C-THRU™ Use Cases

$Rev: 7$

This document summarises use-cases collated from various customers, potential and otherwise. The primary purpose is to justify use-cases for all existing technical product features. This document also serves to keep track of common feature requests, and to act as a discussion point for further customer queries.

The use-cases here rationalise features of C-THRU; they do not introduce the product itself; an understanding of C-THRU is assumed, as provided by either the Functional Overview or Product Datasheet.

Use-cases state *functional requirements* from the perspective of a single customer, putting those requirements in context within that customer's system design.

These use-case summaries focus on how functional requirements relate to C-THRU's features, and addresses these requirements grouped by each feature. The intention here is to demonstrate features applying to different customer's situations, for different reasons. These different uses promote cleanly abstracted features through reuse and generalisation; the intention is that over time C-THRU's feature set remains small and focused, but still generally applicable to multiple situations.

Each use case relates to one or more *application topologies*; the topologies are abstract descriptions of typical situations in which C-THRU may be used, illustrating the essential points of the network topology of those applications.

Information proprietary to customers is not present in this document.

Contents

Bubblephone Limited
Sussex Innovation Centre
Science Park Square
University of Sussex
Brighton BN1 9SB
Tel +44 (0) 1273 704 535
http://www.bubblephone.com/

Application Topologies

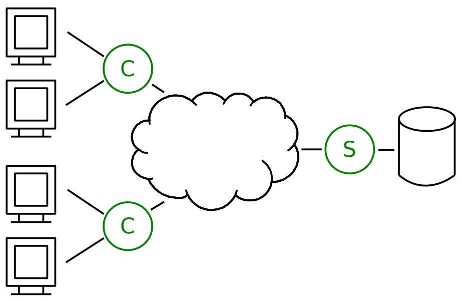
These topologies are illustrative of common situations only; more complex topologies may be created by combining these concepts.

Applications implement a subset of the features of these topologies. For example, a particular application may only require unidirectional traffic over its links, even though generally they may be bidirectional. Unless stated otherwise, all links shown may carry bidirectional traffic. Each link may carry streams, datagrams, or other socket types.

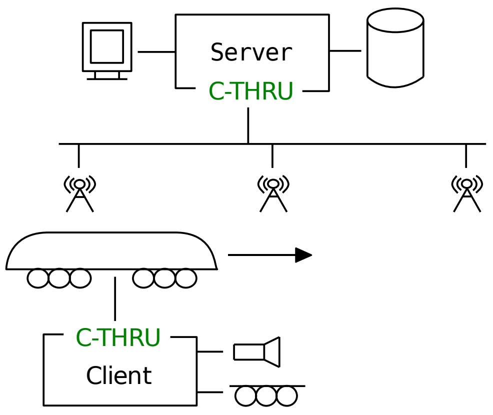
Client/server applications

Client/server topologies consist of one or more client nodes connecting to a single central server; their defining attribute is that the clients are responsible for instantiating connections, and hence must be aware of the server's address.

The simplified example shown illustrates multiple clients with some form of user interface (C), each connecting to a server (S). The clients and server are typically different programs.



The client-to-server connection does not infer the direction of traffic sent once connected. Typical case of traffic in either direction are clients connecting to a server to upload their logs, and clients connecting to retrieve information. The latter is suitable for centralising control of multiple clients.



More complex systems may be topologically equivalent in terms of their network arrangements. For example, [Frazer-Nash] produce monorail vehicles; each of which connects to a central server which orchestrates its movements.

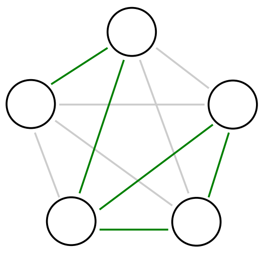
Connections are made over a series of wireless network connections; C-THRU is responsible for roaming between each cell of this radio network as the vehicle moves.

Topologically this arrangement is identical to the client/server situation described above; each vehicle acts as a client and the command and logging system as a server. For C-THRU this is identical to mobile clients roaming between access points. In [Frazer-Nash]'s case this also includes multiple redundant radio networks over different physical transports.

Mesh networks

Mesh networks are highly inter-connected nodes forming a subset of a completely-connected graph. Typically these are formed ad-hoc and have no particular hierarchy. [QinetiQ/1] [Skinkers]

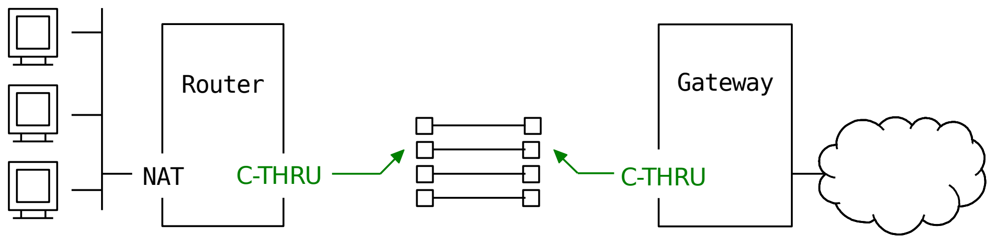
The defining attribute for mesh networks is that connections are formed between equal peers. Address discovery of peers may be dynamic (as is typical for most peer-to-peer systems such as [Skinkers]) or orchestrated by some other means. [QinetiQ/2]



In practice, the subset of connections in use may not include connections between all nodes on the network; some nodes may act as "supernodes", passing along traffic on behalf of node which may not be reached directly. This is particularly applicable for bridging between nodes where meshes are formed overlayed over different underlying networks.

Routing

Both [Cisco] and [QinetiQ/1] are applications which present a NAT interface to their users. This does not affect the feature set of C-THRU, but is illustrated here for context. The scenario illustrated shows an application which doesn't have a user interface; rather its interface is NAT.



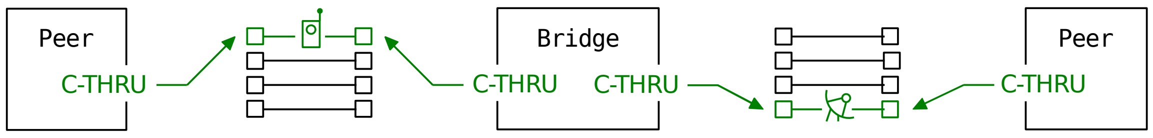
C-THRU provides a point-to-point link. Hence for users to connect to some external (non C-THRU) entity within the application network (for example, a machine on the Internet), a corresponding endpoint must exist for the C-THRU link. Therefore in order for a customer's application to provide a NAT interface to their users, a gateway must also be provided (which simply maps traffic as-is) for users to exit the C-THRU system and pass out onto their application network. This gateway need not be present if a user only need communicate with other users within the C-THRU system.

This particularly suits applications which cannot be modified to use C-THRU, either because they cannot be modified [DSTL] or because they are legacy systems. This topology permits a "black box" approach to interface existing systems to a C-THRU network. The router's user-facing interface is beyond the scope of C-THRU; it need not be NAT, but could be a special-purpose interface to suit a specific application. [MOD]

Bridging

For systems providing an application a variety of different physical transports available, each transport may be intermittently unavailable due to coverage constraints, network weather, or mobile applications simply moving out of range.

Two instances of an application are unable to communicate directly if they have no physical transport in common. This can be resolved by introducing a second application type to bridge the available transports:

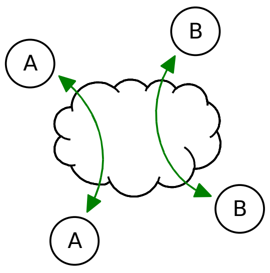


This situation is found typically on mesh style networks [QinetiQ/1] [Skinkers], where this bridge serves a "supernode"-like role, useful as a component for forming meshes over heterogeneous transports.

The bridge can also act as a gateway between two segments of the same transport, for example to map between overlapping address ranges by way of AF_CTHRU, for example to join two private IP ranges.

Overlay networks

Overlay networks tunnel traffic inside a lower-level network, rather than applications participating directly on that network. The topology of the overlaid network (typically a Mesh) need not reflect the underlying network's topology; the overlay abstracts the transport details.



Traffic may be tunnelled either over application layer protocols (commonly HTTP) or over lower level transports.

Overlaid networks can be used to create the illusion of simplicity over complex systems. One common use is for VPN-style networks to abstract the underlying addressing to flatten hierarchical address spaces, or to join disparate spaces.

Multiple unrelated instances of C-THRU may be overlaid over the same network without interfering with each other. [MOD]

Use Cases

Traffic and Transport

Application traffic is transferred over a variety of *connection methods*; each of these may employ a different underlying physical transport, or tunnel inside various application-layer protocols.

1. **Tunnelling:** The C-THRU architecture abstracts physical network connections. These need not be heterogeneous; each of these may tunnel over different protocols, as required per transport. [Frazer-Nash] use a variety of physical networks for redundancy.

   The use of these systems permits certain classes of data to be sent, primarily sensitive traffic which falls under regulations of its transfer. [QinetiQ/1].

2. **Hand-picked methods:** [Frazer-Nash] require a relatively small set of Methods (given that they control the physical links), but these are expected to make use of the available resources in a tightly-controlled manner. Therefore a set of pre-determined methods specific to their application may be appropriate.

   Likewise for regulated environments. [QinetiQ/2] [DSTL] [MOD]

3. **Scalability for lossy data:** Application data transferred includes lossy data such as VoIP traffic [Cisco] and video streams [Frazer-Nash]. It is desired that the quality of these scale according to the available network bandwidth (most likely to be influenced by the number of available WiFi channels, or contention with other traffic).

   For [Frazer-Nash] the video may be "behind time" by a certain amount, since it is non-critical. It is expected that the frame rate vary based on the application's knowledge of the vehicles's position, and therefore the amount of likely motion; for example, video streams of closed doors are considered "uninteresting" and are deprioritised until they open when the vehicle stops.

Connectivity

C-THRU maintains a pool of connected methods in order to facilitate changes; these interface internally to buffers responsible for ensuring no data is lost.

1. **Rapid connection:** Set-up time is critical for certain applications. In the case of [Skinkers], it is so important for the user experience that connections may be relayed through private servers (and their bandwidth paid for) while NAT traversal is being negotiated, since the length of time taken during negotiation is unacceptably long.

   Set-up time is usually relevant for an application which is instantiated by a user. Telephone calls are a typical example; the user does not expect to be kept waiting for their 'phone to ring. In some applications (such as Skype), a ringing sound is "faked" for user feedback, while the call is still connecting in the background.

2. **Connection maintenance:** Switching of methods is transparent to customer applications. This is especially relevant to [Frazer-Nash], where it would be wasteful to have their applications reconnect for each cell of their Radio Data Network during movement of a monorail from one cell to the next, but this also applies to most roaming applications. Maintaining transparency permits application traffic to continue without needing to restart its application protocol session.

   Switching methods (typically over different physical media) also provides for redundant connections for the purpose of failover. [DSTL]

3. **Rapid switching:** [Frazer-Nash] provide heterogeneous wireless uplinks at multiple points along the route of their monorail; the uplinks form the connections making up their Radio Data Network. To maintain a constant uplink, these cells overlap. Connection methods instantiated by the client library must switch-over quickly enough that this connection is not interrupted.

   [QinetiQ/2] require rapid switching to help address BGP's shortcomings for dynamic mobile nodes.

Application Interface

The interface presented by C-THRU to an application implements a socket API; this provides facilities for extensible options per protocol family. C-THRU makes extensive use of these for various purposes as described below.

1. **Peaceful coexistence**: Applications may require non-`PF_CTHRU` sockets in addition to one or more distinct `PF_CTHRU` sockets within the same application. This requirement is placed by the POSIX socket API, where different protocol families are expected to be unrelated.

2. **Dynamic socket options:** The socket API provides a protocol-specific mechanism to set and retrieve options. For `PF_CTHRU`, this is used to provide an application interface for configuring socket-related parameters (such as buffer sizes and timeouts). These may be adjusted dynamically, as the application sees fit. For per-socket options, each `PF_CTHRU` socket may have different values than any other `PF_CTHRU` sockets.

   [DSTL] and [QinetiQ/1] concern themselves with particularly low-throughput high-jitter transports, where atypical buffer sizes (and other fine tuning) will see benefit for overall performance.

   In some cases, applications have knowledge of where buffer sizes could temporarily be reduced to take advantage of momentary spikes in throughput and drops in latency, as per the network "weather". In the case of [QinetiQ/1] this knowledge comes to the application through means unknown to C-THRU (as opposed to by way of C-THRU's probe system).

   In extreme cases, it is desirable to block all traffic until the maximum underlying MTU size is reached, and only then to send data, in much the same manner as Linux's `TCP_CORK` option. This may be unset periodically where the application is aware of when it is best to "flush" data. [QinetiQ/1]

Addressing

C-THRU provides its own optional addressing mechanism (`AF_CTHRU`). This may optionally be used in place of the address families for underlying transports; the rationale for doing so is covered by use-cases here.

1. **Roaming:** Roaming maintains the illusion of a socket remaining connected when the underlying connection method is switched to another connection method. This permits applications to transition transparently between both homogeneous and heterogeneous transports. For heterogeneous roaming, `AF_CTHRU` addresses abstract the underlying address systems for simplicity. [QinetiQ/1]

2. **Dynamic renumbering:** For applications which allocate addresses dynamically, renumbering is straightforward in the usual manner. For router topology applications, the public address of the router is not relevant. Additionally, DNS is not required for the public NAT interface for [QinetiQ/1].

3. **Hierarchy reflects allocation:** Dynamically-allocated addresses are allocated according to the application's natural organisation.

   For [QinetiQ/1], this relates more to who needs to communicate with whom, rather than the corporate structure of their users.

4. **Overlay networks:** Networks such as the emergency services are hierarchical. For rapid response time, [QinetiQ/1] desire to efficiently contact whichever node is relevant within each hierarchy; therefore addresses must be bridged between those networks.

   [QinetiQ/2] require C-THRU for two parts of the same system; firstly for ad-hoc meshes of homogeneous transports, and secondly to form an overlay network heterogeneously linking these meshes together.

Security

Security implications affect the C-THRU system as a whole, not just the client library. This includes the structure of C-THRU's internal protocols, and their authentication against the cache.

1. **Clients maintain no private state:** If an instance of the client library is disassembled, all that should be gained is an understanding of the protocols C-THRU uses (which could be gathered from watching traffic on the wire anyway). This does not permit the attacker any more access than gained if they fabricated their own client endpoint; the clients are "dumb" in that they act as terminals for the protocol, and contain nothing sensitive with regards to C-THRU's internal authentication.

   [QinetiQ/1] express concerns for clients falling into the wrong hands, specifically to avoid it becoming a security threat.

2. **Topology must not be inferable:** [QinetiQ/1] express concerns about the topology of their mesh networks becoming visible if a client is captured, since that may be used to infer information about the command structure for military clients. Since `AF_CTHRU` addresses would be assigned dynamically (i.e. negotiated), this was not seen to be a problem.

   The topology of C-THRU itself is not relevant. [QinetiQ/1]

3. **Auditing:** [Frazer-Nash] desire a third-party security audit of multiple aspects of the system, to serve as quality assurance.

Infrastructure

As for deploying any application, upgrade paths and the customer environments must be taken into account. These use-cases enumerate specific situations encountered which relate directly to the installation and operation of C-THRU.

1. **Dynamic distribution:** Connection methods (and their dependencies) are optionally distributed to instances of the client library at run-time. This provides a route to update the set of available methods as and when new features become available.

   This distribution avoids the need to reinstall upgraded versions in most cases. Upgrades remain for (rare) API changes only. [Skinkers]

2. **Standards conformance:** Particularly for vehicular and defence applications, the product must conform to various industry standards for safety and reliability. These are reviewed case-by-case.

   The C-THRU client library need not conform to MISRA C or SIL 3 (though other parts of the application must). [Frazer-Nash]

3. **Decentralised cache:** The C-THRU cache may be distributed across several physical machines. [QinetiQ/1].

   [Frazer-Nash] explicitly do not wish to have a distributed cache, since it would be non-applicable for their application topology.

4. **Portability:** The C-THRU Client Library is made available for several platforms. [QinetiQ/1] require it for Linux, [Skinkers] and [Microsoft] for Windows, [Cisco] for IOS and [Frazer-Nash] for various embedded systems. Support for additional platforms may be added as required.

Forthcoming Features

These items are not currently implemented; they are either works in progress, or proposals which have no use-cases to justify their need. Some are strategies to improve efficiency of existing features; others provide additional features to the product.

Optimisation

1. **Data De-Duplication:** For transferring large amounts of repetitive data (such as backups of file servers), typical studies identify multiple duplicate copies of the same data which are transferred wastefully.

   By a process of pattern matching, data de-duplication aims to identify these duplicates "on the wire", and to identify and replace frequently occurring patterns with placeholder values pointing to previous instances of the same data. This is not compression; it forms a dictionary of patterns identified pointers, which are transmitted out-of-band in place of the data verbatim. The data is then reconstructed at the receiving end from these pointers to previous occurrences.

   This de-duplication and reconstruction process is transparent to the application; exactly the same data is reproduced. No understanding of the application's protocol is required.

   These use-cases are particularly relevant to applications which cannot conveniently identify their duplicated data themselves. Notably this applies to bridge topology applications, who's client nodes are not directly linked against C-THRU.

2. **Protocol Optimisation:** Unlike data de-duplication (which has no knowledge of the application-layer protocols involved), protocol optimisation implements an endpoint for the application layer protocol at each end of the socket. These endpoints "understand" the application protocol, and can transcribe its contents to a more efficient representation internally (common encodings are convenient for applications to implement, but wasteful on the wire).

   Furthermore, these endpoints may provide semantic optimisations such as rearranging requests, caching responses, or responding to pings locally, often without needing to involve the network at all.

3. **Dissociated directionality:** Some uplinks have significantly different upstreams than downstreams; in certain situations it is best to consider these as two separate links operating independently. [QinetiQ/1]

4. **Broadcasting and Multicasting:** This provides a mechanism for applications to communicate to a group of peers without maintaining a separately instantiated connection to each peer within that group. Traffic from one node to a group may be made more efficient if knowledge of this multicast nature is made available to C-THRU, and can be mapped directly to the underlying transports' abilities.

   This is of interest for "broadcast-like" systems such as [Skinkers], who distribute identical content to many nodes.

5. **Multiplexing:** Some devices have several low-bandwidth links; to perform optimally, C-THRU should use them in parallel, multiplexed across all `PF_CTHRU` sockets. [QinetiQ/1]

   Multiplexing at the C-THRU layer permits legacy applications which can only deal with a single socket to parallelise their traffic across several connections. For example to parallise database replication. [Ioko]

6. **Dynamic QoS:** Under specific situations, [Frazer-Nash] wish to prioritise one particular type of traffic over others. Each type of traffic uses a separate socket (possibly from different instances of the client library). This permits non-essential traffic to be reduced or dropped in emergency situations, where log and control data have precedence.

   This applies particularly when multiplexing several `PF_CTHRU` sockets over one underlying connection method, which cannot be distinguished from each other by QoS as implemented by the underlying OS.

7. **Frequency shaping:** Method selection may skew the profile of connection methods selected to match a desired protocol fingerprint. This is particularly suited to steganographic methods.

Interfaces

The socket API provided as an application interface is a thin layer mapping to the semantics of C-THRU's internal features. These features may be exposed via other interfaces as required, either in addition or as a an alternative to the existing socket API.

1. **Other Socket APIs:** Additional socket interfaces may be provided as required. Typically these would be native to each platform.

   These also include extensions such as the BIO abstraction employed by the OpenSSL API for socket creation. [Ioko]

2. **Triggers on result conditions:** C-THRU probes for various conditions as part of its usual operation. Some of these probes may measure bandwidth where required. This is an expensive undertaking which is desirable to minimise. Therefore, if an application needs to discover the the bandwidth available (for whatever reason), the client library may provide its results in order to save duplication of this discovery. [Skinkers] have specifically requested this feature.

3. **Explicit result instantiation**: Applications may explicitly supply "manifested" probe results, produced from their own knowledge.

   [QinetiQ/2] intend to make use of these to inject routing decisions. Likewise [DSTL] require a form of centralised control for the minutiae of method selection, this control data being broadcast over its own PF_CTHRU connection.

4. **Explicit result invalidation:** This is a mechanism for explicitly invalidating results (either manifest, or probe-generated) on a user-specified condition. This use-case provides failover for multiple paths through the network in much the spirit of BGP. [QinetiQ/2]

5. **Cache-side logs:** Statistics of traversal methods employed by instances of the client library are logged for analysis post-event. This information may be presented from the C-THRU cache; SNMP is expected to suffice for retrieval of this data.

Miscellaneous

1. **Isolated Addresses:** [QinetiQ/1] introduce a concept of "information need-lines": this is a mechanism for determining who is required to talk to whom. This access can be constrained at the addressing level with application-specific subnets expressed using an alternate `AF_CTHRU` family; one node may be present in multiple domains (i.e. have multiple addresses). Hence devices are multihoned if access is required to more than one information domain. This situation is an instance of the application bridge topology.

   [Frazer-Nash] are interested in separating their traffic types by isolating their address spaces using the same mechanism. This prevents other systems from accidentally interfering with their control connections.

2. **Sensitive data over regulated networks:** Sensitive data must only be transferred over an appropriate network. This calls for an interface to flag sensitive data, and of what type. [QinetiQ/1]

3. **Restricted use of regulated networks:** Networks such as TETRA have constrained use; they may only be used for appropriate traffic, hence the internal routing must know to exclude methods involving these from inappropriate traffic [QinetiQ/1].

4. **Explicit coverage constraints:** The ability to connect is based on the premise that prerequisite conditions defined by the user are met. These include latency constraints [Skinkers].

Out of scope features

C-THRU provides a socket API; sockets have a fixed scope. There are several socket-related operations which are commonly discussed, but fall out of the scope of sockets and therefore are not provided by C-THRU; these are listed below for reference.

The design intention is that C-THRU is used to coexist with other products which provide these requirements, which they are better able to do.

1. **Codecs:** Encoding application-specific data is beyond the scope of the socket library. However, C-THRU may provide information to help the application determine when it is appropriate to change codecs for scaling lossy compression to the resources available.
2. **Address resolution:** Existing resolution services may be used to map names to addresses for nodes, namely DNS.
3. **Encryption:** As with standard socket interfaces, any encryption employed is application-layer. This is especially relevant to [QinetiQ/1] and [DSTL], who use proprietary encryption schemes.

   C-THRU's own internal connections are not relevant to customers, provided that they are suitably secure against the relevant exploits.
4. **Supernoding:** This may be provided by Bridge-style application topologies. Under that topology, the application is responsible for passing the traffic between two isolated C-THRU sockets.
5. **Rate Limiting:** Throttling of traffic over particular transports may be provided by capping with QoS from the underlying operating system.

Bibliography

The use-cases and application topologies described by this document were taken from the following:

[Cisco] 12/12/2006. Cisco Systems Ltd.

[Microsoft] 17/07/2007. Microsoft Corporation.

[Skinkers] 02/08/2007. Skinkers Ltd, London.

[Frazer-Nash] 09/01/2008. Frazer-Nash Research Ltd.

[Ioko] 26/11/2008. Ioko365 Ltd.

[DSTL] 27/05/2009. Defence Science and Technology Laboratory.

[MOD] 27/05/2009. Ministry of Defence.

[QinetiQ/1] 05/07/2007, 17/10/2007, 18/09/2008, 04/12/2008. QinetiQ Group PLC.

[QinetiQ/2] 08/02/2008, QinetiQ Group PLC.

Minutes and attendances are available on request.