# Usurp: Distributed NAT Traversal for Overlay Networks

Salman Niazi, Jim Dowling

## To cite this version:

Salman Niazi, Jim Dowling. Usurp: Distributed NAT Traversal for Overlay Networks. 11th Distributed Applications and Interoperable Systems (DAIS), Jun 2011, Reykjavik, Iceland. pp.29-42, 10.1007/978-3-642-21387-8_3 . hal-01583583

HAL Id: hal-01583583
https://inria.hal.science/hal-01583583

Submitted on 7 Sep 2017

# Usurp: Distributed NAT Traversal for Overlay Networks

Salman Niazi and Jim Dowling

Swedish Institute of Computer Science (SICS)

**Abstract.** Many existing overlay networks are not practical on the open Internet because of the presence of Network Address Translation (NAT) devices and firewalls. In this paper, we introduce Usurp, a message routing infrastructure that enables communication between private nodes (behind NATs or firewalls) either by direct connectivity or relaying messages via public nodes (nodes that support direct connectivity). Usurp provides fully distributed NAT-type identification and NAT traversal services using a structured overlay network (SON) built using the public nodes in the system. Private nodes do not join the SON, instead, each private node is assigned a key in the SON's address space and the public node(s) responsible for its key acts as both a rendezvous and relay server to the private node. Usurp is designed as a middleware that existing overlay networks can be built over, enabling them to function correctly in the presence of NATs. We evaluate Usurp using a gossip-based peer sampling service (PSS). Our results show that the PSS running over Usurp preserves its randomness properties and remains connected even in scenarios with high churn rates and where 80% of the nodes are behind NATs. We also show that Usurp only adds a low and manageable overhead to public nodes.

## 1 Introduction

Many elegant distributed algorithms for constructing overlay networks are not practical over the open Internet because of the presence of ugly Network Address Translation (NAT) devices and firewalls. For example, gossiping, a widely used technique for building overlay networks, assumes that any pair of nodes can communicate directly with each other, whereas, in reality, *private nodes* behind NATs do not support direct connectivity with nodes outside their private network. This results in an uneven participation of nodes in gossiping, where *public nodes* (with open IP addresses) have a significantly higher network traffic burden [17,18]. Systems studies have shown that in existing peer-to-peer (P2P) systems only between 20-40% of nodes are public nodes [13,22].

NAT traversal protocols are required to communicate with private nodes, except in the case where the source node resides behind the same NAT. Centralized NAT traversal services are commonly used in existing P2P systems [23]. These include STUN (Session Traversal Utilities for NAT) [19,20] that identifies a node's NAT type, and *relay* and *rendezvous* services that, respectively, forward packets to the private node and enable direct connectivity to the private node in a process commonly known as *hole punching*. Protocols for hole punching do not work for all combinations of NAT types. Depending on the distribution of NAT types in the system, hole-punching for UDP works for 80%-95% of NATs [22,6], and around 52% for TCP [10].

In this paper, we present the first fully distributed NAT identification and traversal protocols. We use these protocols to build Usurp, a NAT-friendly overlay network, that enables any two nodes on the open Internet to communicate, regardless of whether they are public or private. In Usurp, all public nodes join a structured overlay network (SON). Each private node is assigned a unique address in the SON's address space and the public node responsible for that SON address acts as a relay and rendezvous server for the private node. Relay and rendezvous services enable indirect and direct connectivity with private nodes, respectively. All public nodes also provide a NAT-type identification service that enables newly joined nodes to determine whether they reside behind a NAT or not, and what the type of that NAT is. To reduce connection latency using the SON, we introduce a caching mechanism that preserves useful information for future session establishment and reduces the need for lookups on the SON.

Usurp is implemented as a middleware that can be layered below existing overlay network protocols. We introduce an address structure for connecting to both public and private nodes that includes a key in the SON address space, the node's NAT type, and a set of IP addresses (the node's own IP address for public nodes and the address of its *parent(s)* for private nodes). A parent address is an IP address of a public node on the SON responsible for a private node. When a node attempts to connect to a private node, it can first attempt to connect via its parents (in parallel), if it fails then it falls back to the SON to find an active parent. This significantly reduces the need to perform lookups on the SON, and is particularly effective where either public nodes are long-lived or where addresses are quickly expired from the system. Usurp also enables the construction of NAT-aware applications, enabling nodes to send private nodes either small messages with lower latency using relaying (e.g., control packets) or larger messages via a direct connection, but incurring higher latency due to the overhead of hole punching.

We have validated and evaluated Usurp by constructing a gossip-based peer sampling service (PSS) on top of Usurp. Our results show that Usurp enables the PSS to preserve its randomness properties and connectivity even in scenarios with churn rates of 80% and where up to 80/90% of the nodes are behind NATs. For the PSS, we show that Usurp adds only a low and manageable overhead to public nodes.

## 2 NAT classification and traversal

The type of NAT a private node resides behind is important in determining what NAT traversal mechanism to use when communicating with that private node. The original Session Traversal Utilities for NAT (STUN) protocol [20] provides a limited categorization of NATs into one of four types: full-cone, address-restricted cone, port-restricted cone, and symmetric. We adopt a richer classification of NAT types introduced by Roverso in [22], based on the BEHAVE RFC [2] and [14], that classifies a NAT by its *port mapping*, *port allocation* and *port filtering policies*. The port mapping policy defines when a NAT receives an outgoing packet from a private node whether it allocates a new port or uses an existing port on its external public interface. The port allocation policy defines which port should be allocated on the NAT for an outgoing packet when a new mapping is created on the NAT. Finally, the port filtering policy determines whether the NAT forwards an incoming packet to a private node or not, depending on the existing mappings in the NAT and the source IP address and port of the incoming packet. Classical STUN can only accurately determine the filtering policy. We use a modified version of STUN protocol, based on [30] and [22], to determine all three policies. Another difference with STUN is that classical STUN servers require two different public IP addresses. However, most nodes in P2P systems do not have two different public IPs. As such, we use pairs of public nodes to implement a distributed STUN service (DSTUN). Each public node maintains a list of partner STUN nodes, sampled from the SON and ordered by round-trip time (RTT), so whenever a DSTUN server has to send a reply from an different IP address, it simply requests its lowest RTT partner to send the reply. Note that DSTUN does not consider dynamic and multi-layer NATs, more commonly found in corporate networks [6]. We do, however, support UPnP port mapping for NATs [27].

Usurp supports NAT traversal by establishing direct connections using hole-punching for UDP, and where not possible, relaying messages to private nodes using public nodes. We do not support hole-punching using TCP [8] due to its significantly lower success ratio. We support a suite of hole-punching algorithms and the NAT type of both the source and destination nodes is used to determine the traversal technique required to establish a connection between two nodes. When hole-punching is not supported for the combination of the two NAT types we revert to relaying. The hole-punching techniques we support include simple hole punching, port prediction using preservation, and port prediction using contiguity. All of these techniques use a public node acting as a rendezvous server to coordinate the protocol, and vary in how they generate a NAT mapping that will allow traffic to pass through the NAT, and, thus, establish a direct connection. More details on these algorithms can be found in [22].

## 3 Usurp SON

On joining Usurp, a node discovers a number of random public nodes using a bootstrap service. The node then pings these public nodes and runs our NAT-type identification protocol against the node with the lowest RTT. On discovering its NAT-type, the node will either join a SON if it is public, or put a value in the SON if it is a private node.

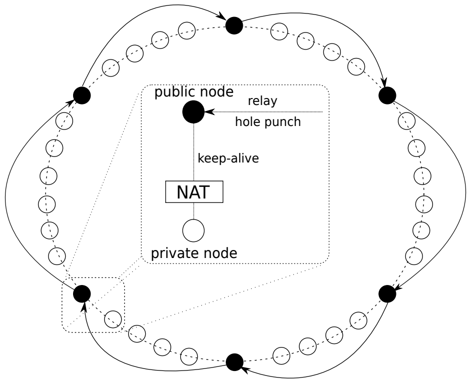
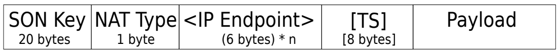


Fig. 1: Usurp's structured overlay network. Filled circles are public nodes, members of the SON. Empty circles are keys representing private nodes. Every private node keeps a NAT mapping open to the public node responsible for its key, so that the public node can handle relay and hole-punching requests for the private node.

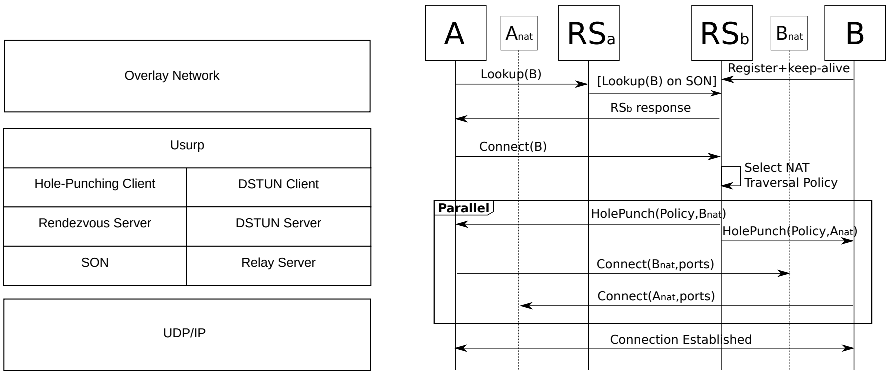


| SON Key<br>20 bytes | NAT Type<br>1 byte | <IP Endpoint><br>(6 bytes) * n | [TS]<br>[8 bytes] | Payload |
|---|---|---|---|---|

Fig. 2: Usurp node descriptor.

For public nodes in the SON, we generate an initial node-Id by hashing the node's public IP address, and then we replace the least significant 16 bits of the node-Ids with the port number. This limits a single public node's ability to mount a Sybil attack as nodes it produces from behind one IP address will most likely be contiguous on the overlay. We use iterative routing, as it has a lower hop count compared to recursive routing, and low latency is crucial for connection establishment. For private nodes, we generate a key by hashing its NAT's public IP address, and then we replace the least significant 16 bits with the last 16 bits of the private IP address. The private node then puts the key with its node descriptor into the SON and then performs $k$ lookups on the SON using the $k$ replication hash keys. The lookup responses return the $k$ public nodes responsible for the keys. The node then registers as a *child* of these parents and keeps the NAT mappings to the parents alive using *heartbeats*. When a public node leaves the SON, its children become children of the new public node responsible for the key-space. The heartbeat period is determined by the NAT mapping timeout, as measured by the NAT-type identification service. As it can take minutes to determine the NAT mapping timeout, the default heartbeat period is initially set to 30 seconds, the shortest NAT mapping timeout for UDP observed by [12], and later updated when the NAT mapping timeout is determined.

Our SON is based on Chord and Usurp's architecture is illustrated in figure 1. Although a lot of extensions have been proposed for Chord, such as biasing Id assignment to load balance data over nodes [25] and network-awareness to reduce latencies [31], we consider these issues to be outside the scope of this



(a) Modular view of Usurp.

(b) Hole-punching using the SON.

Fig. 3: Usurp middleware and hole-punching using the SON.

paper. However, one extension we provide that is an address caching mechanism to preserve connection information for future session establishment. Node descriptors for private nodes include references to their parent addresses, see figure 2. When a node wishes to relay a message or directly connect to a private node, it sends a message to the parents listed in the node descriptor, with fallback to the SON to lookup the active parent only when all parents listed in the node descriptor are not reachable (because the node's parents have changed since the node descriptor was published).

## 4 Connection establishment in Usurp

Usurp is implemented as a middleware and appears as a black box to higher-level overlay network protocols. Usurp takes messages from the upper overlay network layer, see figure 3a. Usurp does not require any change to overlay network protocols, apart from using the addressing scheme from figure 2. The only case where overlay protocols may have to be modified is if they are sensitive to connection setup times of up to a few seconds, as hole-punching may take that long to complete [22]. Figure 3a shows the modular view of our Usurp layer. It consists of DSTUN, hole-punching, relay and SON modules. Public nodes provide DSTUN, relay, hole punching and SON services, while both public and private nodes provide the DSTUN and hole-punching clients of these services.

When a node attempts to connect to a private node, both mechanisms for establishing a connection, hole-punching and message relaying, require establishing a connection to one of the private node's responsible public nodes, a rendezvous server (RVP). The private node must also have a valid NAT mapping for the same RS. In figure 3b, we can see how private node $A$ first looks up the public node $RS_B$, responsible for private node $B$. $A$ sends a connect message to $RS_B$, and $RS_B$ selects the appropriate NAT traversal policy, which is then sent to both private nodes $A$ and $B$. If hole-punching is supported, $A$ and $B$ execute the hole punching algorithm in parallel, sending possibly many packets to ports on $B_{nat}$

and $A_{nat}$, respectively, with the goal of inserting a mapping rule in either $B_{nat}$ or $A_{nat}$ that will allow a direct connection to be established between $A$ and $B$.

The complete Usurp protocol is defined in Algorithm 1. The first step nodes take when joining the system is to request a set of random public nodes from the bootstrap server. The client then pings these public nodes and runs the DSTUN protocol against the available node with the lowest RTT to identify its NAT-type, lines 7–14. UPnP enabled nodes can also act as public nodes. Instead of publishing their private address, they publish a mapped port and the public address of their NAT. If the node is public or supports UPNP port mapping, then it also initializes the DSTUN server and hole punching server modules; and sends a Join request to SON module, lines 16–22. For UPnP nodes, we need to map ports on the NAT, lines 17–18. Public nodes register with their own hole punching server module, line 22.

If the client is behind a NAT then it must register with a public node as its RVP. It performs a lookup for its id on the SON and registers with the public node returned, lines 24–25. Nodes may join or leave the system, causing the RVP responsible for a private node-id to change. Private nodes start a periodic timer to continuously look for any change in their RVP, line 26. If the periodic timer detects any change in a child's RVP node, the client unregisters with the old RVP and registers with the new RVP, lines 28–34.

The event handler from line 35 is triggered every time the upper overlay network layer sends a message over the network. Here, *dst* is the descriptor of the destination node. When the Usurp layer receives a message from the upper layer, it checks the NAT type of the destination node. If the destination node is a public node then the message is send directly to it, line 36–37. Hole punching is tried if the destination node is a private node. In order to start hole punching, first, we need to find out a RVP with whom the destination node is registered. Each destination node descriptor also contains a list of parent nodes responsible for the private node. A RVP selected from the list of the parents in the node descriptor, if all parents addresses are invalid then a lookup is sent to the SON for the destination node's id (key). The SON returns the RVP responsible for the destination node and hole punching is tried using this RVP. If hole punching succeeds then the message is sent to the port defined in the newly created mapping on the destination's NAT. Both nodes participating in the hole punching process know about the newly created mappings in the NATs if the hole punching process succeeds. The message is relayed, using the RVP node, if hole punching between the two nodes is not possible or hole punching fails, line 48-49. When Usurp layer receives a message from the lower network layer it simply delivers it to the upper overlay network layer, line 51–53. We also use a cache that contains open holes, line 39.

## 5 Experimental Evaluation

Our validation of Usurp involved layering a well-known overlay network, Cyclon [28], on top of Usurp and evaluating the performance of Cyclon/Usurp in the

**Algorithm 1** Usurp protocol.

```
internal data
    $id \leftarrow nd$                                              ▷ node's unique identifier
    $nat\_type \leftarrow nd$                                       ▷ NAT policies
    $rs \leftarrow nd$                        ▷ hole punching server a.k.a rendezvous server
end

upon event ⟨INIT | $node\_id$⟩ do
    stun_client.init()
    hp_client.init()
    SON.init()

    $id \leftarrow node\_id$
    $sServers \leftarrow$ bootstrap.getRandomPublicNodes()
    $sServer \leftarrow$ lowest_rtt($sServers$)
    $nat\_type \leftarrow$ run NAT-type Identification with $sServer$

    if nat_type = PUBLIC or nat_type = UPNP_ENABLED_NAT then
        if nat_type = UPNP_ENABLED_NAT then
            map_UPnP_ports()
        stun_server.init()
        hp_server.init()
        SON.join(id)
        hp_server.register(id, nat_type)        ▷ RVP for the public node is the node itself
    else
        $rs \leftarrow$ SON.lookup(id)
        rs.register(id, nat_type)                         ▷ establish out-of-band connection
        run RVP periodic check timer
end event

every $\triangle T$ do                                    ▷ private nodes check for RVP change
    $rs' \leftarrow$ SON.lookup(id)
    if rs' != rs then
        rs.unregister()
        $rs \leftarrow$ rs'
        rs.register(id, nat_type)
end

upon event ⟨MESSAGE | $dst$⟩ do                       ▷ message from the upper overlay layer
    if dst.nat_type = PUBLIC or dst.nat_type = UPNP_ENABLED_NAT then
        send ⟨MESSAGE ⟩ to $dst.address$                            ▷ direct communication
    else                                                     ▷ destination is a private node
        if hp_client.holeExists(id, dst.id) then                        ▷ pre-existing hole
            $dstHole \leftarrow$ hp_client.getDestinationHole(dst.id)
            send ⟨MESSAGE ⟩ to $dstHole$
        else                                                             ▷ do hole punching
            $rs' \leftarrow$ valid parent from destination node descriptor $dst$ OR SON.lookup(dst.id)
            $hp\_resp \leftarrow$ hp_client.doHolePunching(dst.id, rs')
            if hp_resp = SUCCESS then
                $dstHole \leftarrow$ hp_client.getDestinationHole(dst.id)
                send ⟨MESSAGE ⟩ to $dstHole$
            else if hp_resp = HP_NOT_POSSIBLE || hp_resp = HP_FAILED then
                rs'.relay(Message, dst.id)
end event

upon Receive ⟨MESSAGE⟩ do                           ▷ received from the lower network layer
    trigger ⟨DELIVER | MESSAGE⟩
end
```

presence of NATs compared to classical Cyclon in a NAT-free network. Cyclon is gossip-based peer sampling protocol that is widely used to build and maintain more complex overlay networks. Cyclon creates a random graph overlay network that has small diameter, low clustering coefficient and is highly resilient to churn. Each Cyclon node maintains a view that contains a bounded number of addresses of other nodes in the system. After a number of gossiping rounds, the view converges to a random subset of nodes in the system. Cyclon, and gossiping in general, assumes that any node can directly communicate with any other node in the system. In summary, our results show that (i) Cyclon/Usurp preserves the randomness properties of Cyclon, i.e., low clustering coefficient, short paths between nodes, small diameter, uniform random sampling and high resilience to churn and; (ii) public nodes incur an acceptable level of overhead and nodes participate evenly in gossiping, a node's amount of gossiping is not affected by the presence of NATs.

### 5.1 Experimental Setup

We implemented USurp as a message-level simulator using the Kompics platform [1]. Kompics provides a framework for building P2P protocols, and simulation support using a discrete event simulator. We developed a NAT emulator that emulates all the mapping, port allocation and filtering policies. All the messages sent by network layer pass through NAT emulator. In all experiments rule binding expiration time for every NAT was randomly chosen from the set {30, 60, 90, 120, 150, 180 sec}. When any message leaves or enters the NAT, it updates the corresponding rule expiration timestamp.

In our experiments, there is only one node behind each NAT, but in real life there may be multiple nodes behind a single NAT. Multiple nodes affect the success ratio for hole punching protocols by continuously allocating ports on the NAT that would be used by port prediction algorithms that are part of NAT Traversal protocols. We emulate the behaviour of multiple nodes behind each NAT, by attaching a component to the NAT emulator. Every second, it opens a new port on the NAT emulator. This is done by sending a dummy message outside the network. Destination IP and port information in the dummy message is set in such a way that the message always opens a new port on the NAT and never reuses an existing mapping.

The network size is set to 1024 and the latencies between pairs of nodes is modeled using the King data set [11]. Each experiment was run 30 times using different seeds and the results reported here are the averages of results obtained. Instead of initializing all nodes at once, we consider a growing network scenario where nodes gradually join the overlay. The arrival rate between two joins is set to a constant 500ms. We use centralized bootstrap sever that returns 20 random public nodes in the system. We are using Chord SON and in all experiments the successor stabilization timeout for Chord is set to 2 seconds and the finger stabilization timeout is set to 3 seconds. Due to space limitations, no replication is used in our experiments; every private node has only one RVP associated with it. The main parameters to set for Cyclon are the cycle period, which we set to 10 sec, the view size, set to 15, and the shuffle length, set to 5.

### 5.2 Correctness of the overlay network layer

To check the correctness of the overlay network layer we have tried to make the scenarios as realistic as possible. The ratio of open to private nodes is set to 1:4, similar to [7], and percentages of different types of NAT are taken from [22]. The statistics in [22] correspond to data collected by Peerialism, Sweden for a video streaming application. We set 5% of the NATs to support port-mapping using the UPnP Internet Gateway Device protocol.

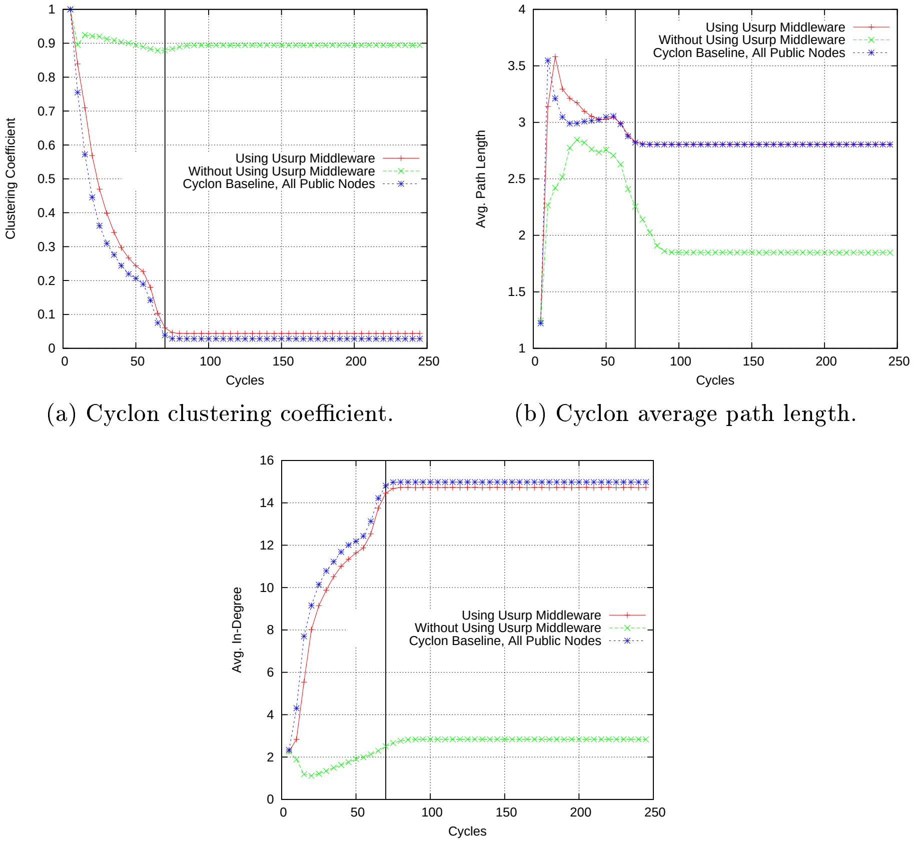
In all the graphs vertical lines represents the end of the growth of the overlay. The join process for all nodes completes around the $70^{th}$ cycle. As can be seen in figures 4a, 4b and 4c, Usurp produces results that are very close to classical Cyclon run using only public nodes - the clustering coefficient, average path length and average in-degree matrices converge very rapidly after all nodes have joined the overlay.

We can also see that if no NAT Traversal strategies are used, Cyclon performs badly in the presence of private nodes. There are few available links between the nodes, i.e., only the links among public nodes and the links from private to public nodes. This results in very low average in-degree and high clustering coefficient. The average path length is smaller because the presence of the NATs caused nodes to fail to join the overlay network. On average, only 75% nodes successfully joined the overlay.

### 5.3 Usurp overhead

We have used the same experiment setup for calculating bandwidth consumption of Usurp as a function of time. On average, the public nodes use five times more bandwidth than the private nodes. This is because the public nodes have dual responsibilities, i.e., they provide SON and RVP services to the remaining 80% of the nodes in the system. Bandwidth consumed by private nodes remains steady at 0.52 KB/s; and for a network of fixed size the bandwidth consumption for public nodes does not grow over time, as can be seen in figure 5a.

For calculating bandwidth consumption as a function of the percentage of private nodes, we use only one type of NAT. The NATs' *mapping policy* is set to Endpoint Independent, *filtering policy* is set to Endpoint Independent and *port allocation policy* is set to Port Preservation. Bandwidth consumed by public nodes grows as the percentage of private nodes in the system increases. Up to 80% of private nodes, for every 10% increase in the number of private nodes, there is on average a 7.72% increase in the bandwidth used by public nodes. However, this linear increase breaks down above 80% private nodes, and we observe a 30% increase in bandwidth consumption for public nodes from 80% to 90% private nodes.



(a) Cyclon clustering coefficient.

(b) Cyclon average path length.

(c) Cyclon average in-degree.

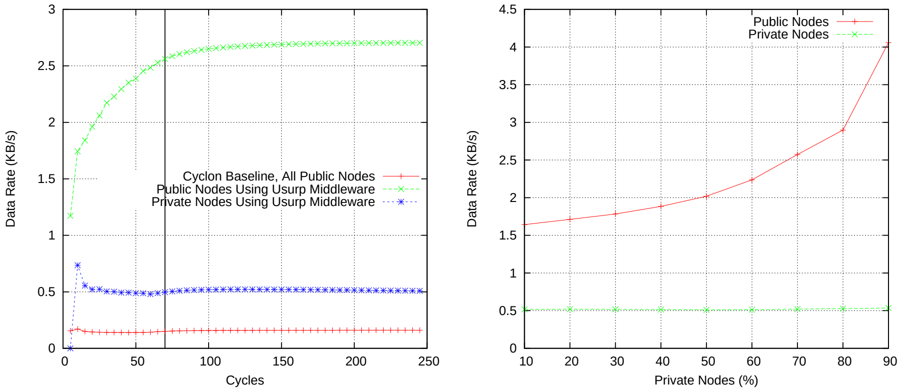
Fig. 4: Randomness properties of the Cyclon/Usurp overlay network.

### 5.4 Churn Resilience

We have tested our solution under high churn and failure rates. We define churn as certain fraction of the nodes joining and leaving the overlay in one gossip cycle; and failure is defined as the fraction of the nodes leaving the overlay in one gossip cycle.

For massive failure analysis, we again use only one type of the NAT as described above. We remove a fraction of nodes after every node has completed at-least 50 cycles. Public and private nodes are randomly removed from the system. Figure 6a, shows the size of the biggest cluster 50 cycles after the failure process has completed. We observe that our solution is highly resilient to massive failures and it can easily tolerate failure of 80% of the nodes. The overlay only starts to partition when the failure rate reaches 90%.

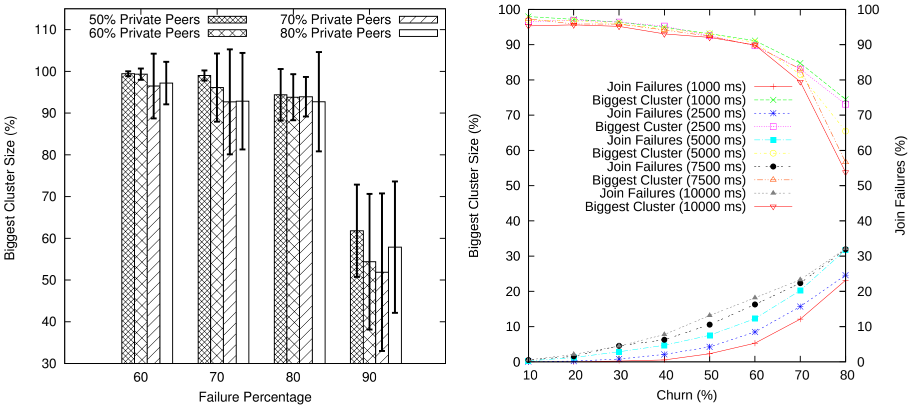
For churn analysis, we use the same scenario described in the first experiment. A fraction of nodes join and leave the system after every node in the system has



(a) Usurp overhead against time.

(b) Usurp overhead for an increasing percentage of private nodes.

Fig. 5: Usurp protocol overhead.



(a) Massive Failures.

(b) Churn.

Fig. 6: Behaviour of Usurp/Cyclon under churn and massive failures.

completed 50 cycles; and data is collected 50 cycles after the churn process has completed. For churn analysis, it is crucial to observe the effect of different finger and successor stabilization rates. In this experiment, finger and successor stabilization rates are set to same values. We observe that under high churn many nodes fail to join the overlay; this is because during the initialization process the bootstrap server returns dead public nodes or the SON ring has not stabilized. The bootstrap server evicts a public node if it does not receive a ping from the node. In all our experiments, the node eviction period was set to 20 seconds. We observe few join failures and high clustering with short finger and successor stabilization rates. Increasing the finger and successor stabilization rates directly effects the performance of the system, as can be seen in figure 6b.

## 6 Related Work

There are proprietary systems, such as Skype [9] and Hamachi, that support distributed NAT connectivity using public nodes, although details on their architecture are not public knowledge. Most existing P2P systems either use centralized servers to provide NAT connectivity [23] or do not support NAT connectivity at all [18]. The idea of connecting public nodes using a SON and having private nodes as clients originated with the Internet Indirection Infrastructure [26], although it did not address NAT traversal. The most similar system to Usurp is Maidsafe SON, a commercial implementation of Kademlia [15], where public nodes act as rendezvous servers. However, private nodes pick a rendezvous parent using bootstrap nodes from their own routing table dump on start-up, so there are no guarantees on whether a node can discover the rendezvous server responsible for a private node - false negatives are possible. Also, they do not separate NAT type identification from NAT traversal, so, similar to Interactive Connectivity Establishment (ICE) [21], as nodes do not know each others NAT type, a connection request results in a node trying to connect using several mechanisms in parallel: direct connection, connection reversal, and hole-punching.

Usurp's node descriptor is similar to that used in Teredo [27], where an address contains the private address and a public address (although, for Teredo the public address is an address on the NAT). Usurp's architecture has similarities to P2PSIP, whose goal is to implement SIP using Chord [4], although Usurp provides a more general connectivity layer. In [29], Wolinsky et al. showed how to bootstrap a P2P system using BruNet [3] and XMPP [24]. Similar to Usurp, they used a SON to implement relaying from public nodes in a SON to private nodes connected to public nodes in the SON.

There has also been work on peer sampling protocols that work in the presence of NATs, similar to Cyclon/Usurp from our evaluation[17,5,16]. Leitão et al. address the problem of balancing load among public and private nodes [17], while Actualized Robust Random Gossiping (ARRG) [5] uses a Fallback Cache containing public nodes to handle partitioning problems. Nylon is a peer sampling protocol that allows any node, whether open or natted, to act as a rendezvous server. However, they only consider the four classical types of NATs and do not take into account success rates of different hole punching protocols for different NAT combinations.

## 7 Conclusions

We have presented Usurp, a distributed NAT Traversal solution that supports node connectivity for overlay network protocols. The layered architecture of our solution allows the reuse of the Usurp layer with other protocols. We demonstrated that our solution does not require any changes to an existing overlay network protocol, Cyclon, and produces results comparable with Cyclon run in a network with only public nodes. We showed that Cyclon/Usurp is resilient to high failure and churn rates with up to 80% of nodes behind NATs, and it has reasonable overhead while preserving the randomness properties of the peer sampling service.

## References

1. Arad, C., Dowling, J., Haridi, S.: Developing, simulating, and deploying peer-to-peer systems using the kompics component model. In: COMSWARE '09: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE. pp. 1–9. ACM, New York, NY, USA (2009)
2. Audet, F., Jennings, C.: Network address translation (nat) behavioral requirements for unicast udp (2007)
3. Boykin, P.O., Bridgewater, J.S.A., Kong, J.S., Lozev, K.M., Rezaei, B.A., Roychowdhury, V.P.: A symphony conducted by brunet. CoRR abs/0709.4048 (2007)
4. Broadbent, T., Bryan, D.A.: P2psip, http://www.p2psip.org/index.php
5. Drost, N., Ogston, E., van Nieuwpoort, R.V., Bal, H.E.: Arrg: real-world gossiping. In: HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing. pp. 147–158. ACM, New York, NY, USA (2007)
6. Ford, B., Srisuresh, P., Kegel, D.: Peer-to-peer communication across network address translators. In: ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference. pp. 13–13. USENIX Association, Berkeley, CA, USA (2005)
7. Ganjam, A., Zhang, H.: Connectivity restrictions in overlay multicast. In: NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video. pp. 54–59. ACM, New York, NY, USA (2004)
8. Guha, S., Biswas, K., Ford, B., Sivakumar, S., Srisuresh, P.: RFC 5382: NAT Behavioral Requirements for TCP (October 2008)
9. Guha, S., Daswani, N., Jain, R.: An Experimental Study of the Skype Peer-to-Peer VoIP System. In: IPTPS'06: The 5th International Workshop on Peer-to-Peer Systems. Microsoft Research (2006), http://saikat.guha.cc/pub/iptps06-skype.pdf
10. Guha, S., Francis, P.: Characterization and measurement of tcp traversal through nats and firewalls. In: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement. pp. 18–18. IMC '05, USENIX Association, Berkeley, CA, USA (2005), http://portal.acm.org/citation.cfm?id=1251086.1251104
11. Gummadi, K.P., Saroiu, S., Gribble, S.D.: King: Estimating latency between arbitrary internet end hosts. In: SIGCOMM Internet Measurement Workshop (2002)
12. Hatonen, S., Nyrhinen, A., Eggert, L., Strowes, S., Sarolahti, P., Kojo, M.: An experimental study of home gateway characteristics. In: ACM SIGCOMM Internet Measurement Conference (IMC) (2010)
13. Huang, Y., Fu, T.Z.J., Chiu, D.M., Lui, J.C.S., Huang, C.: Challenges, design and analysis of a large-scale p2p-vod system. SIGCOMM Comput. Commun. Rev. 38(4), 375–388 (2008), http://dx.doi.org/10.1145/1402946.1403001
14. Huitema, C.: Teredo: Tunneling ipv6 over udp through network address translations (nats) (2006)
15. Hutchison, F.: Nat traversal in maidsafe dht. In: Accessed (Nov'2010): http://code.google.com/p/maidsafe-dht/wiki/NATTraversal (2010)
16. Kermarrec, A.M., Pace, A., Quema, V., Schiavoni, V.: Nat-resilient gossip peer sampling. In: ICDCS '09: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems. pp. 360–367. IEEE Computer Society, Washington, DC, USA (2009), http://dx.doi.org/10.1109/ICDCS.2009.44

17. Leit$\widetilde{a}$o, J., van Renesse, R., Rodrigues, L.: Balancing gossip exchanges in networks with firewalls (April 2010), international Workshop (IPTPS 2010). San Jose, CA
18. Lu, Y., Fallica, B., Kuipers, F.A., Kooij, R.E., Mieghem, P.V.: Assessing the quality of experience of sopcast. Int. J. Internet Protoc. Technol. 4(1), 11–23 (2009)
19. MacDonald, D., Lowekamp, B., Skype: Nat behavior discovery using session traversal utilities for nat (stun). IETF RFC 5780 (May, 2010)
20. Rosenberg, J., Weinberger, J., Huitema, C., Mahy, R.: Stun - simple traversal of user datagram protocol (udp) through network address translators (nats) (2003)
21. Rosenburg, J.: Interactive connectivity establishment (ice). In: IETF Internet Draft (October 2007), http://tools.ietf.org/html/draft-ietf-mmusic-ice-19.txt
22. Roverso, R., Ansary, S.E., Haridi, S.: Natcracker: Nat combinations matter. International Conference on Computer Communications and Networks 0, 1–7 (2009), http://dx.doi.org/10.1109/ICCCN.2009.5235278
23. Roverso, R., Naiem, A., Reda, M., El-Beltagy, M., El-Ansary, S., Franzen, N., Haridi, S.: On the feasibility of centrally-coordinated peer-to-peer live streaming. In: Consumer Communications and Networking Conference (2011)
24. Saint-Andre, P., Smith, K., Tronçon, R.: XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies. O'Reilly Media, Inc. (May 2009)
25. Schutt, T., Schintke, F., Reinefeld, A.: Structured overlay without consistent hashing: Empirical results. In: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid. pp. 8– . CCGRID '06, IEEE Computer Society, Washington, DC, USA (2006), http://portal.acm.org/citation.cfm?id=1134822.1134923
26. Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet indirection infrastructure. In: SIGCOMM. pp. 73–86 (2002)
27. Thaler, D.: Teredo extensions (2011)
28. Voulgaris, S., Gavidia, D., Steen, M.V.: Cyclon: Inexpensive membership management for unstructured p2p overlays. Journal of Network and Systems Management 13, 2005 (2005)
29. Wolinsky, D.I., St. Juste, P., Boykin, P.O., Figueiredo, R.J.O.: Addressing the p2p bootstrap problem for small overlay networks. In: Peer-to-Peer Computing. pp. 1–10. IEEE (2010), http://dx.doi.org/10.1109/P2P.2010.5569960
30. Y.Takeda: Symmetric nat traversal using stun (Jun 2010), http://tools.ietf.org/id/draft-takeda-symmetric-nat-traversal-00.txt
31. Zhu, Y., Hu, Y.: Efficient, proximity-aware load balancing for dht-based p2p systems. IEEE Trans. Parallel Distrib. Syst. 16, 349–361 (April 2005), http://dx.doi.org/10.1109/TPDS.2005.46