TCG Trusted Network Connect

TNC IF-TNCCS: TLV Binding

Specification Version 2.0
Revision 16
22 January 2010
Published

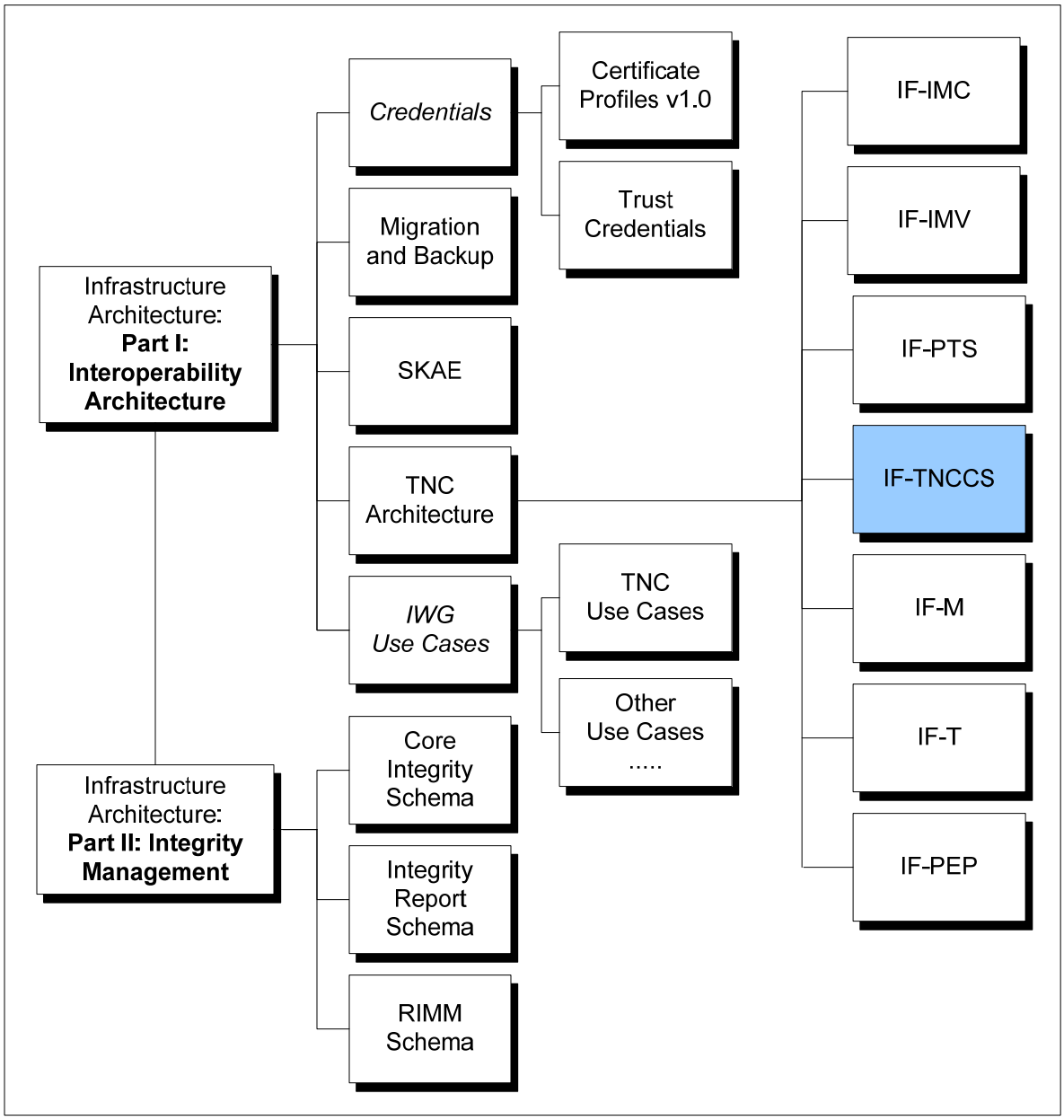
Contact:
admin@trustedcomputinggroup.org

TCG PUBLISHED

TCG

Disclaimers, Notices, and License Terms

TNC Document Roadmap



Acknowledgements

The TCG wishes to thank all those who contributed to this specification.

Aman Garg	**3Com**
Bipin Mistry	**3Com**
Scott Kelly	**Aruba Networks**
Amit Agarwal	**Avaya**
Mahalingam Mani	**Avaya**
Jeffery Dion	**The Boeing Company**
Steven Venema	**The Boeing Company**
Peter Wrobel	**CESG**
Mark Townsend	**Enterasys**
Michael McDaniels	**Extreme Networks**
Hidenobu Ito	**Fujitsu Limited**
Houcheng Lee	**Fujitsu Limited**
Sung Lee	**Fujitsu Limited**
Kazuaki Nimura	**Fujitsu Limited**
Boris Balacheff	**Hewlett-Packard**
Mauricio Sanchez	**Hewlett-Packard**
Ren Lanfang	**Huawei**
Dr. Jiwei Wei	**Huawei**
Han Yin	**Huawei**
Diana Arroyo	**IBM**
Emily Ratliff	**IBM**
Lee Terrell	**IBM**
Guha Prasad Venataraman	**IBM**
Chris Hessing	**Identity Engines**
Morteza Ansari	**Infoblox**
Stuart Bailey	**Infoblox**
Ivan Pulleyn	**Infoblox**
Ravi Sahita (Co-editor)	**Intel Corporation**
Ned Smith	**Intel Corporation**
Barbara Nelson	**iPass**
Chris Trytten	**iPass**
Josh Howlett	**JANET (UK)**

Roger Chickering	**Juniper Networks**
Charles Goldberg	**Juniper Networks**
Steve Hanna (Co-editor, TNC co-chair)	**Juniper Networks**
PJ Kirner	**Juniper Networks**
Lisa Lorenzin	**Juniper Networks**
Dean Sheffield	**Juniper Networks**
John Jerrim	**Lancope**
Tom Price	**Lumeta**
Matt Webster	**Lumeta**
Gene Chang	**Meetinghouse Data Communication**
Alex Romanyuk	**Meetinghouse Data Communication**
John Vollbrecht	**Meetinghouse Data Communication**
Bernard Aboba	**Microsoft Corporation**
Mudit Goel	**Microsoft Corporation**
Ryan Hurst (Co-Editor)	**Microsoft Corporation**
Tom Kelnar	**Microsoft Corporation**
Paul Mayfield	**Microsoft Corporation**
Ram Vadali	**Microsoft Corporation**
Jun Wang	**Microsoft Corporation**
Sandilya Garimella	**Motorola**
Joseph Tardo	**Nevis Networks**
Pasi Eronen	**Nokia Corporation**
Meenakshi Kaushik	**Nortel Networks**
Ron Pon	**Nortel Networks**
Mike McCauley	**Open Systems Consultants**
Bryan Kingsford	**Symantec Corporation**
Paul Sangster (TNC co-chair)	**Symantec Corporation**
Matthew Gast	**Trapeze Networks**
Curtis Simonson	**University of New Hampshire InterOperability Labs**
Brad Upson	**University of New Hampshire InterOperability Labs**
Lauren Giroux	**U.S. National Security Agency**
Chris Salter	**U.S. National Security Agency**
Jeff Six	**U.S. National Security Agency**
Rod Murchison	**Vernier Networks**

Michelle Sommerstad	**Vernier Networks**
Scott Cochrane	**Wave Systems**
Thomas Hardjono	**Wave Systems**
Greg Kazmierczak	**Wave Systems**

Table of Contents

1 Introduction

1.1 Scope and Audience

The Trusted Network Connect Work Group (TNC-WG) has defined an open solution architecture that enables network operators to enforce policies regarding the security state of endpoints in order to determine whether to grant access to a requested network infrastructure. This security assessment of each endpoint is performed using a set of asserted integrity measurements covering aspects of the operational environment of the endpoint. Integrity measurements are carried between the TNC Client and TNC Server on a protocol called IF-TNCCS (Trusted Network Connect Client-Server), as shown in figure 1 below.

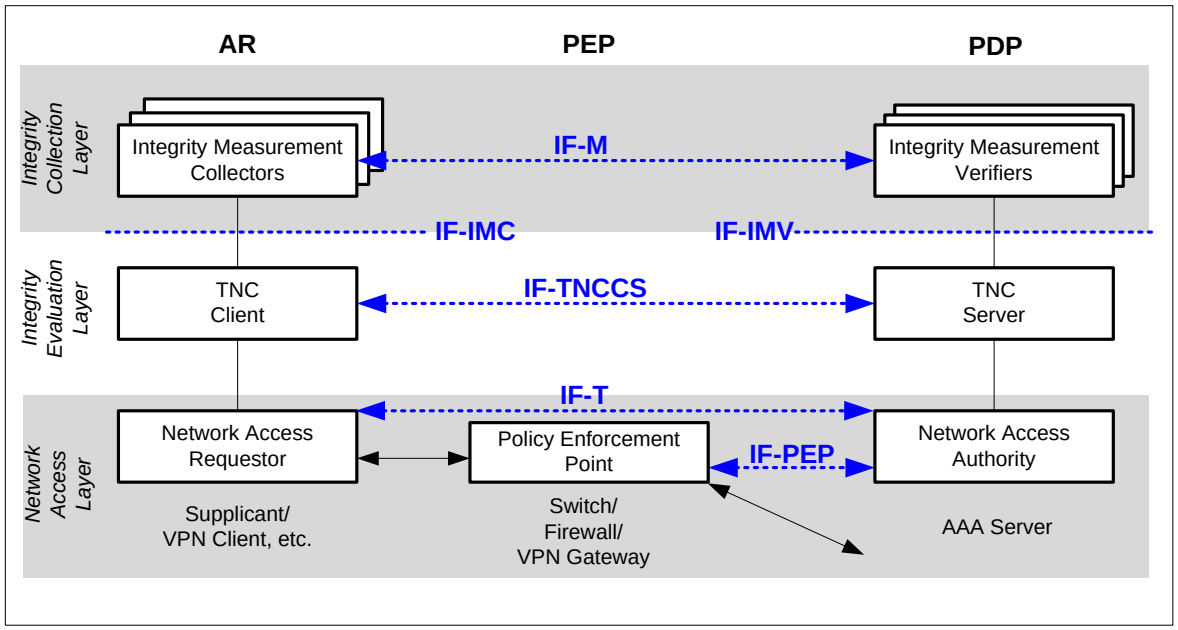


Figure 1 - TNC Architecture

This specification defines a new version of IF-TNCCS that is based on earlier versions of IF-TNCCS such as IF-TNCCS 1.1 [10] and IF-TNCCS-SOH 1.0 [11]. The goals of this new version of IF-TNCCS are:

- to support the same use cases and functionality as the earlier versions of IF-TNCCS
- to be compatible with IETF's PB-TNC protocol [5], thereby becoming the single agreed-upon standard client-server Network Access Control (NAC) protocol going forward

Before reading this document any further, the reader should review and understand the TNC Architecture specification [9]. If the reader is building a TNC Client that supports IF-IMC, the reader is encouraged to read the IF-IMC specification [6] prior to reading this document. If the reader is building a TNC Server that supports IF-IMV, the reader is encouraged to read the IF-IMV specification [7] prior to reading this document.

1.2 Interoperable with IETF PB-TNC

One of the goals of the Trusted Network Connect WG is to maximize interoperability using open standards. As part of fulfilling this goal, the TNC WG chose to take the TCG standard IF-TNCCS 2.0 protocol to the IETF for standardization. The initial version of IF-TNCCS 2.0 was placed in "public review" status until the IETF standardization process had completed allowing both the TCG and IETF to publish interoperable standards at approximately the same time.

The 2.0 version of this specification defines the IF-TNCCS protocol that is interoperable with PB-TNC [5]. It is the current intention of the TNC WG to keep the IF-TNCCS and PB-TNC protocols interoperable for the future.

1.3 IETF Terminology Mapping to TNC

In case readers of this specification are also looking at the IETF Network Endpoint Assessment (NEA)'s PB-TNC specification, this section provides some guidance on how the terminology aligns between the IETF and NEA specifications. For a full description of these terms, see IETF RFC 5209 [12].

PA-TNC - IETF NEA name for the application layer protocol that is interoperable with the TNC's IF-M [13]. "PA" is short for "Posture Attribute" protocol and "-TNC" highlights that the protocol is based upon work originally submitted by the TNC and is interoperable with IF-M.

PB-TNC - IETF NEA name for the protocol between the NEA client to NEA server that is interoperable with the TNC's IF-TNCCS 2.0. "PB" is short for "Posture Broker" protocol and "-TNC" highlights that the protocol is based upon work originally submitted by the TNC and is interoperable with IF-TNCCS 2.0.

Posture – IETF NEA term for "measurement information" used by TNC. The posture is returned from the NEA client (typically from its Posture Collectors) as part of an assessment. This is synonymous with the measurement information returned by the TNC client's IMCs.

Posture Collector – IETF NEA term synonymous with TNC's Integrity Measurement Collector (IMC)

Posture Validator – IETF NEA term synonymous with TNC's Integrity Measurement Validator (IMV)

1.4 Keywords

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1]. This specification does not distinguish blocks of informative comments and normative requirements. Therefore, for the sake of clarity, note that lower case instances of must, should, etc. do not indicate normative requirements.

2 Background

2.1 Role of IF-TNCCS

IF-TNCCS describes a standard way for the TNC Client and the TNC Server to exchange messages. More specifically, this interface defines a protocol and format for carrying:

(a) Messages from Integrity Measurement Collectors (IMCs) to Integrity Measurement Verifiers (IMVs) (such as integrity measurements)

(b) Messages from IMVs to IMCs (such as requests for additional integrity measurements, or remediation instructions)

(c) Messages from TNC Clients to TNC Servers (such as control messages)

(d) Messages from TNC Servers to TNC Clients (such as the IF-TNCCS Access Recommendation message)

Note that the contents of the messages being passed between the IMCs and IMVs ((a) and (b) above) are opaque to the IF-TNCCS layer. IF-TNCCS relies on the underlying transport protocol (IF-T) to provide a secure authenticated channel to protect the messages in transit between the TNC Client (TNCC) and the TNC Server (TNCS) and ensure they are delivered to the correct TNCC or TNCS.

2.2 Supported Use Cases

Use cases that IF-TNCCS supports are as follows. To better understand the use cases, consult the example in section 6.1.

TNCC Initiated Assessment Use Case

1. A TNCC initiates an integrity assessment (initial assessment or reassessment).
2. The IMCs send IF-M messages (typically integrity measurements). The TNCC receives these messages, collects them in a batch, and delivers them to the TNCS via the IF-TNCCS protocol. The TNCC may include standard and/or vendor-specific TNCC-TNCS messages in the batch also (that is, messages that are intended for consumption by the TNCC and the TNCS, not the IMCs or IMVs).
3. The TNCS receives the batch of messages. It processes any TNCC-TNCS messages itself and delivers the IF-M messages to the IMVs. After reviewing these messages, each IMV may provide an IMV Action Recommendation to the TNCS, indicating what action it recommends should be taken with respect to the endpoint's network access. Each IMV may also optionally respond by sending its own IF-M messages (remediation instructions, requests for more information, or other things) back to the IMCs. The TNCS delivers these IF-M messages (perhaps with some additional messages intended for the TNCC's consumption) to the TNCC through IF-TNCCS.
4. This exchange of messages may continue for multiple round trips within a single integrity assessment. Eventually, the IF-M message exchange will end naturally or the TNCS will terminate it. The TNCS will send a final batch of messages to the TNCC, including the TNCS Action Recommendation.

TNCS Initiated Assessment Use Case

This is the same as the TNCC Initiated Assessment Use Case except that the first step is that the TNCS initiates an integrity assessment and the IMVs get to send messages first. Note that this differs from the current TNC architecture, where IMCs always send messages first. This change will avoid an extra round trip in scenarios where the IMVs need to start the exchange and the TNCS is initiating the handshake.

Language Preference

A TNCC informs the TNCS of the endpoint users' language preferences through IF-TNCCS using standard TNCC-TNCS messages. The TNCS may expose the endpoint users' language preferences to the IMVs through IF-IMV. This allows the TNCS and/or IMVs to adapt strings intended to be human-readable so they conform to the users' language preference before sending them to the TNCC.

Note that this version of IF-TNCCS includes support for multi-user endpoints where the users have different preferred languages. A TNCC can specify more than one preferred language. Strings can be sent from the TNCS to the TNCC in several languages.

Reason Strings
One or more IMVs provide reason strings to a TNCS, giving the reason for their IMV Action Recommendations. The TNCS passes these reason strings to the TNCC through IF-TNCCS using standard TNCC-TNCS message(s).

Multi-protocol TNC Client or TNC Server
In order to ensure a smooth transition from IF-TNCCS 1.X and IF-TNCCS-SOH 1.X to IF-TNCCS 2.0, a TNC Client or TNC Server wishes to support more than one of these protocols simultaneously.

2.3 Non-supported Use Cases

- None

2.4 Requirements

Here are the requirements that the IF-TNCCS 2.0 protocol must meet in order to successfully play its role in the TNC architecture. In addition to these requirements, requirements specified by the IETF NEA Working Group [12] must be met since compliance with those requirements is a goal of this specification.

- Meets the needs of the TNC architecture

  The IF-TNCCS 2.0 protocol MUST support all the functions and use cases described in the TNC architecture as they apply to the relationship between the TNC Client and the TNC Server.

- Efficient

  The TNC architecture delays network access until the endpoint is determined not to pose a security threat to the network based on its asserted integrity information. To minimize user frustration, the IF-TNCCS 2.0 protocol MUST minimize delays and make IMC-IMV communications as rapid and efficient as possible. Efficiency is also important when you consider that some network endpoints are small and low-powered and that some networks have high latency, high cost, or low bandwidth. Also, some transport protocols are half-duplex with a limited fragment size and require a full round trip per fragment.

- Extensible

  IF-TNCCS will need to expand over time as new features are added to the TNC architecture. The IF-TNCCS 2.0 protocol MUST allow new features to be added easily, providing for a smooth transition and allowing newer and older architectural components to continue to work together. It MUST include support for vendor-specific extensions.

- Easy to use and implement

  The IF-TNCCS 2.0 protocol MUST be easy for TNC Client and TNC Server vendors to use and implement. It should allow them to enhance existing products to support the TNC architecture and integrate legacy code without requiring substantial changes. The protocol should also

make things easy for system administrators and end-users. Components of the TNC architecture should plug together automatically without requiring manual configuration.

- Transport Independent

  The IF-TNCCS 2.0 protocol MUST be capable of operating over any IF-T [8] transport protocol, so long as this transport protocol meets the assumptions listed in section 2.6. Half-duplex and full-duplex transports MUST both be supported.

- Scalable

  The IF-TNCCS 2.0 protocol MUST be highly scalable. For example, it MUST support having a large number (at least a hundred) IMCs and IMVs active in a single handshake and sending IF-M messages. The message type used for identifying and routing TNCC-TNCS and IF-M messages MUST support large numbers (hundreds) of standard defined message types and large numbers (hundreds) of vendor specific message types for each vendor with thousands of vendors. It MUST allow new message types to be defined over time.

- Able to coexist and function with IF-TNCCS 1.X and/or IF-TNCCS-SOH 1.X

  In order to ensure a smooth transition from IF-TNCCS 1.X and IF-TNCCS-SOH 1.X to IF-TNCCS 2.0, the IF-TNCCS 2.0 protocol MUST be designed so that a TNC Client or TNC Server can support any and all of these protocols at once (e.g. by making it easy to quickly detect the difference between the different protocols). Note that this does not impose a requirement on any TNCC or TNCS to support multiple protocols. It simply enables a TNCC or TNCS to do so.

- Vendor neutral

  The IF-TNCCS 2.0 protocol MUST be vendor neutral. However, it MUST include support for vendor-specific extensions.

- Fully interoperable

  The IF-TNCCS 2.0 protocol MUST be designed so that any TNC Client and TNC Server that comply with the specification will interoperate (assuming that they support a common IF-T transport protocol and have a compatible set of IMCs and IMVs and policies). Further, the IF-TNCCS 2.0 protocol MUST be interoperable with IETF's PB-TNC protocol.

- Internationalized

  The IF-TNCCS 2.0 protocol MUST provide a way for the TNCC to inform the TNCS of the endpoint user's language preference using standard TNCC-TNCS message(s). Any strings intended to be human-readable MUST be adaptable so that they conform to the user's language preference.

- Reason String Support

  The IF-TNCCS 2.0 protocol MUST provide a standard way for IMV and/or TNCS reason strings to be passed to the TNCC.

2.5 Non-Requirements

There are certain requirements that the IF-TNCCS 2.0 protocol explicitly is not required to meet. This list may not be exhaustive (complete).

- Fully compatible with existing IF-TNCCS 1.X or IF-TNCCS-SOH 1.X clients and servers

  The IF-TNCCS 2.0 protocol does not need to be fully compatible with existing IF-TNCCS 1.X or IF-TNCCS-SOH 1.X clients and servers. That is, there is no expectation that a TNC Client that only supports IF-TNCCS 1.X or IF-TNCCS-SOH 1.X will work with a TNC Server that only supports IF-TNCCS 2.0. While it would be great if this was possible, this is probably not consistent with the other requirements for IF-TNCCS 2.0. Therefore, it has been agreed that this is not a requirement. However, it is required (as noted above) that a TNC Client or TNC Server be able to support all of these protocols at once (or any ones that it may wish to support), in order to ensure a smooth transition from IF-TNCCS 1.X or IF-TNCCS-SOH 1.X to IF-TNCCS 2.0.

- Secure

  IF-TNCCS 2.0 does not provide security services itself. Instead, it relies on IF-T for secure transport of messages between the TNC Client and the TNC Server. This includes authentication, encryption, integrity protection, and replay protection.

2.6 Assumptions

Here are the assumptions that the IF-TNCCS 2.0 protocol makes about other components in the TNC architecture.

- Format of integrity measurements

  The format of the IMC-IMV messages that IF-TNCCS 2.0 conveys between the TNC Client and the TNC Server is opaque to IF-TNCCS 2.0. Each IMC-IMV message is simply represented as a binary piece of data within the IF-TNCCS 2.0 protocol.

- Transport

  IF-T is the underlying transport protocol for ALL IF-TNCCS 2.0 communication. It is assumed that IF-T will provide a reliable transport mechanism, ensuring the timely delivery of IF-TNCCS 2.0 messages in the same order in which they were sent. It is also assumed that IF-T will secure all IF-TNCCS 2.0 communications, providing adequate authentication, encryption, integrity protection, and replay protection.

- Fragmentation

  The IF-T protocol is expected to provide fragmentation when needed. However, it is understood that some IF-T protocols require a complete round trip for each fragment. Therefore, IF-TNCCS 2.0 will be designed to reduce fragmentation.

2.7 Message Diagram Conventions

This specification defines the syntax of the IF-TNCCS 2.0 messages using diagrams. Each diagram depicts the format and size of each field in bits. Implementations MUST send the bits in each diagram as they are shown, traversing the diagram from top to bottom and then from left to right within each line (which represents a 32-bit quantity). Multi-byte fields representing numeric values must be sent in network (big endian) byte order. Descriptions of bit fields (e.g. flags) values are described referring to the position of the bit within the field. These bit positions are numbered from the most significant bit through the least significant bit so a one byte field with only bit 0 set has the value 0x80.

3 IF-TNCCS Protocol

This section gives an overview of the IF-TNCCS 2.0 protocol.

3.1 Protocol Overview

The IF-TNCCS 2.0 protocol carries batches of IF-TNCCS messages between a TNC Client (TNCC) and a TNC Server (TNCS). It encapsulates IF-M messages and manages the TNCC-TNCS session. It runs over an IF-T transport protocol.

In order to work well over half-duplex IF-T protocols (such as those based on EAP [8]), IF-TNCCS 2.0 supports half-duplex operation. However, when a full-duplex IF-T protocol is employed (such as those based on TLS [15]), IF-TNCCS 2.0 can take advantage of the full-duplex feature to allow the TNC Server to initiate a handshake retry even when the TNC Server sent the last message. During a TNC handshake, the TNCC and TNCS take turns sending batches of messages to each other. While the half-duplex nature of this exchange could slow handshakes that require many round trips or bidirectional multimedia exchanges, this is not a problem in practice because integrity checks do not typically involve multimedia or a large number of round trips. The benefit of working over half-duplex transports outweighs any limitations imposed.

Each IF-TNCCS 2.0 batch consists of a header followed by a sequence of IF-TNCCS 2.0 messages. Each IF-TNCCS 2.0 message has a Type-Length-Value (TLV) format with a few flags. The TLV format allows a recipient to skip messages that it does not understand.

The PB-TNC specification defines certain standard IF-TNCCS 2.0 message types. It also permits vendors to define their own vendor-specific message types. One of the most important standard IF-TNCCS 2.0 message types is PB-PA. A message with this type contains a PA-TNC (IF-M) message. A TNCC or TNCS that receives such a message does not interpret the IF-M message within. Instead, it delivers the IF-M message to the appropriate set of Integrity Measurement Collectors (IMCs) or Integrity Measurement Verifiers (IMVs).

A TNCS will often need to communicate with several TNCCs at once. The reverse may also be true, as when an endpoint has multiple network interfaces connected to different networks. Each TNCC-TNCS connection is instantiated as a separate IF-TNCCS session. There may be several sessions between a single TNCC/TNCS pair but this is unusual.

3.2 IF-TNCCS 2.0 State Machine

Figure 2 illustrates the state machine for IF-TNCCS 2.0, which shows the set of states that an IF-TNCCS 2.0 session can have and the possible transitions among these states. The following paragraphs describe this state machine in more detail.

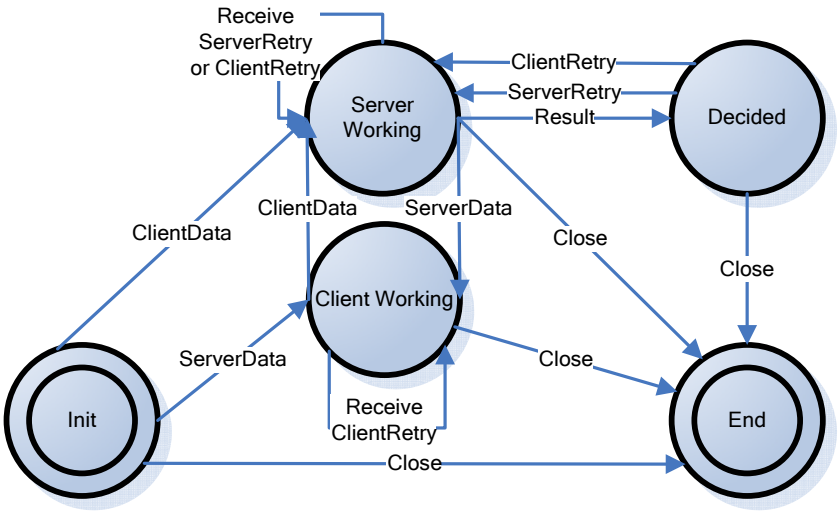


Figure 2: IF-TNCCS 2.0 state machine

Many state machine transitions are triggered by the transmission or reception of an IF-TNCCS 2.0 batch of a particular type. The type of an IF-TNCCS batch is indicated by the Batch Type field in the IF-TNCCS 2.0 header for that batch. For brevity, this document says “a FOO batch” instead of “an IF-TNCCS 2.0 batch whose Batch Type field contains FOO”. Other transitions are triggered by closing the underlying IF-T connection (Close) or by receiving an IF-TNCCS batch of a particular type (e.g. Receive ClientRetry).

An IF-TNCCS 2.0 session starts in the Init state when the underlying transport protocol (IF-T) establishes a connection between a TNCC and a TNCS. If the TNCC initiated the underlying transport session, it starts by sending a ClientData batch to the TNCS, thus causing a transition to the Server Working state. If the TNCS initiated the transport session, the TNCS starts by sending an IF-TNCCS batch of type ServerData to the TNCC, thus causing a transition to the Client Working state.

The TNCC and TNCS may now alternate sending ClientData and ServerData batches to each other. Only the TNCC can send a data batch when the session is in the Client Working state and only the TNCS can send a data batch when the session is in the Server Working state.

The most common way to end an exchange is for the TNCS to send a Result batch. This causes a transition into the Decided state. This is not a terminal state. The IF-T session remains open and another exchange can be initiated by having the TNCC send a ClientRetry batch. This can be useful when the TNCC (or more likely an IMC) discovers a suspicious condition on the endpoint, for example. If the underlying transport protocol (IF-T) supports full-duplex operation, the TNCS can also initiate another exchange from the Decided state by sending a ServerRetry batch. This can be useful when the policy changes on the server, for example.

Whether a Server Retry or Client Retry message is sent or both, the next state is the Server Working state. From this state, the TNC Server sends a Server Data batch and the new exchange begins. The state transitions marked “Receive ClientRetry” and “Receive ServerRetry or ClientRetry” indicate that it is permissible to receive such messages in the indicated states, generally when the TNC Client sent a ClientRetry message at roughly the same time as the TNC Server decided to send a ServerRetry. In that case, a ClientRetry message may be received while in the Server Working or Client Working state or a ServerRetry message may be received while in the Server Working state. These messages are redundant and therefore ignored, as indicated by the relevant transitions, which don't cause a state change. There is no need to have a legal state transition for receiving a ServerRetry message while the client is in a Client Working state because the client can only arrive at that state after a retry by receiving a ServerData message and a TNC is

not allowed to send a ServerRetry message right after a ServerData message. Receiving a ServerRetry message in that circumstance is not normal; it's an error.

The only terminal state is the End state. This state is reached if the underlying IF-T connection closes or if a Close batch is received (after which the IF-T connection is closed). A TNCS or TNCC SHOULD send a Close batch before closing the IF-T connection, if possible. However, this may not always be possible, especially if the IF-T connection is half duplex or if the connection close is caused by some external factor, such as pulling the network plug. In that case, the TNCC and TNCS will generally receive some form of notification from the Network Access Requestor (NAR) and Network Access Authority (NAA) that the IF-T connection has been closed. The Close batch or IF-T connection termination notification causes the transition to the End state.

Note that a TNCC and TNCS may not always have exactly the same state for a given IF-TNCCS 2.0 session. For example, say that a session is in the Client Working state and the TNCC transmits a ClientData batch. While this batch is in transit (transmitted by the TNCC but not yet received by the TNCS), the TNCC will think that the session is in Server Working state but the TNCS will think that the session is in Client Working state. However, this is a temporary condition and does not cause problems in practice. The only possible issue is that a TNCC or TNCS does not know whether the other party has received its message until it receives a response from the other party.

If a half-duplex transport is used, the TNCS cannot send a ServerRetry batch when the session is in the Decided state because the TNCS sent the most recent batch (the Result batch) and this would violate the half-duplex nature of the transport protocol. Instead, a server that wishes to initiate a new exchange in the Decided state when a half-duplex transport is in use should close the IF-T connection without sending a Close batch and start a new IF-TNCCS 2.0 session.

Any TNC Server and TNC Client that implements IF-TNCCS 2.0 MUST follow the state machine described in this section.

3.3 Layering on IF-T

IF-TNCCS 2.0 batches are carried over protocol bindings of the IF-T protocol, which provides the interaction between a Network Access Requestor (NAR) and a Network Access Authority (NAA). IF-TNCCS 2.0 counts on IF-T to provide a secure transport. In particular, IF-T MUST support mutual authentication of the NAA and the NAR, confidentiality and integrity protection for IF-TNCCS 2.0 batches, and protection against replay attacks. IF-TNCCS is unaware of the underlying transport protocols being used. IF-TNCCS operates directly on IF-T; no further layer of IF-TNCCS is expected. TCG has published several IF-T bindings such as IF-T: Binding to TLS [15] and IF-T for Tunneled EAP Methods [8]. However, other IF-T protocols may be used so long as they meet the requirements listed in this and other TNC specifications.

3.4 Example of IF-TNCCS Encapsulation

This section shows how a typical IF-TNCCS 2.0 batch might be carried inside of IF-T for Tunneled EAP Methods.

Within the top-level IF-T header, the IF-TNCCS 2.0 header is packaged next, followed by two PB-PA messages that contain IF-M messages meant for the IMCs or IMVs on the platform.

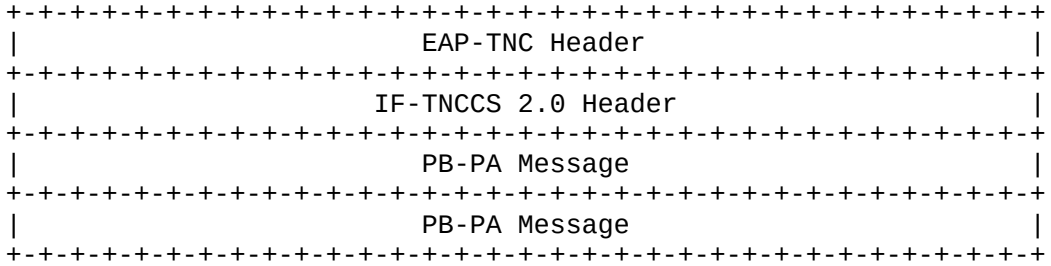


Figure 3: Example of EAP-TNC encapsulated IF-TNCCS message

3.5 Interoperability with older IF-TNCCS versions

A TNCC or TNCS may need to support IF-TNCCS 1.X, IF-TNCCS-SOH 1.X, and IF-TNCCS 2.0, all at the same time. To make this easier, the TLV Binding for IF-TNCCS 2.0 has been designed so that it can easily be distinguished from these other protocols. Simply looking at the first byte in the message allows one of these protocols to be distinguished from the others.

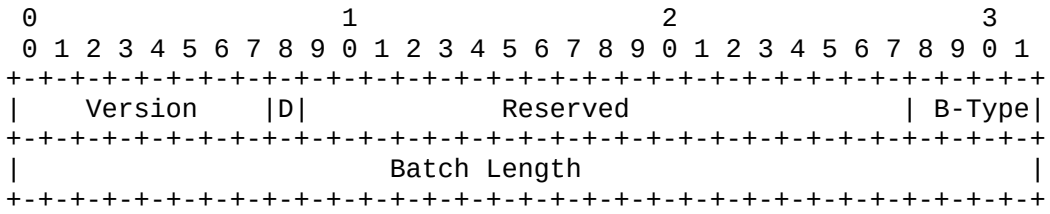
An IF-TNCCS 1.X message with UTF-8 encoding always begins with a byte with one of the following decimal values: 9, 10, 13, 32, or 60 (white space or the '<' character that starts the XML document). An IF-TNCCS-SOH 1.X message always begins with a byte with decimal value 0 (the first byte of the SoH Header structure). An IF-TNCCS 2.0 batch encoded with the TLV Binding for IF-TNCCS 2.0 (this version of this specification) always begins with a byte with decimal value 2. Therefore, these three message formats can easily be distinguished. Future versions of IF-TNCCS should be carefully designed to ensure that messages can be easily distinguished by looking at the first byte.

4 TLV Binding for IF-TNCCS 2.0

This section contains the TLV Binding for IF-TNCCS 2.0. Other bindings for the IF-TNCCS 2.0 protocol can be described later, if necessary. If a TNCC or TNCS receives a batch that violates the requirements of this specification, it MUST respond by sending a fatal Invalid Parameter error in a Close batch unless this document specifies otherwise.

4.1 IF-TNCCS 2.0 Header

Every IF-TNCCS 2.0 batch MUST start with the following header. An IF-TNCCS 2.0 batch MUST contain only one instance of this header followed by one or more IF-TNCCS 2.0 messages. The IF-TNCCS 2.0 messages are defined in subsequent sections of this specification.



Version (8 bits): This field MUST be set to 2 when the batch conforms to this specification (IF-TNCCS 2.0). Later versions of IF-TNCCS may define other values for this field. If a TNCC or TNCS receives a Version value that it does not support, it MUST respond with an IF-TNCCS Close batch that contains only a fatal Version Not Supported error code and whose Version header field has the value 2. Implementations responding to an IF-TNCCS 2.0 message containing a supported version MUST use the same Version number to minimize the risk of version incompatibility. IF-TNCCS 2.0 message initiators that support multiple IF-TNCCS 2.0 protocol versions SHOULD be able to alter which version of IF-TNCCS 2.0 message they send based on prior message exchanges with a particular peer TNCC or TNCS.

Directionality (D) (1 bit): When a TNCC is sending an IF-TNCCS 2.0 message, the Directionality bit MUST be set to 0. When a TNCS is sending an IF-TNCCS 2.0 message, the Directionality bit MUST be set to 1. This helps avoid any situation where two TNCCs or two TNCSs engage in a dialog. It also helps with debugging.

Reserved (19 bits): This field is reserved. For this version of this specification, it MUST be set to 0 on transmission and ignored on reception. Future versions of this specification may allow senders to set some of these bits and recipients to interpret them.

B-Type (Batch Type) (4 bits): This field is used to drive the state machine described in section 3.2. This field MUST have one of the values from the following table. If any other value is received, the recipient MUST ignore the contents of the batch and send a fatal Invalid Parameter error code in a Close batch. In addition, if the value received is not permitted for the current state, according to the state machine in section 3.2, the recipient MUST ignore the contents of the batch and send a fatal Unexpected Batch Type error code in a Close batch.

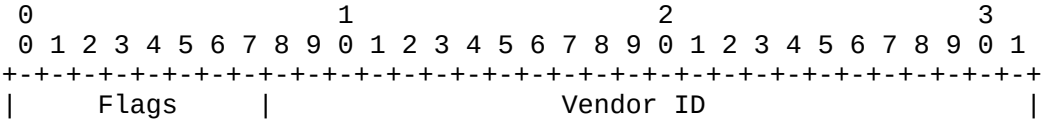
Batch Type Number	Batch Type Name	Definition
1	TNCCS20_BT_CLIENT_DATA	**The TNCC may send a batch with this Batch Type to convey messages to the TNCS. A TNCS MUST NOT send this Batch Type. A batch with this type may be empty (contain no messages), if the**

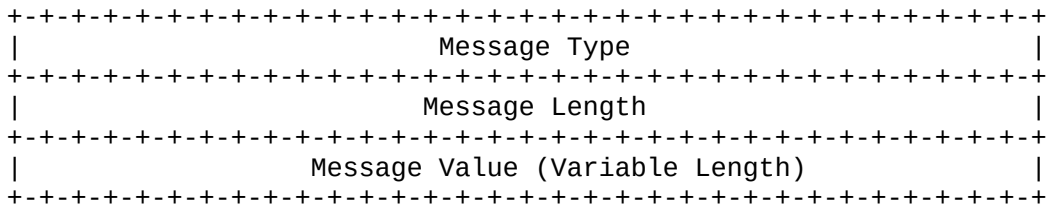
		TNCC has nothing else to send.
2	TNCCS20_BT_SERVER_DATA	**The TNCS may send a batch with this Batch Type to convey messages to the TNCC. A TNCC MUST NOT send this Batch Type. A batch with this type may be empty (contain no messages), if the TNCS has nothing else to send.**
3	TNCCS20_BT_RESULT	**The TNCS may send a batch with this Batch Type to indicate that it has completed its evaluation. The batch MUST include a PB-Assessment-Result message and MAY include a PB-Access-Recommendation message.**
4	TNCCS20_BT_CLIENT_RETRY	**The TNCC may send a batch with this Batch Type to indicate that it wishes to restart an exchange. A TNCS MUST NOT send this Batch Type. A batch with this type may be empty (contain no messages), if the TNCC has nothing else to send.**
5	TNCCS20_BT_SERVER_RETRY	**The TNCS may send a batch with this Batch Type to indicate that it wishes to restart the exchange. A TNCC MUST NOT send this Batch Type. A batch with this type may be empty (contain no messages), if the TNCS has nothing else to send.**
6	**TNCCS20_BT_CLOSE**	**The TNCC or TNCS may send a batch with this Batch Type to indicate that it is about to close the IF-T connection. A batch with this type may be empty (contain no messages) if there is nothing to send. However, if the termination is due to a fatal error then the batch MUST contain an error message.**

Batch Length (32 bits) – This length field contains the size of the full IF-TNCCS batch in octets. This length includes the IF-TNCCS 2.0 header and all the IF-TNCCS 2.0 messages in the batch. In other words, it includes the entire contents of the batch. This field MUST contain at least the value 8 for the fixed-length fields in this header. Any TNCC or TNCS that receives an IF-TNCCS 2.0 message with a Batch Length field whose value is less than 8 MUST send a fatal Invalid Parameter error code in a Close batch.

4.2 IF-TNCCS 2.0 Message

All IF-TNCCS 2.0 messages have the same overall structure, which is described in this section. Of course, the format and semantics of the Message Value field will vary, depending on the values of the Vendor ID and Message Type fields.





Flags (8 bits)

This field defines flags impacting the processing of this message.

Bit 0 of this flags field (the most significant bit) is known as the NOSKIP flag. If this flag is cleared (value 0), then the recipient (a TNCC or TNCS) may skip (ignore) this message if the message type is not understood or the recipient cannot or will not process the message as required in the definition of that message. If this flag is set (value 1), then recipients MUST NOT skip this attribute.

This flag does not mean that all recipients must support this message. Instead, any recipient that receives a message with this flag set to 1 but cannot or will not process it as required MUST NOT act on any part of the IF-TNCCS batch. Instead, the recipient MUST respond with a fatal Unsupported Mandatory Message error code in a Close batch. In order to avoid taking action on some messages in a batch only to later find an unsupported NOSKIP flagged message, recipients of an IF-TNCCS 2.0 batch might choose to scan all of the messages in the batch prior to acting upon any of the messages, checking to determine whether one of them is an unsupported message with the NOSKIP flag set.

The other bits in this Flags field are reserved. For this version of IF-TNCCS 2.0, they MUST be set to 0 on transmission and ignored on reception.

Vendor ID (24 bits)

This Vendor ID field identifies a vendor by using the SMI Private Enterprise Number (PEN). Any organization can receive its own unique PEN from IANA, the Internet Assigned Numbers Authority. This Vendor ID qualifies the Message Type field so that each vendor has 2^32-1 separate Message Types available for their use.

Message types standardized by the TCG must use the TCG SMI PEN (0x005597) in this field. Message types standardized by the IETF use a value of 0 for this field. Additionally, the Vendor ID `0xffffff` is reserved. A TNCC/TNCS MUST NOT send messages in which the Vendor ID has this reserved value (`0xffffff`). If a TNCC or TNCS receives a message in which the Vendor ID field has this reserved value (`0xffffff`), it MUST respond with a fatal Invalid Parameter error code in a Close batch.

TNC Clients and TNC Servers MUST NOT require support for particular vendor-specific extensions and MUST interoperate with other parties despite any differences in the set of vendor-specific extensions supported (although they MAY permit administrators to configure them to require support for specific extensions).

Message Type (32 bits)

This Message Type field identifies the type of the IF-TNCCS message contained in the Message Value field. The Message Type `0xffffffff` is reserved. A TNCC/TNCS MUST NOT send messages in which the Message Type field has this reserved value (`0xffffffff`). If a TNCC or TNCS receives a message in which the Message Type field has this reserved value (`0xffffffff`), it MUST respond with a fatal Invalid Parameter error code in a Close batch. Unless otherwise prohibited in the definition of a particular IF-TNCCS 2.0 Message

Type, a single IF-TNCCS 2.0 batch may contain multiple messages with the same message type and/or Vendor ID.

The TCG and any other organization with a PEN can define 2^32 – 1 unique IF-TNCCS Message Types, as long as the organization's PEN is placed in the Vendor ID field of the message. Since the IF-TNCCS Message Type is qualified by the Vendor ID, there is no risk of conflicts as long as each organization uses its own PEN for the Vendor ID and manages its own set of 2^32-1 message type values.

As part of the IETF's approval of the PB-TNC specification, the PB-TNC specification includes a set of IF-TNCCS Message Types (aka PB-TNC Message Types) that provide all the basic functions needed for IF-TNCCS 2.0. Please consult the PB-TNC specification for definitions of these message types. All of the requirements and recommendations (MUSTs and SHOULDs) in that specification that pertain to these message types apply to IF-TNCCS 2.0 as well. It is envisioned that future TNC specifications will assign IF-TNCCS Message Type values that use the TCG PEN.

Note that the IF-TNCCS 2.0 Message Type field is completely separate from the IF-M Subtype field. The same value (e.g. 0) may have different meanings as an IF-TNCCS 2.0 message type and as an IF-M Subtype.

Message Length (32 bits)

This field specifies the length of this IF-TNCCS 2.0 message in octets. It includes this header (the fields Flags, Vendor ID, Message Type, and Message Length). Therefore, this value MUST always be at least 12. Any TNCC or TNCS that receives a message with an IF-TNCCS 2.0 Message Length field whose value is less than 12 MUST send a fatal Invalid Parameter error code in a Close batch in response.

Message Value (variable length)

The syntax and semantics of this field varies, depending on the values in the Vendor ID and Message Type fields.

5 Security Considerations

As described in the Assumptions section of this document, IF-T is assumed to provide reliable and secure transport for the IF-TNCCS 2.0 protocol (including authentication, confidentiality, integrity protection, and replay protection). Still, it is useful to describe the possible threats to IF-TNCCS and the countermeasures that are employed. This section does that.

5.1 Threat Model

There are several possible threats to the IF-TNCCS 2.0 protocol.

Untrusted intermediaries on the network between the Access Requestor and the Policy Decision Point may attempt to observe data sent between the TNC Client and TNC Server via IF-TNCCS 2.0, modify this data in transit, reorder it, or replay it. They may also attempt to mount a denial of service attack against either party or truncate the exchange prematurely. If successful, these attacks may result in improper access decisions relating to the Access Requestor, failure to reassess access control decisions in light of changed circumstances, improper remediation instructions sent to the Access Requestor (which could lead to the compromise of the Access Requestor), unauthorized access to confidential information about the Access Requestor's health and/or identity, improper reason strings or other messages that might be displayed to the user, access to reusable credentials such as posture assertions, denial of service on the Access Requestor, and even complete denial of access to the network (if a denial of service attack against the Policy Decision Point is successful).

Trusted intermediaries between the Access Requestor and the Policy Decision Point include the Network Access Requestor and the Network Access Authority. These parties are considered trusted because they are responsible for properly implementing the security protections provided by IF-T. If they fail to do so properly, these security protections may be diminished or eliminated altogether. The possible attacks are the same as those listed in the previous paragraph. To give one fairly likely example, if a Network Access Requestor fails to properly authenticate and authorize the Network Access Authority (whether through implementation error or through user configuration to "trust anyone"), the improperly authorized Network Access Authority may mount any of the previously described attacks against the Access Requestor.

Compromise of any of the trusted parties (the TNC Client, the Network Access Requestor, the Network Access Authority, or the Policy Decision Point) may result in failures that are equivalent to those listed in the first paragraph. These failures may be even more dangerous since they will not be detectable by observing network traffic or by examining and comparing audit logs. Failure to properly secure communications between the TNC Client and the Network Access Requestor or between the TNC Server and the Network Access Authority is usually indistinguishable from compromise of those parties. Compromise of the operating system or other critical software, firmware, or hardware components on the Access Requestor or Policy Decision Point will typically result in an equivalent result. And an attacker's ability to gain privileged access to the Access Requestor or Policy Decision Point (even for a brief time, long enough to disable or misconfigure security settings) is generally equivalent as well. If the Access Requestor or Policy Decision Point are dependent on other services for their proper operation (including Integrity Measurement Collectors, Integrity Measurement Verifiers, directories, and patch management services), compromise of those services may result in compromise or failure of the dependent parties. Of course, compromise or failure of Policy Decision Point components is most serious since this would probably affect a large number of Access Requestors while the effects of Access Requestor compromise might well be limited to a single machine.

5.2 Countermeasures

The primary countermeasure against attacks by untrusted network intermediaries is the security provided by the IF-T protocol. Any candidate IF-T protocols should be carefully examined to ensure that all the threats described above are adequately addressed.

As noted above, compromise or erroneous operation of any of the trusted parties is a serious matter with substantial security implications. This includes the TNC Client, the TNC Server, the Network Access Requestor, and the Network Access Authority. These are all security-sensitive components so they should be built and managed in accordance with best practices for security devices. This is especially important for the Policy Decision Point since a compromise of this device would affect the security and availability of the entire network (similar to compromise of a AAA server). Communications between the trusted parties must also be secured. For example, if the TNC Server and the Network Access Authority are separate components, their communications must be secured.

Since the Access Requestor may be a mobile device with little physical security (such as a laptop computer or even a public telephone), it should generally be assumed that some proportion of Access Requestors will be compromised and therefore hostile. The Policy Decision Point should be designed to be robust against hostile Access Requestors. Once a compromised Access Requestor is detected, it can be treated in a manner equivalent to an untrusted party and should pose no greater threat than any other untrusted party.

Countermeasures against a compromised Policy Decision Point (or a component thereof such as a TNC Server or a Network Access Authority) include prevention of compromise, detection of compromise, and mitigation of the effects of compromise. For prevention, the Policy Decision Point should implemented using secure implementation techniques (e.g. secure coding and minimization) and managed using secure practices (e.g. strong authentication and separation of duty). For detection, the behavior of the Policy Decision Point should be monitored (e.g. via logging especially of remediation instructions, intrusion detection systems, and probes that impersonate a valid Access Requestor and record Policy Decision Point behavior) and any anomalies analyzed. For mitigation, Access Requestors should not blindly follow remediation instructions received from a trusted Policy Decision Point. At least for patches and other dangerous actions, they should validate these actions (e.g. via user confirmation) before proceeding. It should not be possible to configure an Access Requestor to trust all Policy Decision Points without proper authentication and authorization.

6 Use Case Walkthrough

This section provides informative (non-binding) walkthroughs of one typical IF-TNCCS use case, showing how IF-TNCCS 2.0 supports this use case. The walkthrough describes one particular example IF-TNCCS 2.0 exchange. It does not contain any normative text but is intended to serve solely as an example. A sequence diagram that illustrates this walkthrough is included at the end of this section.

6.1 TNCC Initiated Assessment Use Case

In this use case, the TNCC initiates an assessment. After exchanging a few IF-TNCCS batches, the TNCS decides to recommend full access.

6.1.1 IF-T Connection Setup

The first step is to have the NAR establish the IF-T connection. Of course, this is out of scope for IF-TNCCS but it is a necessary prerequisite since the IF-T connection must be established in order to carry IF-TNCCS traffic. IF-T connection establishment should include mutual authentication of the AR and PDP and establishment of a secure channel between them that will be used to carry the IF-TNCCS handshake. The TNCC and TNCS will both be in the Init state at the end of this step.

6.1.2 First TNCC Message

Since this is a TNCC initiated assessment, the TNCC sends the first IF-TNCCS batch. This batch contains an IF-TNCCS 2.0 header whose B-Type field has the value ClientData followed by two IF-TNCCS 2.0 messages. The first message in the batch is a PB-PA message containing an IF-M message revealing the endpoint's OS type (which the TNCC would have obtained from the relevant IMC). The second message is a PB-Language-Preference that specifies the user's language preferences (e.g. "Accept-Language: en-US"). After the TNCS receives this batch, both the TNCC and TNCS will be in the Server Working state.

6.1.3 First TNCS Message

The TNCS responds by sending a batch of type ServerData that requests further information about the endpoint. The only message in the batch is a PB-PA message containing an IF-M message inquiring about the posture of the anti-virus component (presumably sent by an anti-virus IMV). After the TNCC receives this batch, both the TNCC and TNCS will be in the Client Working state.

6.1.4 Second TNCC Message

The TNCC responds by sending a batch of type ClientData containing the requested information about the posture of the anti-virus component. The only message in the batch is a PB-PA message containing an IF-M message containing the requested information about the posture of the anti-virus component (presumably sent by the IMC responsible for the anti-virus component). After the TNCS receives this batch, both the TNCC and TNCS will be in the Server Working state.

6.1.5 TNCS Result

The TNCS responds by sending a batch of type Result containing the PB-Access-Recommendation determined by the TNCS. The only message in the batch is a PB-Access-Recommendation that indicates the TNCS' access recommendation. In this example, the TNCS has decided to recommend full network access so the Access Recommendation Code field will have the value 1 for Access Allowed. After the TNCC receives this batch, both the TNCC and TNCS will be in the Decided state.

6.1.6 IF-T Connection Teardown

Once the IF-TNCCS session is complete, the IF-T connection can be closed if it is now longer needed. This will result in a Close notification to the TNCC and TNCS. After that notification is received, both the TNCC and TNCS will be in the End state.

6.2 Sequence Diagram for TNCC Initiated Assessment

The following sequence diagram (Figure 4) illustrates the TNCC Initiated Assessment use case, as described in section 6.1.

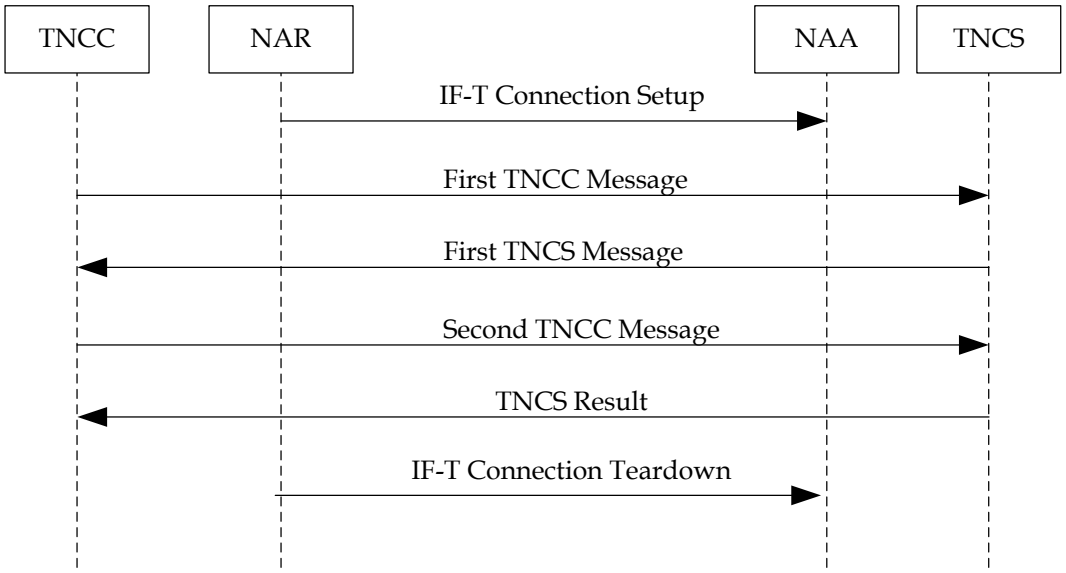


Figure 4 – TNCC Initiated Assessment Sequence Diagram

Appendix A: IF-TNCCS 2.0 C Structures

This section provides a C header file for the TLV binding of the IF-TNCCS 2.0 protocol.

```
/* tnc_if_tnccs_2_0.h
 *
 * Trusted Network Connect IF-TNCCS protocol version 2.0
 * July 8, 2009
 *
 * Copyright(c) 2007-2010, Trusted Computing Group, Inc. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * • Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 * • Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in
 *   the documentation and/or other materials provided with the
 *   distribution.
 * • Neither the name of the Trusted Computing Group nor the names of
 *   its contributors may be used to endorse or promote products
 *   derived from this software without specific prior written
 *   permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 * Contact the Trusted Computing Group at
 * admin@trustedcomputinggroup.org for information on specification
 * licensing through membership agreements.
 *
 * Any marks and brands contained herein are the property of their
 * respective owners.
 *
 */

#ifndef _TNCCS_2_0_
#define _TNCCS_2_0_

#define IANA_SMI_PEN_TCG 0x005597

/*****************************************************************
    TNCCS 2.0 TLV Header
```

```
*****************************************************************/

typedef struct _tag_tnccs20_header {
    unsigned int u8_version:8;
    unsigned int u1_directionality:1;
    unsigned int u19_reserved:19;
    unsigned int u4_batch_type:4;
    unsigned int u32_batch_length;
} t_s_tnccs20_header;

typedef enum _tag_tnccs20_batch_type {
    TNCCS20_BT_CLIENT_DATA = 1,
    TNCCS20_BT_SERVER_DATA,
    TNCCS20_BT_RESULT,
    TNCCS20_BT_CLIENT_RETRY,
    TNCCS20_BT_SERVER_RETRY,
    TNCCS20_BT_CLOSE
} t_e_tnccs20_batch_type;

typedef enum _tag_tnccs20_ctrl_dirn {
    TNCCS20_CTRL_TNCC_TO_TNCS =0,
    TNCCS20_CTRL_TNCS_TO_TNCC =1
} t_e_tnccs20_ctrl_dirn;


/*****************************************************************
    TNCCS 2.0 Message
*****************************************************************/

typedef struct _tag_tlv_msg {
      unsigned int u8_flags:8;
      unsigned int u24_vendor_id:24
      unsigned int u32_msg_type;
      unsigned int u32_msg_length;
      unsigned char p_u8_msg_value[1]; /* actually variable size */
} t_s_tnccs20_tlv_msg;

#endif //_TNCCS_2_0_
```

7 References

7.1 Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", Internet Engineering Task Force RFC 2119, March 1997.

[2] Phillips, A. and M. Davis, “Tags for the Identification of Languages”, Internet Engineering Task Force RFC 4646, September 2006.

[3] Alvestrand, H., “Content Language Headers”, Internet Engineering Task Force RFC 3282, May 2002.

[4] Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax”, Internet Engineering Task Force RFC 3986, January 2005.

[5] Sahita, R., Hanna, S., Hurst, R., and K. Narayanan, “PB-TNC: A Posture Broker (PB) Protocol Compatible with Trusted Network Connect (TNC)”, Internet Engineering Task Force RFC 5793, March 2010.

7.2 Informative References

[6] Trusted Computing Group, *TNC IF-IMC*, Specification Version 1.2, February 2007.

[7] Trusted Computing Group, *TNC IF-IMV*, Specification Version 1.2, February 2007.

[8] Trusted Computing Group, *TNC IF-T for Tunneled EAP Methods*, Specification Version 1.1, May 2007.

[9] Trusted Computing Group, *TNC Architecture for Interoperability*, Specification Version 1.4, May 2009.

[10] Trusted Computing Group, *TNC IF-TNCCS*, Specification Version 1.2, May 2009

[11] Trusted Computing Group, *TNC IF-TNCCS-SOH*, Specification Version 1.0, May 2007

[12] Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, ”Network Endpoint Assessment (NEA): Overview and Requirements”, RFC 5209, June 2008.

[13] Trusted Computing Group, *TNC IF-M*, Specification Version 1.0, March 2010

[14] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, “Extensible Authentication Protocol (EAP)“, RFC 3748, June 2004.

[15] Trusted Computing Group, *TNC IF-T: Binding to TLS*, Specification Version 1.0, May 2009.