# Tor and NAT devices: increasing bridge & relay reachability or, enabling the use of NAT–PMP and UPnP by default

Jacob Appelbaum
jacob@torproject.org

Tor Tech Report 2012-08-001
August 22, 2012

**Abstract**

Tor relays and Tor bridges require at least a single reachable TCP port. This document discusses the current methods for enabling and enhancing Tor bridge and Tor relay reachability when behind a consumer grade NAT device. Such reachability improvements are extremely important for embedded devices that provide Tor relaying or Tor bridging services. We propose the use of NAT–PMP and/or UPnP protocol(s) to ensure that inbound connectivity is possible.

## 1 Introduction

Tor [1] is the second-generation onion router – it is both client and server software in a single program. When *Tor* is configured to be a relay or bridge, it requires a public IPv4 address and at least a single TCP port for its Onion Router Port (ORPort). Tor does not directly need to be assigned this IP – it is only required that inbound connectivity is possible on the advertised TCP port on the respective IP.

While other terms such as network address and port translation (NAPT), port address translation (PAT) or IP masquerading are used, we have chosen to use Network Address Translation (NAT) for brevity and simplicity.

The ORPort handles connections from either Tor clients or other Tor relays. When placed behind a NAT device, the *Tor* relay or bridge is generally able to make outgoing connections without restriction. In general, incoming connections are filtered and require that the NAT device forward traffic. In general consumer grade NAT devices have a single IPv4 address bound to its respective Wide Area Network (WAN) interface and that IP address is shared between many systems on the NAT device's Local Area Network (LAN) interface. Most consumer

grade NAT devices allow for services to be reachable behind the NAT device with manual configuration and many allow for automatic configuration with one of two major protocols, UPnP and NAT–PMP.

Tor only supports IPv6 for bridges at the moment and as NAT is not an issue with IPv6, we consider IPv6 to be largely out of scope for this document but we mention it when applicable.

While other methods of NAT traversal are possible, we consider them to be out of scope at this time due to their reliance on third parties (STUN[1], TURN[2], ICE[3], etc) or because they require specialized client software (pwnat [2], etc) to access services offered behind a NAT device; this topic is discussed in section 2. The former grouping of third–party–dependent solutions is almost certainly never a viable NAT piercing technique for anonymity reasons. The latter technique implemented by *pwnat* is much more attractive. It is generally friendly to privacy and does not rely on NAT router configuration.

This document explores methods of programmatically configuring NAT routers to improve Tor relay and bridge reachability. Automatic configuration of NAT routers is easy to use but slightly controversial. Many consumer grade NAT devices do support some form of port forwarding or NAT traversal. They are quite straightforward to configure in a programmatic manner. It is often the case that these devices support UPnP and/or NAT–PMP by default which allows for simple NAT traversal. Nearly all of these routers also support a web interface for configuration of port forwarding; we do not consider methods of automating configuration through the web interface.

## 2 Why we shouldn't rely on distant third parties

Full reachability is a serious problem on the modern internet. Offering a service requires bi–directional IPv4 connectivity. While it may seem attractive to attempt to use protocols STUN, ICE, or other NAT punching techniques that involve a third party, such techniques are generally bad for anonymity. We assume that readers of this paper are familiar with the work by Grothoff et al in Autonomous NAT Traversal [2], the basic ideas of NAT–PMP and UPnP and we cover the important aspects as they apply to Tor in section 3.

In the case of Tor relays, we expect that a third party relay, such as an ICE or STUN service becomes a kind of centralized area of the network with high value. When many relays route through a single third party, we may find that the anonymity properties provided by all of those nodes is reducible to the third party itself. The Tor network assumes that there is no global or network wide observer of all Tor nodes but adding a NAT-piercing third party would create one or a class of centralized systems to ease such monitoring. While asymmetric connection properties may create a more subtle security analysis, we think caution is prudent.

In the case of Tor bridges, in addition to the challenges faced by relays, we expect that any given third party used to broker bridge connections would simply become blocked, perhaps even before the bridge itself is blocked. Additionally, monitoring of the third party may reveal

[1] https://en.wikipedia.org/w/index.php?title=STUN&oldid=480053418

[2] https://en.wikipedia.org/w/index.php?title=Traversal_Using_Relays_around_NAT&oldid=480053422

[3] https://en.wikipedia.org/w/index.php?title=Interactive_Connectivity_Establishment&oldid=472491355

every bridge attempting to connect to the third party – this is certainly not a good property in terms of censorship resistance. Even if this service is run by a trusted party, we have no way to protect it from surveillance.

# 3 NAT traversal protocols

There are two protocols commonly used by consumer grade routers for NAT configuration. The most common protocol is the UPnP Forum managed standard, UPnP, and the second, but still extremely common protocol, is the Apple standard NAT–PMP.

These two protocols along with experimental ideas such as pwnat [2] perform NAT traversal with nearby third parties – generally NAT devices. STUN, ICE and other third party NAT traversal protocols use a well known third party on the wider internet, UPnP and NAT–PMP attempt to create a state of bi-directional connectivity through the nearest upstream NAT device. While this does not strictly remove the third party from the NAT traversal problem, we ensure that the third party is already in the network path.

Only pwnat [2] as described by Grothoff et al in their seminal Autonomous NAT traversal stands alone in its ability to produce results without a specialized protocol supported by a third party during NAT traversal. It however requires that both the client and server run specific software to complete connections. In the future it seems that a pwnat based pluggable transport for Tor bridges would be suitable for certain reachability cases. It is unlikely to work for the entire Tor network which requires that all relays communicate directly with all other relays.

## 3.1 UPnP

Configuration of upstream NAT devices is possible with the UPnP protocol in two different ways. The first method is available in the Vidalia Tor controller and the second is with the *tor–fw–helper* utility.

## 3.2 NAT–PMP

Configuration of upstream NAT devices with the NAT–PMP is not currently supported by the Vidalia Tor controller. There is currently no known plan to add NAT–PMP support to the Vidalia controller. It is possible to use the *tor–fw–helper* utility with the NAT–PMP protocol.

# 4 Tor controllers and NAT devices

When *tor* is launched or controlled by a controller such as Vidalia and it is running as a relay or bridge, the controller may attempt to configure any upstream NAT devices to forward required TCP ports. Vidalia does this with UPnP at the request of the user and not by default.

When *tor* is launched without such a controller, the current stable *tor* program itself has no knowledge about NAT piercing techniques. It requires a pre–configured upstream NAT device or it must be running on a system with a public IPv4 address.

# 5 Tor and NAT devices

The 0.2.3.x branch of *tor* includes two new options, *PortForwarding* and *PortForwardingHelper*, to assist in the configuration of upstream NAT devices. The default *PortForwardingHelper* is *tor–fw–helper* and *PortForwarding* is disabled by default.

The *tor–fw–helper* utility supports both UPnP and NAT–PMP if compiled with the requisite libraries. This utility is not currently built by default and each protocol must be manually specified and enabled at build time.

When *tor–fw–helper* is built with UPnP and NAT–PMP, and *PortForwarding* is enabled with *tor* configured as a relay or bridge, *tor* is generally able to pierce supported NAT devices without any additional software. This is especially important when embedded devices with Tor are considered as relay or bridge devices.

# 6 Security concerns

We presume that if Tor enables UPnP and NAT–PMP by default it will increase Tor router and Tor bridge availability without requiring manual reconfiguration of NAT devices. It is generally important to note that NAT devices are largely deployed to resolve IPv4 resource allocation issues. While NAT devices perform similar functions to network firewalls they are not by design a security boundary. If a network operator prefers to not use UPnP and NAT–PMP, it is suggested that they disable support for the protocols on their respective NAT devices. Furthermore, such a network operator must manually configure their respective routers to ensure that the required ports are globally reachable.

By Tor shipping with support for UPnP and NAT–PMP, we might encourage users to enable this feature on their NAT devices and thus make them more vulnerable to other applications also using the enabled UPnP or NAT–PMP services offered by their NAT device. We believe that this is a reasonable trade–off. Users have to positively affirm their desire to enable such a feature on their routers and such consent is a reasonable standard for such a trade–off. Users who are behind such a NAT device will have log messages indicating that UPnP or NAT–PMP was the NAT traversal method used by the Tor relay or Tor bridge.

Perhaps the greatest security concern is that Tor would depend on libnatpmp and libminiupnpc. This code could have vulnerabilities which may be exploitable and *tor–fw–helper* would raise the overall attack surface of a given Tor relay or Tor bridge.

# 7 Suggestions for improvement

Future releases of *tor* should build *tor–fw–helper* by default with full UPnP and NAT–PMP support enabled at compile time. Vidalia should be extended to understand the *tor PortForwarding* and *PortForwardingHelper* options rather than including UPnP or NAT–PMP library code directly. When *tor* is configured to be a relay or a bridge, it should ensure that *PortForwarding* is enabled without further user intervention. The exception is that IPv6 enabled bridges will not and should not ever need NAT traversal services; on such IPv6 enabled devices, we should not

automatically launch our *PortForwardingHelper* unless we also have a corresponding IPv4 IP address that must be globally reachable.

We suggest that libminiupnpc and libnatpmp must be audited for security related concerns before any full scale *tor–fw–helper* deployment. Any changes or security improvements should be sent to the upstream authors.

# 8 Conclusion

When *tor–fw–helper* is built and shipped in various binary releases and when *tor PortForwarding* is enabled, Tor bridge and relay reachability and availability will improve. Tor bridges and relays previously hidden behind many NAT devices will become available. Furthermore, NAT piercing on embedded devices will improve and graphical Tor controllers will be able to easily support these new features without significant increases in size or complexity.

# Acknowledgements

I would like to thank the University of Washington Security and Privacy Research Lab and other anonymous cypherpunks who contributed valuable feedback.

# References

[1] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004. https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf.

[2] Andreas Müller, Nathan Evans, Christian Grothoff, and Samy Kamkar. Autonomous NAT traversal. In *10th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 1–4. IEEE, August 2010.