Comparison of Tor Datagram Designs

Steven J. Murdoch

November 7, 2011

1 Background

A number of performance-related problems have been noted with the current Tor architecture, resulting in many users restricting their Tor usage to only tasks which are of high sensitivity, or not using Tor at all. The most comprehensive analysis of Tor's performance was performed by Reardon [Rea08]. In this work, the author identified the major cause of latency was delay in the output queue at Tor nodes, resulting from TCP flow control. This delay is considered higher than necessary due to:

- High-bandwidth streams unfairly trigger congestion avoidance on low-bandwidth streams;
- Packet dropping and re-ordering on one stream triggers unnecessary delay on other streams.

It has been proposed that to improve performance, the node-to-node communication should be by unreliable datagrams (UDP), rather than the current reliable in-order streams (TCP). This is hoped to improve Tor performance by:

- Allowing better end-to-end congestion management;
- Reducing queue lengths on nodes;
- Preventing cell-loss on one circuit delaying cells on other circuits.

Also, moving to an underlying datagram transport may make it easier to support transporting UDP in addition to TCP. More detailed analysis has been performed by Reardon and Goldberg [RG09].

2 Tor's current architecture

To understand proposals for modifications to Tor, it is helpful to understand the current architecture. This section will describe Tor's protocol stack, and while it does not use exactly the same terminology as the Tor Protocol Specification, the description chosen is designed to make it easier to compare with alternative proposals.

2.1 Tor circuit extension

Figure 1 shows the scenario when the circuit Initiator (normally termed the Onion Proxy) has a circuit which currently goes through one intermediate hop, and terminates on Previous. The Initiator then wishes to extend this circuit to terminate on Next. In this scenario the active layers are:

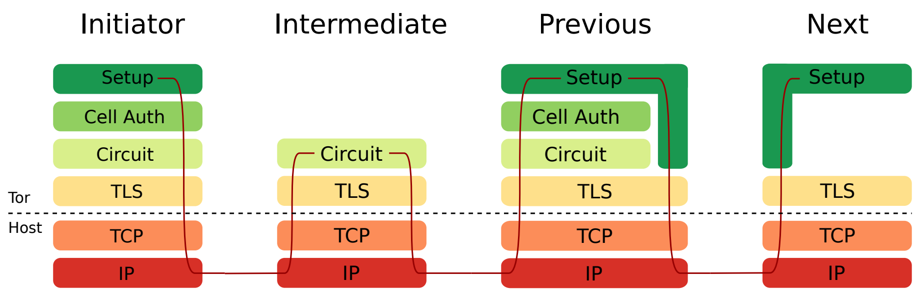

Figure 1: Protocol stack active when extending a circuit

IP The host operating system IP stack (and lower layers) is responsible for **routing IP packets** between the host and other Tor-nodes.

TCP The host operating system TCP stack is responsible for providing **hop-by-hop congestion control**, **in-order delivery**, and **reliability** for TLS data.

TLS The TLS stack built into Tor (OpenSSL) is responsible for providing **hop-by-hop authentication**, **integrity** and **confidentiality**.

Circuit The Tor circuit cryptography layer is responsible for providing **confidentiality**. It also **de-multiplexes** between different circuits being carried by TLS connections and performs **label-switching** routing.

While the Tor software on all nodes is identical, on intermediate nodes, only the layers up to and including Circuit are active. Cells received from the incoming TLS connection are decrypted, label-switched, and routed to the outgoing TLS connection according to its routing table.

Cell auth Tor's cell authentication provides **end-to-end integrity**

Setup Tor's setup protocol provides **connection setup and management** tightly coupled to **end-to-end authentication and key exchange**. It is also responsible for end-to-end congestion control.

The Setup layer can interact directly with the TLS layer when a control message is intended for the directly connected party, or via the Circuit and Cell Auth layers when the control message must pass through intermediaries.

2.1.1 The circuit extension process

In the case of circuit extension, the Initiator first wraps the control message with authentication (Cell Auth), two layers of encryption (Circuit) and passes the cell down to TLS.

Intermediate then removes one layer of encryption and passes the cell down to TLS.

When Previous receives the cell from TLS, the final layer of encryption is removed, the authentication tag is verified and the control message is processed. Previous will see that the control message indicates circuit extension (`RELAY_EXTEND`), and will send a `CREATE` control message to Next.

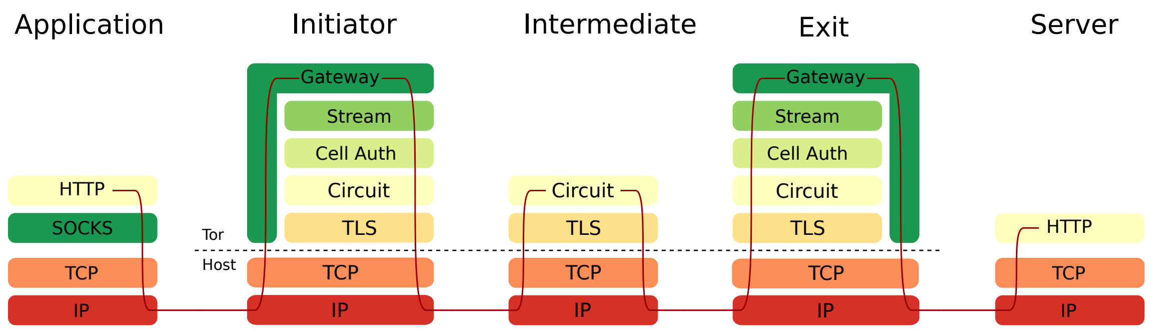

Figure 2: Protocol stack active when carrying data

2.2 Data transport

When an application wishes to send data over Tor, some additional layers are brought into play.

Application The client application wishes to send some stream data over Tor.

SOCKS The SOCKS layer encapsulates the stream data with a SOCKS header and carries out the SOCKS handshake.

TCP and IP In addition to transporting TLS data between Tor nodes, the host TCP/IP stack is also responsible for transporting the SOCKS stream to the Tor initiator (typically running on the same host as the application).

Gateway At the Initiator, Tor's Gateway layer receives SOCKS packets from the application SOCKS layer, extracts the payload data, and splits it into cells. These are then encapsulated in the same way as control messages, and passed to the Exit node via any intermediaries. This layer is also responsible for multiplexing multiple application streams over one circuit.

At the Exit, the Gateway layer receives payload data encapsulated in cells, and sends it out to the appropriate host via the host TCP/IP stack.

Cell Auth, Circuit, TLS These layers are unchanged as intermediate nodes cannot differentiate between control and data transport cells.

2.2.1 The data transport process

Based on the SOCKS handshake between SOCKS on the Application, the Initiator may create and extend circuits as necessary until it has established a circuit with the Exit.

The Gateway layer at the Initiator then then instructs the Gateway layer at the exit to make a plain TCP connection to the host requested by Initiator, and send the application stream data to it.

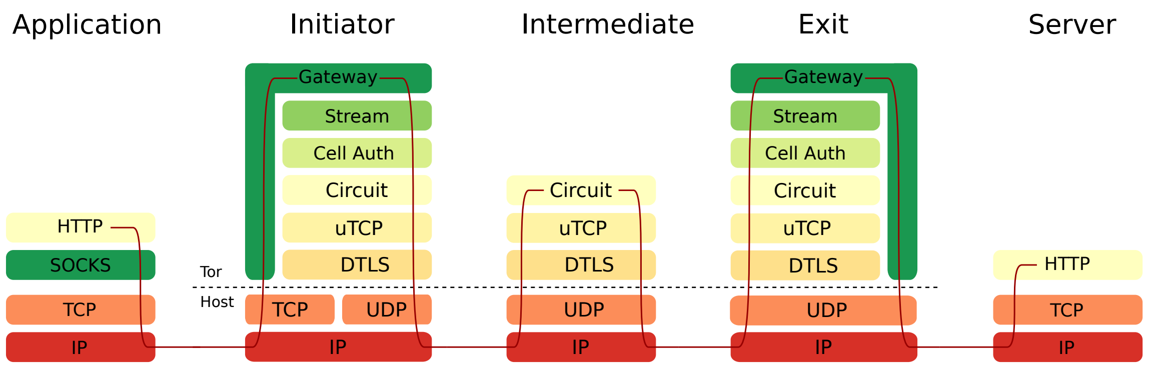

Figure 3: The main change in Reardon's proposal is to have 1:1 circuit per TCP connection mapping, rather than Tor's n:1, but TCP connections are still hop-by-hop in both schemes. To stop the individual connections being apparent to a network observer, TCP frames are wrapped in DTLS encryption. To allow this wrapping to be performed, a user-space TCP stack is employed, which has the added advantage of reducing usage of kernel-level sockets (a scarce resource on some platforms) and allowing greater customization of congestion control.

3 Alternative architecture proposals

This section discusses alternative architectures for Tor, involving the addition of a datagram hop-by-hop transport.

3.1 Reardon

Reardon and Goldberg [RG09] propose replacing the TLS layer with DTLS (a datagram variant of DTLS) and replacing TCP with UDP, as shown in Figure 3. DTLS still provides confidentiality and authenticity, however UDP does not provide reliability, in-order delivery, or congestion control.

The authors therefore propose adding TCP back in, but with each pair of nodes having a separate hop-by-hop user-space TCP connection for each circuit, rather than one kernel-space TCP connection for all circuits. For efficiency, the user-space TCP header is compressed by removing redundant fields.

Since the user-space TCP provides reliable in-order delivery of Tor cells, there needs to be no change to the cell encryption or authentication. A user-space TCP stack also allows more versatile congestion management; for example dropping cells before they acknowledged when the corresponding exit circuit is congested.

3.2 Viecco: UDP-OR

Viecco [Vie08], as shown in Figure 4 uses end-to-end TCP rather than the hop-by-hop approach of Reardon, and uses the host TCP stack rather than a user-space one. Also, control traffic is not sent within a TCP stream so is unreliable. If messages are lost the initiator must detect a timeout and repeat the action. TCP is end-to-end, thereby allowing middle nodes to drop and re-order packets, but leaving open the possibility of fingerprinting attacks.

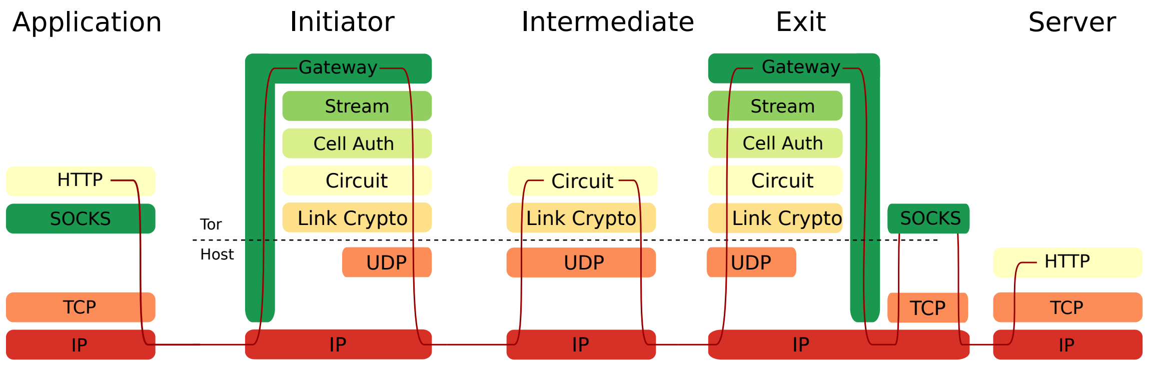

Figure 4: In the architecture proposed by Viecco, the initiator TCP stack is responsible for splitting SOCKS payload data into TCP frames. Tor would then transport these frames directly (after compressing and sanitizing some fields), for it to be reassembled by the host TCP stack on the exit node, have any SOCKS header removed, and be emitted again via the host TCP stack. Intermediate nodes use UDP, with reliability and congestion control managed by the initiator and exit. A custom link encryption/authentication scheme is used, but DTLS could equally take its place. The Circuit and Cell Auth cryptography schemes are updated to handle dropped or re-ordered cells.

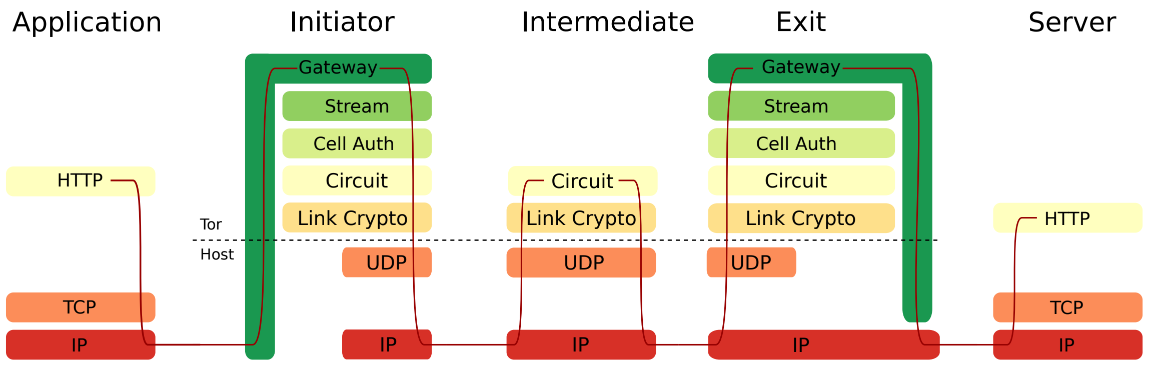

Figure 5: Like Viecco's proposal, TCP is end-to-end, however in Freedom this is initiator to server, rather than initiator to exit.

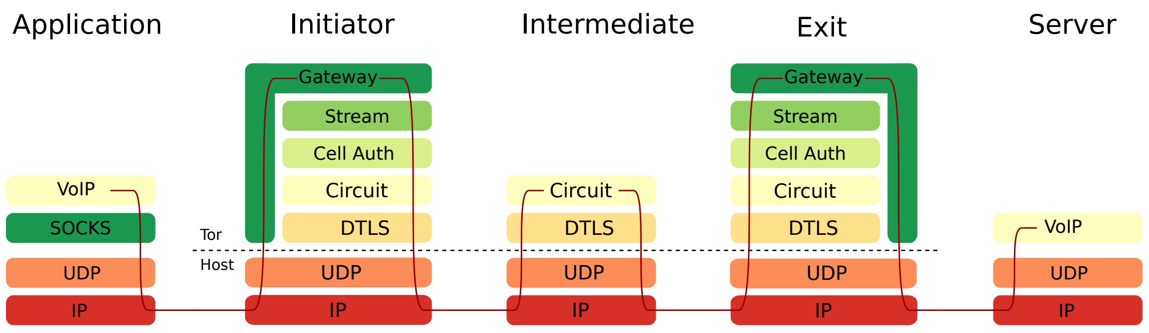

Figure 6: The Liberatore scheme is largely the same as Tor, except that data is received over UDP SOCKS rather than TCP SOCKS, transported over DTLS/UDP rather than TLS/TCP, and emitted as UDP rather than TCP. To handle the lack of in-order delivery, the Circuit cryptography is changed to have an explicit rather than implicit sequence number, and Cell Auth authenticates cells on the basis of the hash of a single cell rather than a running hash over all cells received in the circuit so far.

3.3 Freedom 2.0

Full details on Freedom [BSG00] is not available but it appears that, while like Viecco's proposal, TCP is end-to-end, in Freedom the ends are the Initiator and destination Server, rather than Initiator and Exit. Like Viecco's proposal, the initiator is responsible for reliability, through its host TCP stack. However, SOCKS is not used (Freedom captures application data in the host network stack), and control traffic is sent over TCP rather than UDP, and thus may assume reliable in-order delivery. Freedom also performs protocol filtering, before the data stream is split into TCP frames.

3.4 Cebolla and IPPriv

Other examples of the Freedom design are Cebolla [Bro02] and IPPriv [KBLC08]. Cebolla is notable for having a restricted topology, however in terms of transport protocol it appears to be similar to Freedom. IP packets are captured through a tun device on the initiator, encrypted, and then sent as UDP packets. At the exit nodes, IP packets are emitted over the tun device. As such, the characteristics of the initiator TCP stack are exposed to the destination.

Kiraly [KBLC08] proposes an anonymous communication system based on IPSec: IPPriv. It operates in a similar way to Tor, with telescoping circuits and link encryption, but uses IPSec for both. Unlike Tor circuit encryption, IPSec adds a header whether or not authentication is enabled. Therefore cells must be padded to hide how many layers of encryption is needed, and there is a limit on path length. Architecturally, IPPriv is similar to Freedom: IP packets are captured at the initiator and emitted at the exit nodes.

3.5 Liberatore: 100-tor-spec-udp

Liberatore [Lib06] proposes a design (shown in Figure 6) for the transport of UDP over Tor, but which does not offer in-order reliable delivery, so therefore cannot be used for TCP. It is intended to work in parallel to the existing TCP variant of Tor, and all control traffic is sent over the existing TLS/TCP connections between nodes; only UDP payload cells are sent over the DTLS/UDP links.

4 Transport protocols

The transport protocol is used, at a minimum for meeting the in-order reliable transport properties expected by applications to be provided by TCP.

4.1 Kernel-mode TCP

One option for a transport protocol is to use the kernel TCP stack as a transport protocol. When TCP sessions are end to end, this introduces a serious anonymity vulnerability as the host operating system could be fingerprinted (if TCP sessions are hop by hop, as in Reardon's proposal, this would not be an issue). There are also other challenges of using the kernel-mode TCP stack. Firstly, special operating system access would be needed to intercept packets from virtual network interfaces. Secondly, Tor would be unable to have low level control over the TCP congestion control algorithms.

4.2 User-mode TCP

Reardon uses the Daytona TCP stack, which has the difficulty of not being publicly available and under a license incompatible with Tor's There have been initial attempts to port the FreeBSD TCP stack to user-space, but these are not yet mature. In any case, Tor will be the primary user of any user-mode TCP stack for the foreseeable future, which could come with significant maintenance costs.

4.3 User-mode SCTP

When reliability is initiator-to-exit, or hop-by-hop, there is no need to use TCP. An alternative transport protocol is SCTP, which offers similar functionality to TCP, but with some extra features. There have been proposals to port the FreeBSD SCTP stack to user-space, and this would be a potential candidate for use in Tor, but the same arguments for user-mode TCP apply to SCTP.

4.4 µTP

µTP is a reliable in-order transport protocol using the LEDBAT (Low Extra Delay Background Transport) congestion avoidance algorithm, so as the achieve the following goals:

- Use all available bandwidth on a link
- Add little latency
- Yield to TCP flows using the same bottleneck link

Its major advantage over other user-space alternatives is that it is implemented in libutp[1], and this implementation has seen wide usage. Therefore if Tor were to adopt this library, we would not be entirely responsible for maintenance, and we have reasonable expectation that there would not be blocking bugs. However, µTP is designed to yield to TCP, whereas Tor will likely aim to be TCP-friendly but not necessarily yield to it. Also, µTP does not have an explicit method to preserve fairness between flows sharing the same link – one of the main goals of a transport protocol for Tor. Nevertheless, it may be possible to tweak the parameters of libutp to be more suitable for Tor.

[1] `https://github.com/bittorrent/libutp/`

4.5 CurveCP

CurveCP is a transport protocol offering congestion management and reliable in-order delivery. It also implements mandatory encryption and authentication. While both of these are required, they are not suitable for use within datagram-Tor directly. Firstly, hop-by-hop encryption and authentication may need to be performed without reliable in-order delivery. Secondly, the circuit encryption must be done without increasing length but CurveCP increases message lengths to accommodate the authentication tag.

Even so, it would be possible to use CurveCP as the transport protocol, and accept the inefficiency of encrypting data which is already encrypted. End-to-end authentication is desirable, so this feature would be of use although it would be inefficient to use the CurveCP handshake protocol when the two ends already share a key.

Alternatively, CurveCP could be refactored to separate out the congestion control and reliable in-order delivery and use this as an end-to-end transport protocol. Also, the encryption and authentication could be used for hop-to-hop links, but with reliable in-order delivery disabled.

A user-mode implementation for CurveCP is available[2], but does not include an explicit copyright statement. However related software from the author (Curve25519) was released into the public domain, so it is likely that CurveCP will also be.

5 Design decisions to be made

5.1 Reliability and congestion control end-points and granularity

A variety of options are available for the end-points for reliability and congestion control protocols. In Tor, TCP is used for both reliability and congestion control, on a hop-by-hop basis, with link level granularity. In Reardon's proposal, TCP is still hop-by-hop, but at circuit level granularity. With Viecco's proposal, TCP is initiator-to-exit, at stream level. Finally, Freedom, IPPriv, and Cebolla all have TCP initiator-to-server.

An advantage of initiator-to-exit/server reliability is that intermediate nodes may drop cells when load is high, and rely on congestion control to reduce the data rate. In contrast, with hop-by-hop reliability, once a cell has been acknowledged, it may not be dropped. However, circuit level cryptography is made easier and more efficient if it can assume reliability from the underlying transport.

A risk of initiator-to-exit/server reliability is that the characteristics of the reliability protocol are exposed to nodes other than those which the initiator directly connects to. This raises the possibility of fingerprinting, especially if the initiator's host networking stack is used.

Link level granularity should in principle have lower overhead, but has the disadvantage that lost cells from one circuit will cause unnecessary delay on other circuits. Circuit level granularity, as proposed by Reardon, removes this problem and thus decreases latency when there is packet loss. Stream level granularity requires initiator-to-exit/server reliability as the stream level is only exposed to the exit node and server, but in principle should reduce unnecessary delay even more.

Of course it is not necessary for reliability and congestion control to be linked, but existing transport protocols offer both. Tor currently uses TCP for link level congestion control, but uses a custom algorithm for circuit and stream level. Unless stream level granularity is used for the transport-protocol-provided congestion control, it is likely that some other congestion control algorithm will be needed to preserve fairness between different streams on a circuit.

[2] See `curvecp/` in `http://hyperelliptic.org/nacl/nacl-20110221.tar.bz2`

5.2 First-hop protocol

The primary reason to use a datagram transport is to reduce congestion within the core network. Therefore, it is not essential to use datagram transport for the connection from initiator to first-hop. Bridge users, who desire censorship-resistance may therefore wish to continuing using TLS over TCP, rather than datagrams. This will be relatively simple to accommodate with hop-to-hop reliability. However, with initiator to exit/server reliability, either there would need to be TCP within TCP (and the consequential performance impact), or the bridge node would need to be the TCP end point, rather than initiator.

It may however be desirable to use a datagram transport for the first-hop when used in conjunction with pluggable transports. This is because the pluggable transport would not need to provide reliability and so could run over UDP with little difficulty (e.g. disguising traffic as VoIP). Even with TCP-based pluggable transports, switching to datagram transport for the first hop could be useful. For example, while HTTP runs over TCP, it does not provide in-order reliable transport when multiple connections would be used (as is the case for any realistic implementation).

5.3 Migration path

Clearly it is essential that there be a smooth transition between the existing TCP transport and datagram transport. It is also necessary that the initial users of datagram transports are sufficiently numerous so as not to be deanonymized. It would also be desirable to make maximum usage of nodes which have been upgraded to support datagram transports.

Therefore, Tor would have to support both TCP and datagram transports until a sufficient proportion of the network has upgraded. If TCP is still used for first-hop connections, Tor nodes would have to support TCP for the foreseeable future.

With hop-by-hop reliability, it would be possible to use datagram transports for segments of a circuit where a pair of nodes support datagram transports. In contrast, with initiator-to-exit/server reliability, all hops for a circuit would need to support datagram transports for a circuit to use datagram transports. This means that more circuits would able to, at least partially, use datagram transports for hop-by-hop reliability.

So as to preserve the anonymity set of datagram transport users, clients should not use datagram transports until a sufficient number of other clients support them. This could be achieved by having a flag in the consensus which states whether datagram transports should be used. This flag would only be set once enough of the network supports datagram transports, on the assumption that clients upgrade roughly as frequently as nodes.

The need for collective action is less severe for hop-by-hop reliability because the datagram transport is only visible on a hop-by-hop basis. However, it should be assumed that whether a circuit is using hop-by-hop reliability will be visible to other hops on the circuit based on traffic characteristics. For initiator-to-exit/server reliability whether the initiator supports datagram transport will be clearly visible to all hops on the circuit.

5.4 Transport protocol

A transport protocol will need to be selected, such as µTP, CurveCP, TCP or SCTP (user-mode or kernel-mode).

5.5 Hop-to-hop encryption and authenticity

For all schemes considered, hop-by-hop encryption is required to hide which packet belongs to which circuit; authenticity is also highly desirable. Currently Tor uses TLS, but this requires a reliable link layer, which is likely to be eliminated to reduce unnecessary delay in conditions of packet loss. The "natural" choice would be to use DTLS, as adopted by Reardon, but a custom protocol, such as that adopted by Viecco, is another possibility. CurveCP includes a suitable link layer encryption, but the implementation tightly couples the encryption/authentication layer with the reliability and congestion control.

5.6 Socket usage

One current limiting factor for Tor nodes running on Windows is limitations on number of sockets. All the proposed datagram schemes with user-space reliability protocols reduce socket usage on intermediate nodes, which may be beneficial if the IOCP functionality in libevent does not completely solve this issue. However high socket usage is potentially still an issue on exit nodes, except for those proposals which use initiator-to-server reliability.

5.7 Circuit encryption and cell authentication

Tor's current circuit encryption scheme is AES CTR mode without explicit IVs. This approach depends on a reliable in-order transport so where reliability is above circuit encryption in the protocol stack (as it is with initiator-to-exit/server reliability), a different approach would be needed. The most obvious way of extending circuit encryption would be to include an IV in every cell, which would increase the protocol overhead but allow cells to be dropped or re-ordered without affecting the decryption of others. For hop-by-hop reliability, circuit encryption is above reliability and so no change is needed.

Tor's current cell authentication scheme is to include a running digest over all cells sent on this circuit. The digest is only 32 bits, on the basis that a circuit is destroyed if the digest doesn't match and therefore in the event that a cell is not detected as corrupted immediately, it is very likely that the following cell will be rejected and the circuit destroyed.

Calculating a running digest is possible with hop-by-hop reliability so there need be no change to cell authentication for this approach. However with Liberatore's and Viecco's approach, cell authentication is performed at a lower layer in the protocol stack than any reliability protocol and so a running digest is no longer suitable. Therefore a cell digest could be used, but to give similar security guarantees, the digest length would need to be extended.

Tor does not guarantee authenticity of cells, mainly because this would require a non-length-preserving cell encryption and thus limit path lengths and require padding to hide how many layers of encryption are in place. The path length limit is no longer an issue because there are other methods in place for Tor to limit path length, however padding may still be undesirable for efficiency reasons. Tor's lack of cell authenticity permits tagging attacks, but this is explicitly permitted by the Tor threat model. If this is considered a problem, an approach like Kiraly's, which includes hop-by-hop cell authentication, could be adopted.

5.8 Carrying UDP traffic

The primary motivation for Tor datagram transports is to improve the performance of TCP over Tor. However, it may also be desirable to allow UDP to be sent over Tor, for example VoIP traffic. All the schemes which use initiator-to-server reliability will naturally support UDP, as Tor would not need to even know the protocol type of packets. Liberatore's proposal is explicitly designed to support UDP, although UDP packets will not be indistinguishable from TCP packets because they are inside different link-layer encapsulations. Viecco's proposal could be easily extended to transport UDP, if the SOCKS server on the exit node were extended to support UDP. Reardon's proposal is more challenging to support UDP because reliability is ensured at the circuit layer and so any dropped cells will be re-transmitted, which will defeat the purpose of the application using UDP in the first place. Circuits could be marked as not needing reliability, but a different circuit encryption and cell authentication scheme would need to be used for these; also such circuits would be distinguishable from TCP circuits by intermediate nodes.

5.9 Carrying ACK messages

ACK messages needed for reliability and congestion control may need to be treated specially because they are much shorter than data packets. It would be possible to pad them to the same size as data packets, but this may be inefficient. With hop-by-hop reliability, ACK packets would be encrypted using DTLS which only adds a small amount of padding, so an external adversary would likely be able to tell which packets are ACK messages and thus learn something about the circuits being carried. With initiator-to-server/exit reliability the situation is more problematic as now information on traffic characteristics would be more visible to middle nodes.

Unreliable control messages Viecco proposes that control messages should not be carried in a reliable transport. A transport session is set up only between Initiator and Exit and only used for carrying data. This is best option in terms of minimizing state on intermediate Tor nodes, and is the closest to the standard Internet router model. It also reduces the overhead of setting up transport streams for carrying control messages. However it is complex from a protocol-design perspective as all participants must assume that any control message may be dropped at any point and arbitrarily re-ordered. In principle control messages may be corrupted too, but the hop-by-hop integrity protocol should prevent this from happening.

6 Recommendations and future study

This section proposes a set of provisional recommendations and raises questions which should be answered in further analysis.

6.1 Architecture

The most fundamental question to be resolved is the overall architecture: hop-by-hop reliability (e.g. Reardon), initiator-to-exit reliability (e.g. Viecco) or initiator-to-server reliability (e.g. Freedom). There are arguments for each approach, but in the absence of conclusive performance results, one approach is to guide the architecture by engineering and deployment difficulties. Initiator-to-server reliability would require low-level access on exit nodes, so as to generate raw packets, which could put pressure on the already scarce resource of exit bandwidth. Initiator-to-exit has the problem that the cryptographic protocols would need to be modified to handle the lack of reliable in-order delivery of cells. In comparison, this makes hop-by-hop reliability (e.g. Reardon) the most promising approach.

6.2 Transport protocol

No single candidate for transport protocol is the obvious choice. The anonymity and engineering difficulties with using the kernel-mode TCP stack suggest that this approach is not suitable, except perhaps for experimentation. User-mode TCP or SCTP stacks are possible, but are not yet available in a usable form. µTP and CurveCP are readily available, but µTP would be the easiest to integrate and so is a good choice for testing. Further study is needed as to whether investing engineering time in user-mode TCP or SCTP is a good choice, compared to analyzing and tuning µTP so that it has the properties required.

6.3 Other trade offs

A consequence of adoption hop-by-hop reliability is that Tor would remain able only to carry TCP traffic. This may be the prudent engineering choice, so as to avoid having to change many aspects of Tor at the same time. Nevertheless, if VoIP and similar protocols are strongly desirable, it may be worth revisiting this decision.

References

[Bro02] Zach Brown. Cebolla – pragmatic ip anonymity. Technical report, June 2002. http://www.cypherspace.org/cebolla/cebolla.pdf.

[BSG00] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[KBLC08] C. Kiraly, G. Bianchi, and R. Lo Cigno. Solving performance issues in anonymization overlays with a L3 approach. Technical Report DISI-08-041, University of Trento, September 2008. http://disi.unitn.it/locigno/preprints/TR-DISI-08-041.pdf.

[Lib06] Marc Liberatore. Tor unreliable datagram extension proposal. Proposal 100, The Tor Project, February 2006. https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/100-tor-spec-udp.txt.

[Rea08] Joel Reardon. Improving Tor using a TCP-over-DTLS tunnel. Master's thesis, University of Waterloo, September 2008. http://hdl.handle.net/10012/4011.

[RG09] Joel Reardon and Ian Goldberg. Improving Tor using a TCP-over-DTLS tunnel. In *Proceedings of the 18th USENIX Security Symposium*, August 2009.

[Vie08] Camilo Viecco. UDP-OR: A fair onion transport design. In *HotPETS*, 2008. http://www.petsymposium.org/2008/hotpets/udp-tor.pdf.