein "Symmetric NAT".

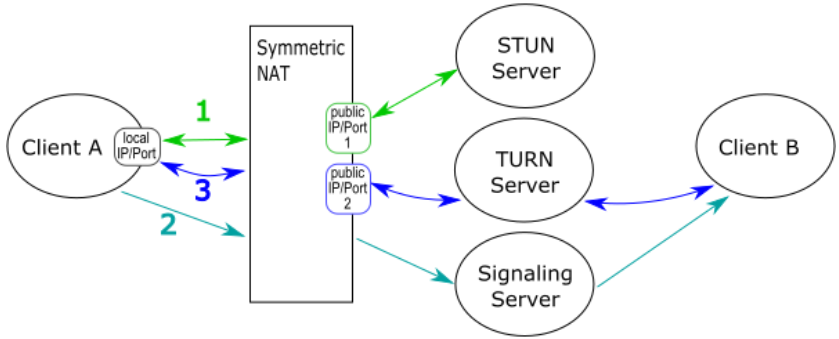


Abb. 10. Erfolgreicher Kommunikationsversuch über ein "Symmetric NAT" durch Nutzung eines TURN-Servers.

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.

Symmterisches 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!

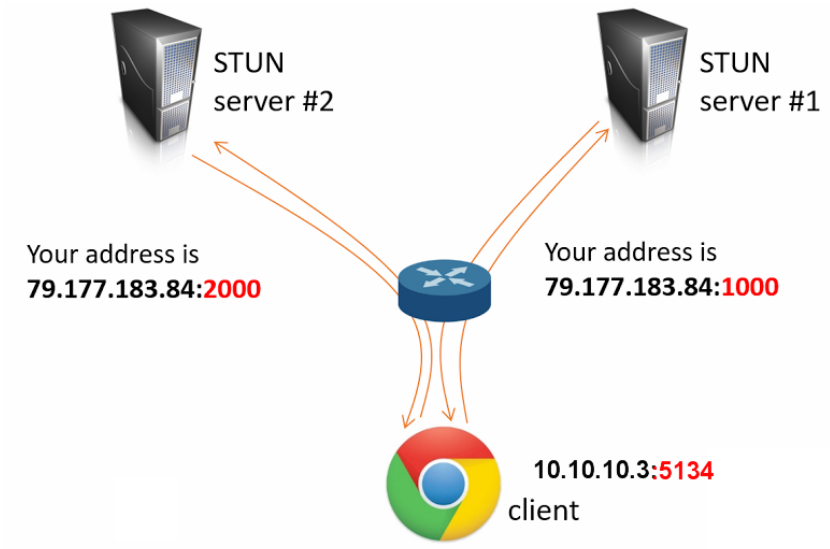


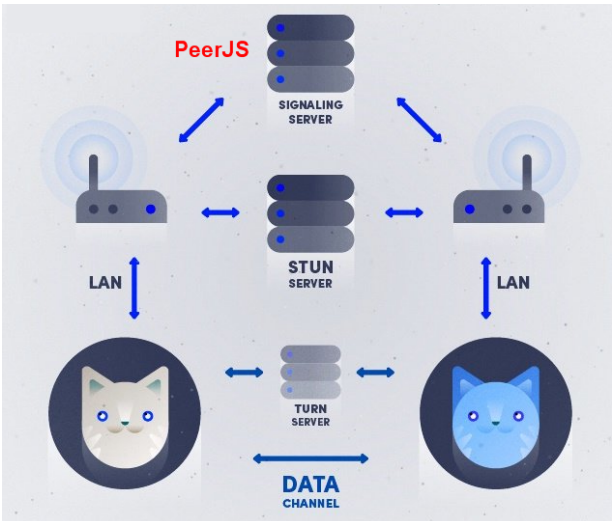
Abb. 11. Hier ist das Problem bei einem sym. NAT gezeigt: Der Klient fragt zwei STUN Server nach seinem öffenteln Port Tupel und bekommt zwie verschiedene Antworten. Und von außen kann über dieses Mapping auch keine Nachricht an den Port gesendet werden!

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 werdem
  - 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. 12. Alle Server zusammen: STUN+TURN+PeerJS

WEB Browser-Browser Cluster

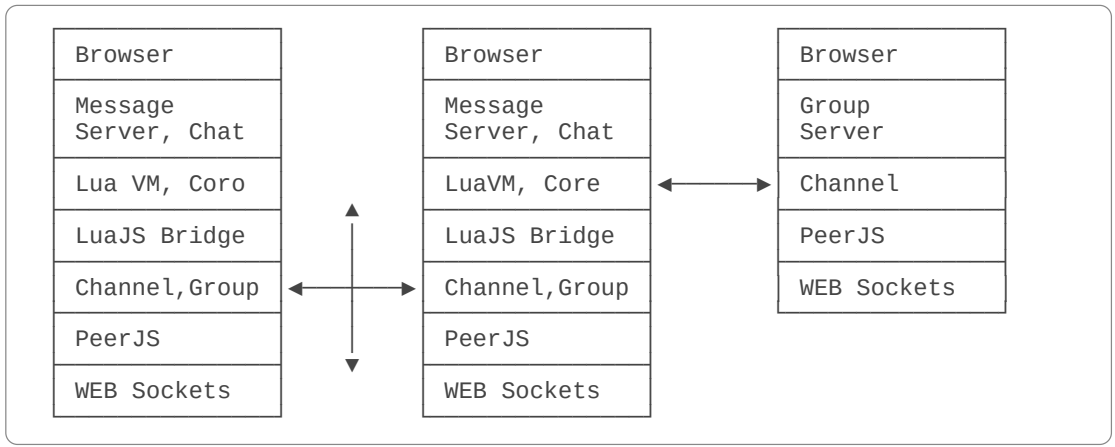


Abb. 13. Browser Cluster mit Gruppenserver (Master und Koordinator)

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 Realais 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 Kanale 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 wichtigistes 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

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