Internet Protocols
Laboratory

Autonomous networks need standards

Colin Perkins

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow | This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License.
To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Autonomous networks need standards

Standard protocols

Standard management APIs

Common infrastructure

…as a substrate for autonomous reasoning

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

standards are slow

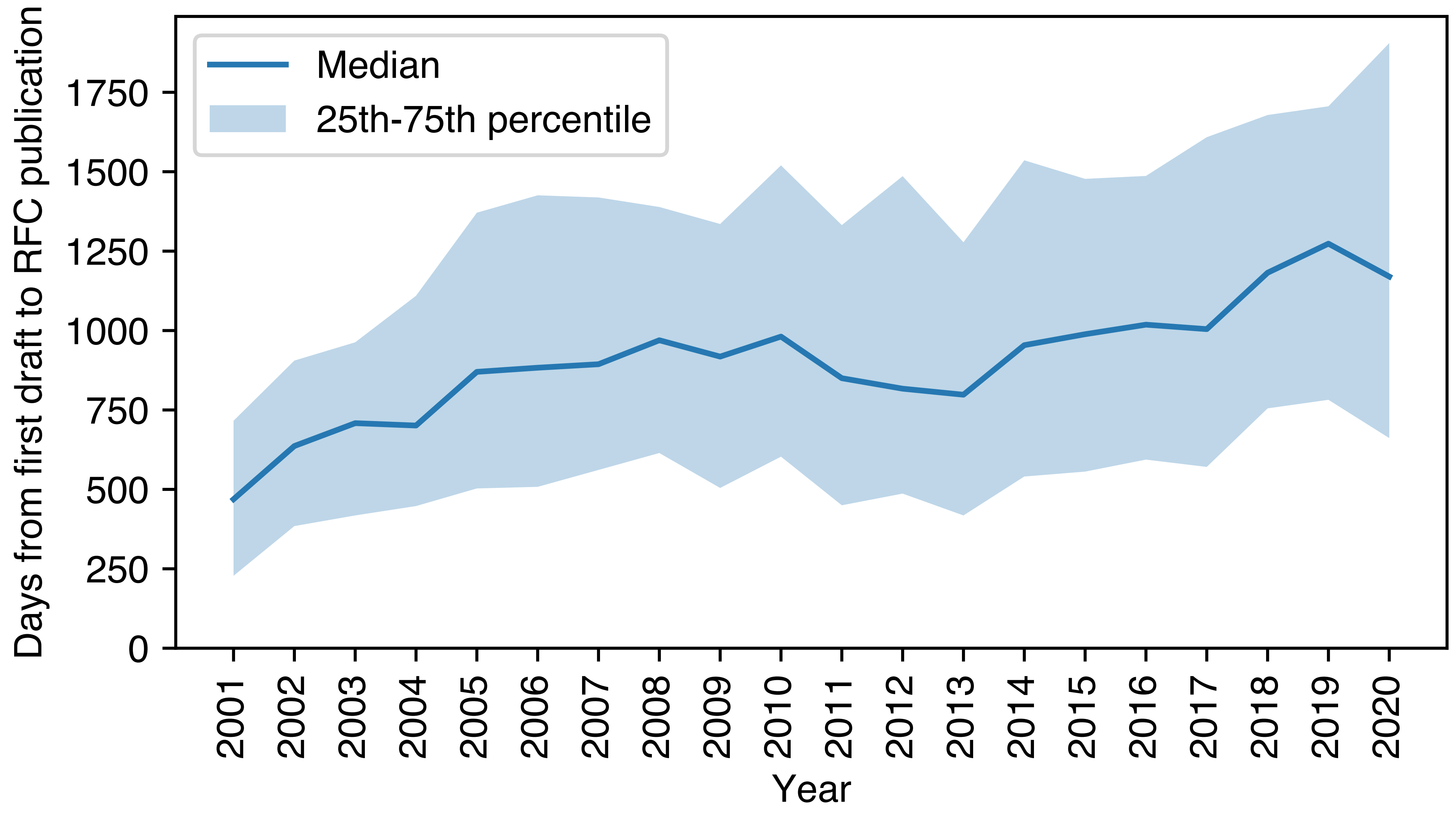
Effective standards require consensus

Consensus takes time

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

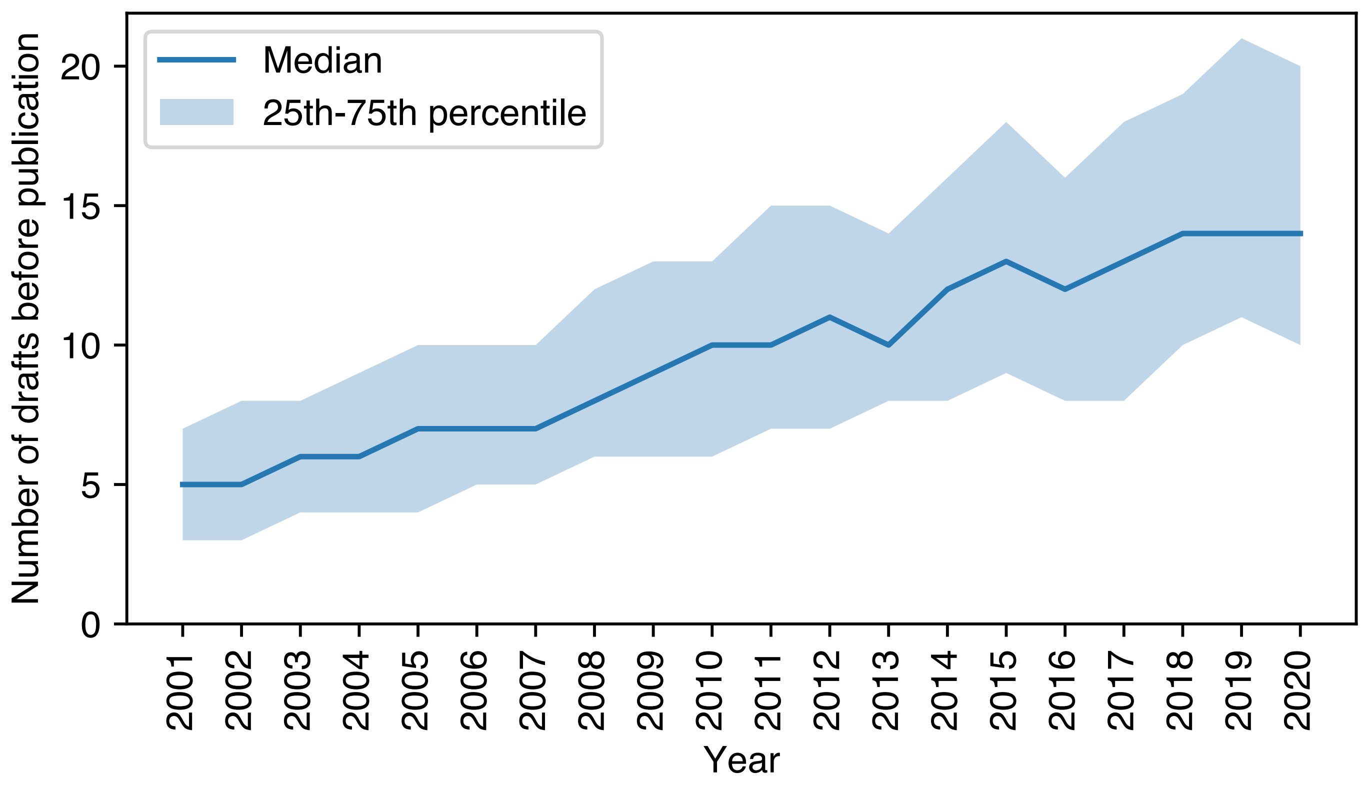
standards are getting slower

- IETF standards are taking longer to publish, but page counts remain broadly constant
- The median number of days to publication was 469 in 2001, rising to 1170 in 2022

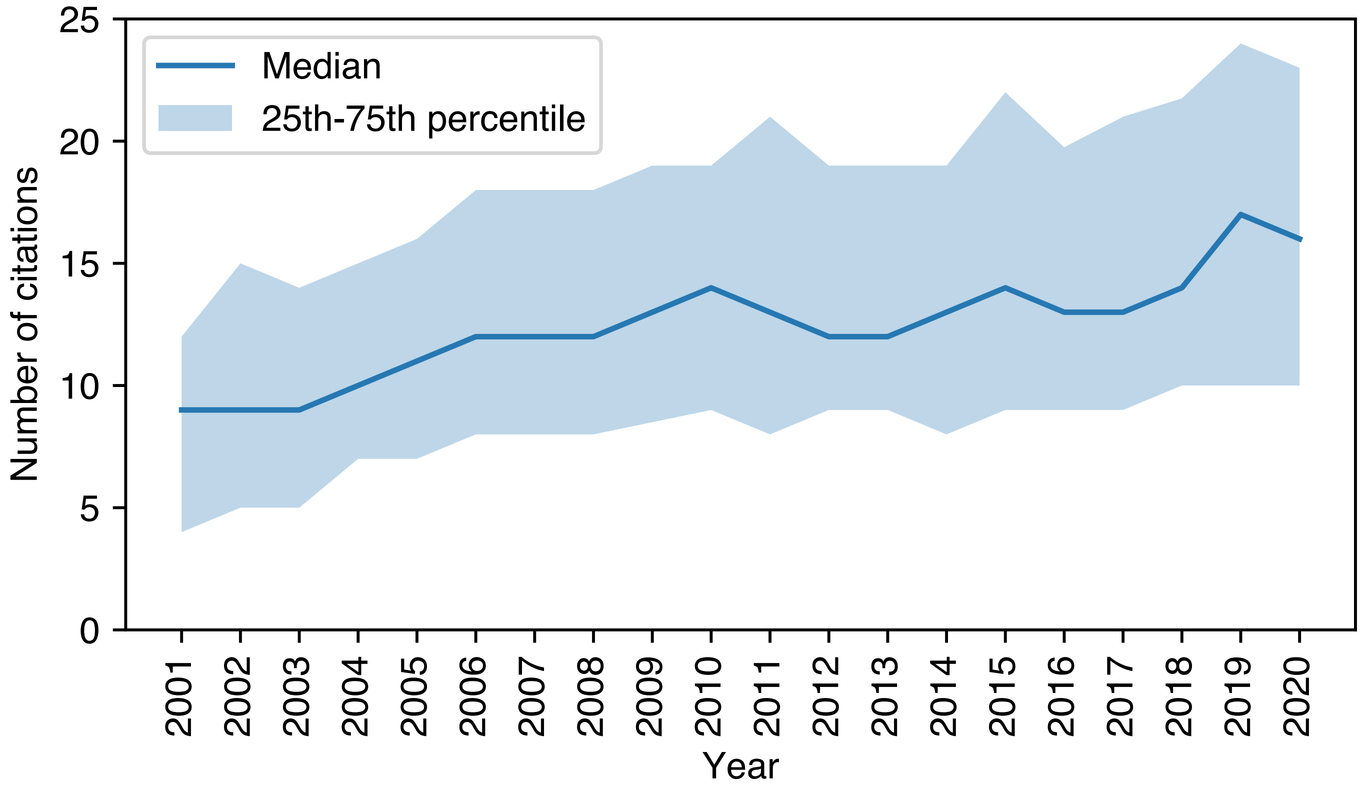


Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

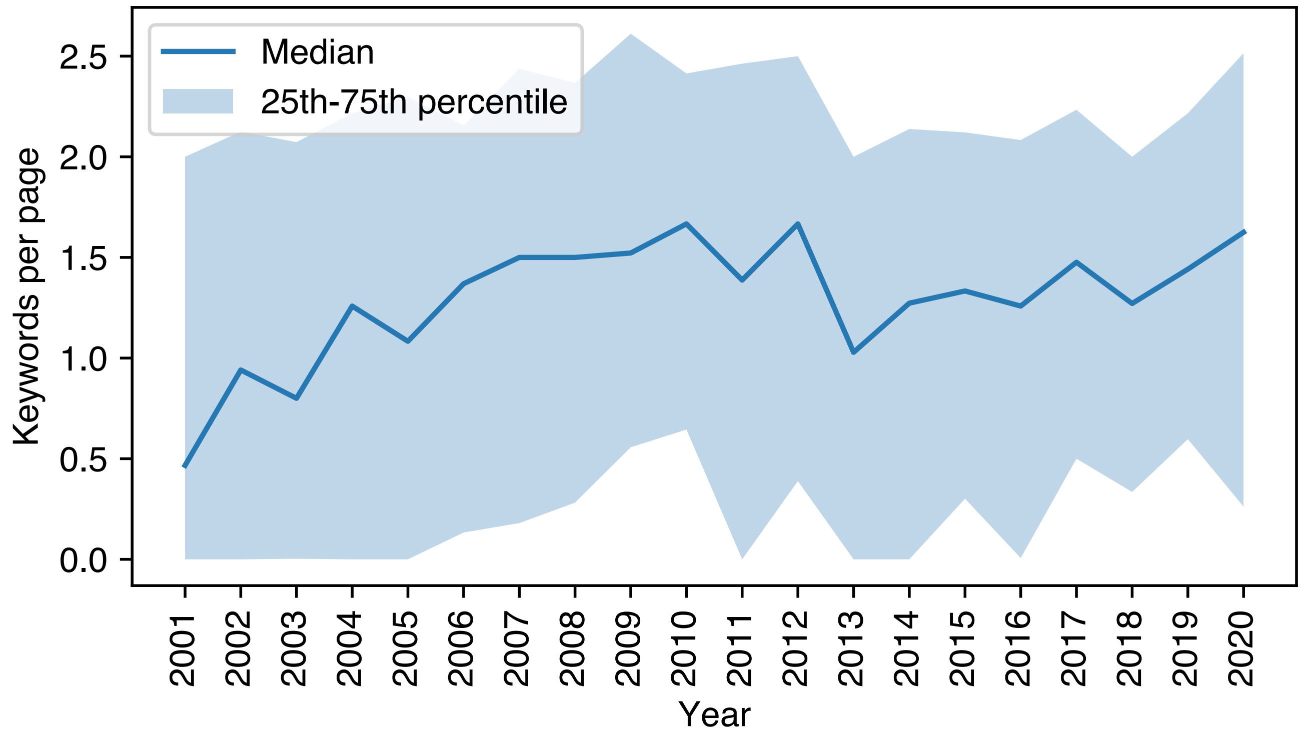
standards are getting slower



Median number of revisions made prior to publication has doubled



New drafts are citing increasing numbers of prior RFCs



Drafts are increasingly using normative language

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

This is **normal** and **expected**

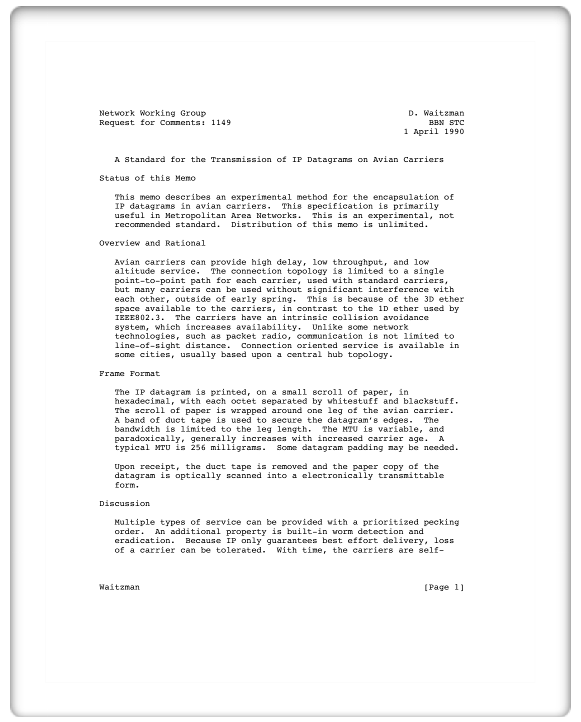
How to improve standards productivity?

- New directions
  - Standards development is slow due to the complexity of the deployed base
  - Continual reinvention solves the problem → new topics, new standards
- But the core infrastructure needs to evolve
  - Need better ways to build standards

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

Limiting Factors

- Consensus takes time
- Cognitive limits of standards developers



```
Network Working Group                                        D. Waitzman
Request for Comments: 1149                                       BBN STC
                                                            1 April 1990

   A Standard for the Transmission of IP Datagrams on Avian Carriers

Status of this Memo

   This memo describes an experimental method for the encapsulation of
   IP datagrams in avian carriers.  This specification is primarily
   useful in Metropolitan Area Networks.  This is an experimental, not
   recommended standard.  Distribution of this memo is unlimited.

Overview and Rational

   Avian carriers can provide high delay, low throughput, and low
   altitude service.  The connection topology is limited to a single
   point-to-point path for each carrier, used with standard carriers,
   but many carriers can be used without significant interference with
   each other, outside of early spring.  This is because of the 3D ether
   space available to the carriers, in contrast to the 1D ether used by
   IEEE802.3.  The carriers have an intrinsic collision avoidance
   system, which increases availability.  Unlike some network
   technologies, such as packet radio, communication is not limited to
   line-of-sight distance.  Connection oriented service is available in
   some cities, usually based upon a central hub topology.

Frame Format

   The IP datagram is printed, on a small scroll of paper, in
   hexadecimal, with each octet separated by whitestuff and blackstuff.
   The scroll of paper is wrapped around one leg of the avian carrier.
   A band of duct tape is used to secure the datagram's edges.  The
   bandwidth is limited to the leg length.  The MTU is variable, and
   paradoxically, generally increases with increased carrier age.  A
   typical MTU is 256 milligrams.  Some datagram padding may be needed.

   Upon receipt, the duct tape is removed and the paper copy of the
   datagram is optically scanned into a electronically transmittable
   form.

Discussion

   Multiple types of service can be provided with a prioritized pecking
   order.  An additional property is built-in worm detection and
   eradication.  Because IP only guarantees best effort delivery, loss
   of a carrier can be tolerated.  With time, the carriers are self-


Waitzman                                                        [Page 1]
```

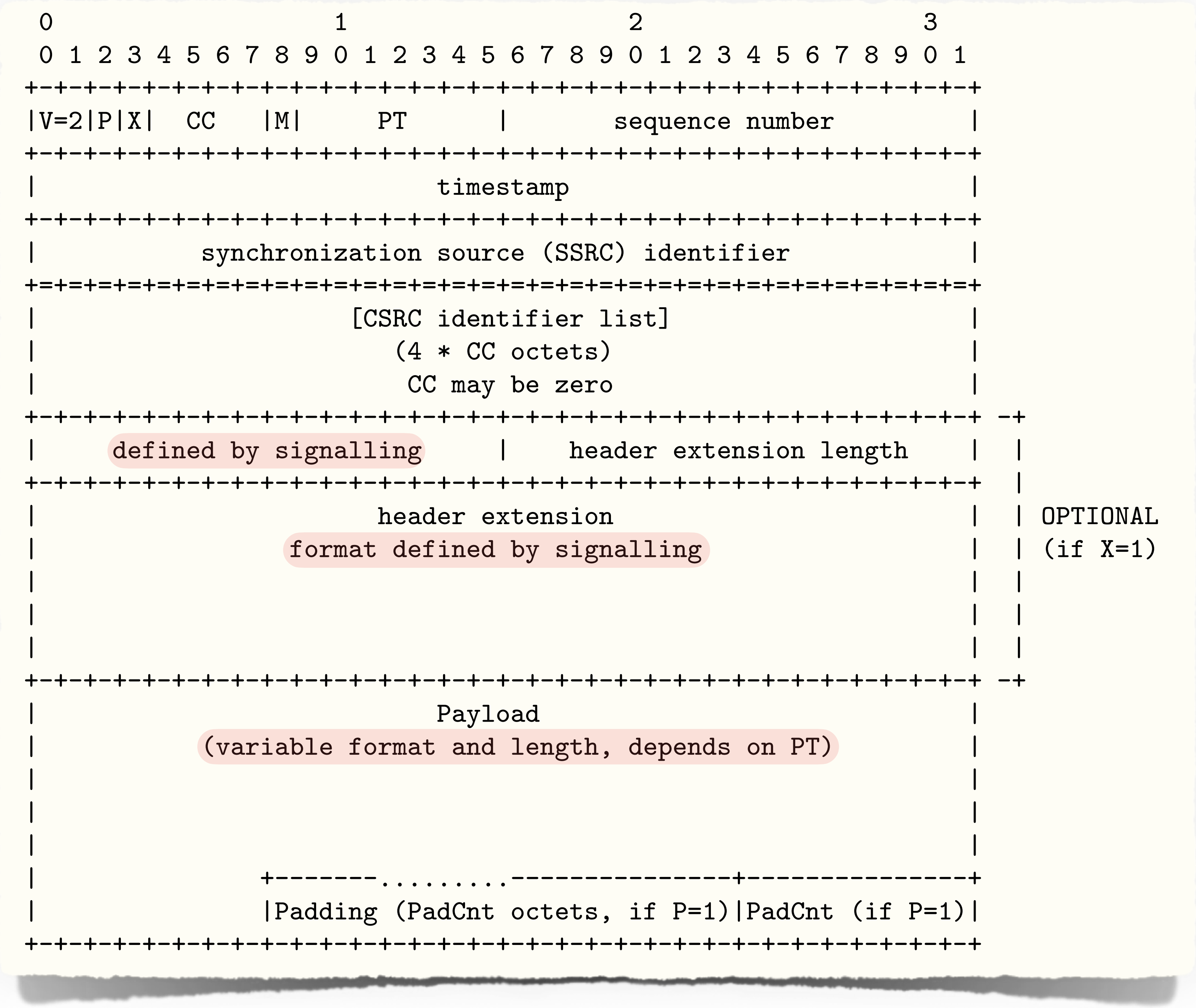
- Feedback loop between standards documents and consensus building
  - Improve the way we write documents
  - Improve the way we reach consensus

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

Improving the Standards Process

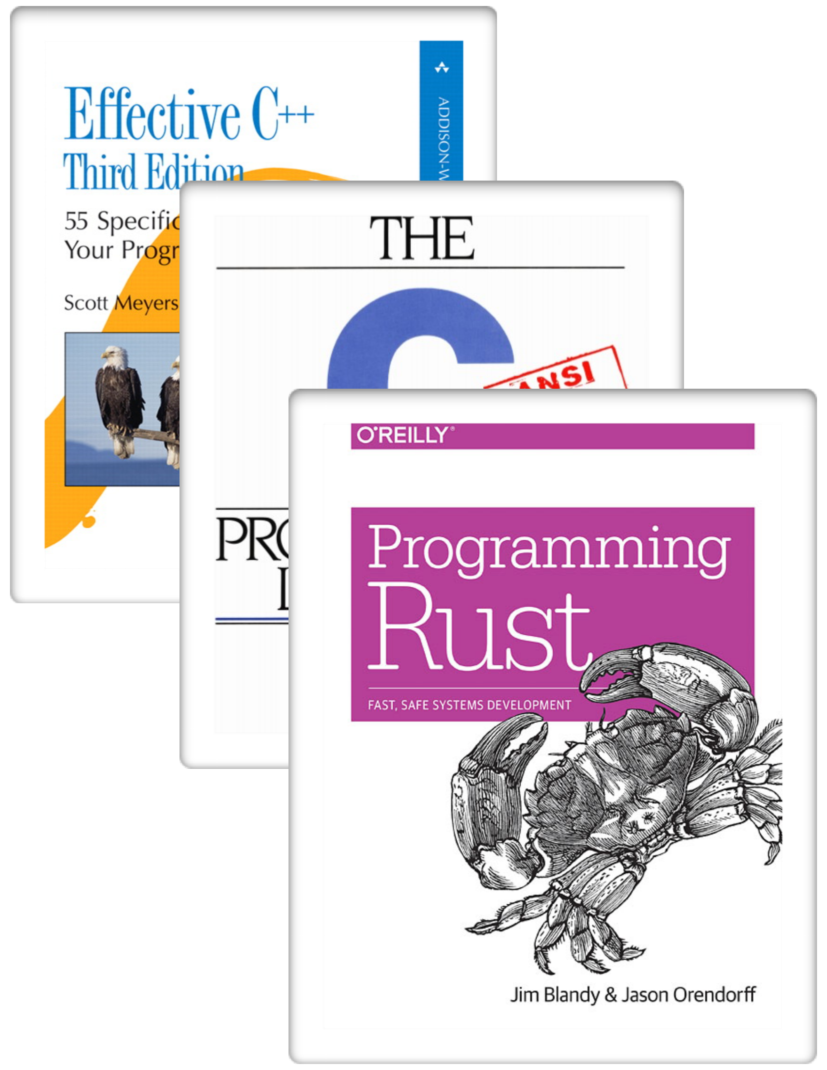
- Goal: increase testability of specifications, reduce cognitive load
  - Extract features from work-in-progress standards
    - Packet formats and parsers
    - State machines
  - Find bugs during development – not when PhD students run formal analysis tools on published standards
  - Make the idea of automated protocol testing and analysis a normal part of the standards process

Structured Parsing



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                    [CSRC identifier list]                     |
|                        (4 * CC octets)                        |
|                        CC may be zero                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ -+
|    defined by signalling      |    header extension length    |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|                       header extension                        |  | OPTIONAL
|                 format defined by signalling                  |  | (if X=1)
|                                                               |  |
|                                                               |  |
|                                                               |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ -+
|                            Payload                            |
|          (variable format and length, depends on PT)          |
|                                                               |
|                                                               |
|                                                               |
|               +-------.........---------------+---------------+
|               |Padding (PadCnt octets, if P=1)|PadCnt (if P=1)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Typed Intermediate Representation



- Impose enough structure on existing ad-hoc formats to be parsable – but not so much that they become hard to write by hand

- Basic type is a bit string
- Add **`struct`** and **`enum`**-like types
- Expressions to determine presence of optional fields
- Constraints on field sizes and values
- Persistent parsing state
- Transformation function

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

Representing State Machines (1/2)

- Session types can model protocol state machines

- A typed model of communication
  - Derive from text of protocol standards document – currently manual, aiming to automate
  - Type check to **ensure consistency of specification**

```
Γ  =  s[su] : ss ⊕ tcb_new(TcbInfo).ss&{
          error_no_room(ErrorInsufficientResources).end,
          tcb_created(SocketFd).μT.ss&{
              read_queue(Data).ss ⊕ {
                  write_queue(Data).T,
                  close_init(Close).ss&close_init(Close).ss&close_init(Close).end}
              close_init(Close).ss&close(Close).end
          },
          connection_aborted(Close).end},
      s[ss] : su&tcb_new(TcbInfo).su ⊕ {
          error_no_room(ErrorInsufficientResources).end,
          tcb_created(SocketFd).cs&syn(SynSet).cs ⊕ syn_ack(SynAckSet).μT.cs&{
              acceptable(AckSet).su ⊕ read_queue(Data).su&{
                  write_queue(Data).cs ⊕ {
                      acceptable(AckSet).T,
                      rto_exceeded(AckSet).cs&retransmit(AckSet).cs ⊕ {
                          ack(Ack).T,
                          retry_threshold_exceeded(RstSet).su ⊕ connection_aborted(Close).end}},
                  close_init(Close).ss ⊕ fin(FinSet).ss&{
                      fin_ack(FinAckSet).ss ⊕ ack(AckSet).cu ⊕ close(Close).end,
                      fin(FinSet).ss&ack(AckSet).ss ⊕ ack(AckSet).cu ⊕ close(Close).end}},
              rto_exceeded(AckSet).ss&retransmit(AckSet).ss ⊕ {
                  ack(Ack).T,
                  retry_threshold_exceeded(RstSet).su ⊕ connection_aborted(Close).end},
              fin(FinSet).su ⊕ close_init(Close).ss&ack(AckSet).ss ⊕ ack(AckSet).cu ⊕ close(Close).end}},
      s[cs] : cu&tcb_new(TcbInfo).cu ⊕ {
          error_no_room(ErrorInsufficientResources).end,
          tcb_created(SocketFd).ss ⊕ syn(SynSet).ss&syn_ack(SynAckSet).μT.cu&{
              write_queue(Data).ss ⊕ {
                  acceptable(AckSet).ss&{
                      acceptable(AckSet).cu ⊕ read_queue(Data).T,
                      fin(FinSet).cu ⊕ close_init(Close).cu&close_init(Close).ss ⊕ {
                          fin_ack(FinAckSet).ss&ack(AckSet).cu ⊕ close(Close).end,
                          fin(FinSet).ss&ack(AckSet).ss ⊕ ack(AckSet).cu ⊕ close(Close).end},
                      rto_exceeded(AckSet).ss ⊕ retransmit(AckSet).ss&{
                          ack(SegAckSet).T,
                          retry_threshold_exceeded(SegRstSet).cu ⊕ connection_aborted(Close).end}},
                  rto_exceeded(AckSet).ss ⊕ retransmit(AckSet).ss&{
                      ack(SegAckSet).T,
                      retry_threshold_exceeded(SegRstSet).cu ⊕ connection_aborted(Close).end},
                  fin(FinSet).ss&ack(AckSet).ss ⊕ ack(AckSet).cu ⊕ close(Close).end}},
      s[cu] : cs ⊕ tcb_new(TcbInfo).cs&{
          error_no_room(ErrorInsufficientResources).end,
          tcb_created(SocketFd).μT.cs ⊕ {
              write_queue(Data).cs&{
                  read_queue(Data)).T,
                  close_init(Close).client_system ⊕ close_init(Close).client_system&close(Close).end},
```

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

Representing State Machines (2/2)

- Session types can model protocol state machines
- A typed model of communication
  - Derive from text of protocol standards document – currently manual, aiming to automate
  - Type check to **ensure consistency of specification**
  - Derive Rust code automatically from typed model of the protocol
  - Type **check implementation for consistency with the specification**

```
pub type ServerSystemSessionType = St![
     (RoleServerUser & Open).
     (RoleServerUser + TcbCreated).
     (RoleClientSystem & Syn).
     (RoleClientSystem + SynAck).
     ServerSystemSynRcvd
];

Rec!(pub ServerSystemSynRcvd, [
     (RoleClientSystem & {
          Ack. // acceptable
               (RoleServerUser + Connected).
               ServerSystemCommLoop,
          Ack. // unacceptable
               (RoleClientSystem + {
                    Ack.ServerSystemSynRcvd,
                    Rst.(RoleServerUser + Close).end
               })
     })
]);
```

Lessons Learned (1/2)

Modelling real-world protocols is feasible

TCP packet parser derived from unmodified RFC 9293

TCP state machine derived from session type model

Proof-of-concept, but especially robust, but interwork with other TCP implementations

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

Lessons Learned (2/2)

- Packet formats have surprising complexity
  - Typed packet description languages feasible and integrate with existing standards process
- Session types have potential for protocol and implementation verification, but need integration with the standards process
  - Different styles of specification – formal vs. informal, but also difference in approach to specifying protocols
  - Session types model sequence of endpoint behaviour rather than state-based approach
  - Session types describe expected behaviour, RFC has more focus more on failure modes

- IRTF research group starting to think about these issues in the standards community:
  - Usable Formal Methods Research Group
  - https://www.irtf.org/ → UFMRG
  - https://datatracker.ietf.org/rg/ufmrg/about/
  - Would value more input from this community

Colin Perkins | https://csperkins.org/ | Copyright © 2024 University of Glasgow

More Information

Session Types for the Transport Layer: Towards an Implementation of TCP*

Samuel Cavoj
samuel@cavoj.net
University of Glasgow

Ivan Nikitin
ivan@niktivan.org
University of Glasgow

Colin Perkins
csp@csperkins.org
University of Glasgow

Ornela Dardha
ornela.dardha@glasgow.ac.uk
University of Glasgow

Session types are a typing discipline used to formally describe communication-driven applications with the aim of fewer errors and easier debugging later into the life cycle of the software. Protocols at the transport layer such as TCP, UDP, and QUIC underpin most of the communication on the modern Internet and affect billions of end-users. The transport layer has different requirements and constraints compared to the application layer resulting in different requirements for verification. Despite this, to our best knowledge, no work shows the application of session types at the transport layer. In this work, we discuss how multiparty session types (MPST) can be applied to implement the TCP protocol. We develop an MPST-based implementation of a subset of a TCP server in Rust and test its interoperability against the Linux TCP stack. Our results highlight the differences in assumptions between session type theory and the way transport layer protocols are usually implemented. This work is the first step towards bringing session types into the transport layer.

1 Introduction

Session types [11] are a typing discipline for communication protocols. They can describe the sequence of messages exchanged between participants over a communication channel and can be used to verify that the protocol is implemented correctly or has certain desirable properties. Further, session types can be realised within programming languages and used to type-check the implementation of a protocol against a session type definition, with type errors indicating inconsistencies between implementation and the session type. Session types have been an active area of research since the beginning of the 1990s [11] and have been implemented in a number of programming languages including C [26], Java [13] and Rust [14, 15] and other programming languages [9, 16, 25, 27, 29].

Network protocols that are part of the Internet Protocol suite (TCP/IP) are the foundation of the Internet. They are responsible for interoperability between different devices, operating systems, and applications. To ensure that different implementations of the same protocol are compatible, they must adhere to a technical specification which, in the case of Internet protocols, is defined in a series of documents, known as RFCs [8], developed by the Internet Engineering Task Force (IETF). Specifically, the latest version of the TCP protocol specification is defined in RFC 9293 [7].

The IETF follows a consensus-based process when developing standards [4, 30], with protocol specifications being developed in working group meetings and on mailing lists over a multi-year period. The resulting RFCs are written primarily in English prose, allowing the documents to be used in the

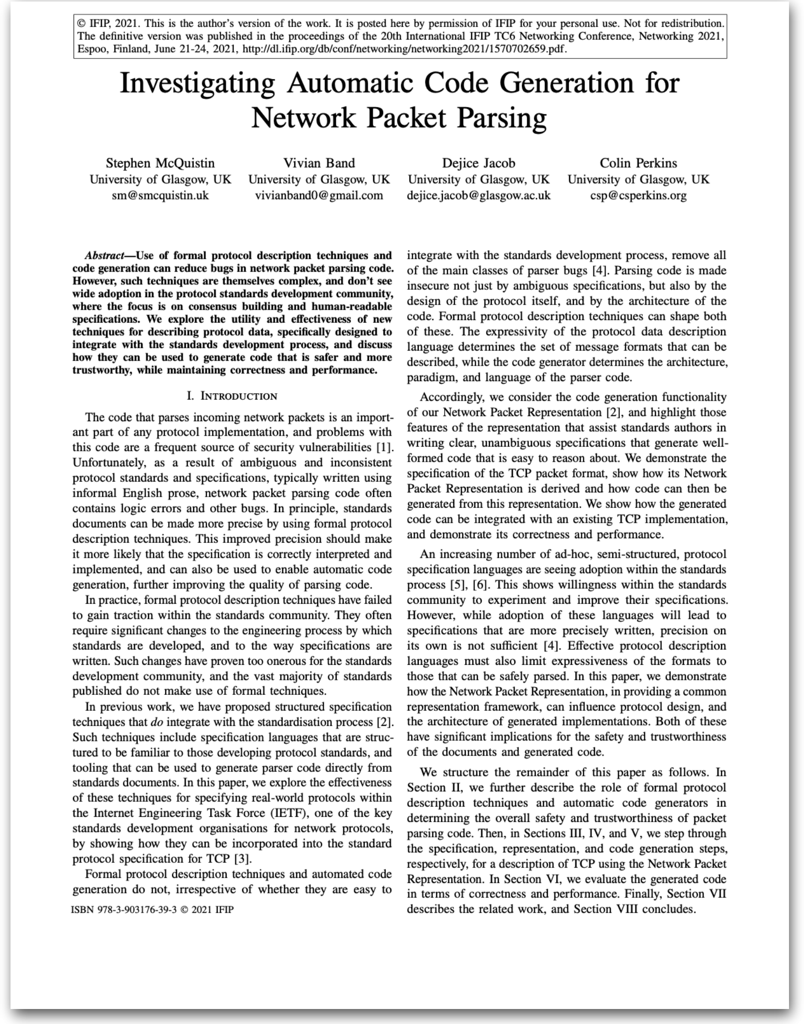
---

[*] Supported in part by the UK EPSRC grants EP/X027309/1 and EP/S036075/1.

D. Costa, R. Hu (Eds.): Programming Language Approaches to
Concurrency and Communication-cEntric Software 2024 (PLACES'24)
EPTCS 401, 2024, pp. 22–36, doi:10.4204/EPTCS.401.3

© S. Cavoj, I. Nikitin, C. Perkins, O. Dardha
This work is licensed under the
Creative Commons Attribution License.

S. Cavoj, I. Nikitin, C. S. Perkins, and O. Dardha, "Session Types for the Transport Layer: Towards an Implementation of TCP", PLACES, 2024. DOI:10.4204/EPTCS.401.3



© IFIP, 2021. This is the author's version of the work. It is posted here by permission of IFIP for your personal use. Not for redistribution. The definitive version was published in the proceedings of the 20th International IFIP TC6 Networking Conference, Networking 2021, Espoo, Finland, June 21-24, 2021, http://dl.ifip.org/db/conf/networking/networking2021/1570702659.pdf.

Investigating Automatic Code Generation for Network Packet Parsing

Stephen McQuistin
University of Glasgow, UK
sm@smcquistin.uk

Vivian Band
University of Glasgow, UK
vivianband0@gmail.com

Dejice Jacob
University of Glasgow, UK
dejice.jacob@glasgow.ac.uk

Colin Perkins
University of Glasgow, UK
csp@csperkins.org

***Abstract*—Use of formal protocol description techniques and code generation can reduce bugs in network packet parsing code. However, such techniques are themselves complex, and don't see wide adoption in the protocol standards development community, where the focus is on consensus building and human-readable specifications. We explore the utility and effectiveness of new techniques for describing protocol data, specifically designed to integrate with the standards development process, and discuss how they can be used to generate code that is safer and more trustworthy, while maintaining correctness and performance.**

I. INTRODUCTION

The code that parses incoming network packets is an important part of any protocol implementation, and problems with this code are a frequent source of security vulnerabilities [1]. Unfortunately, as a result of ambiguous and inconsistent protocol standards and specifications, typically written using informal English prose, network packet parsing code often contains logic errors and other bugs. In principle, standards documents can be made more precise by using formal protocol description techniques. This improved precision should make it more likely that the specification is correctly interpreted and implemented, and can also be used to enable automatic code generation, further improving the quality of parsing code.

In practice, formal protocol description techniques have failed to gain traction within the standards community. They often require significant changes to the engineering process by which standards are developed, and to the way specifications are written. Such changes have proven too onerous for the standards development community, and the vast majority of standards published do not make use of formal techniques.

In previous work, we have proposed structured specification techniques that *do* integrate with the standardisation process [2]. Such techniques include specification languages that are structured to be familiar to those developing protocol standards, and tooling that can be used to generate parser code directly from standards documents. In this paper, we explore the effectiveness of these techniques for specifying real-world protocols within the Internet Engineering Task Force (IETF), one of the key standards development organisations for network protocols, by showing how they can be incorporated into the standard protocol specification for TCP [3].

Formal protocol description techniques and automated code generation do not, irrespective of whether they are easy to integrate with the standards development process, remove all of the main classes of parser bugs [4]. Parsing code is made insecure not just by ambiguous specifications, but also by the design of the protocol itself, and by the architecture of the code. Formal protocol description techniques can shape both of these. The expressivity of the protocol data description language determines the set of message formats that can be described, while the code generator determines the architecture, paradigm, and language of the parser code.

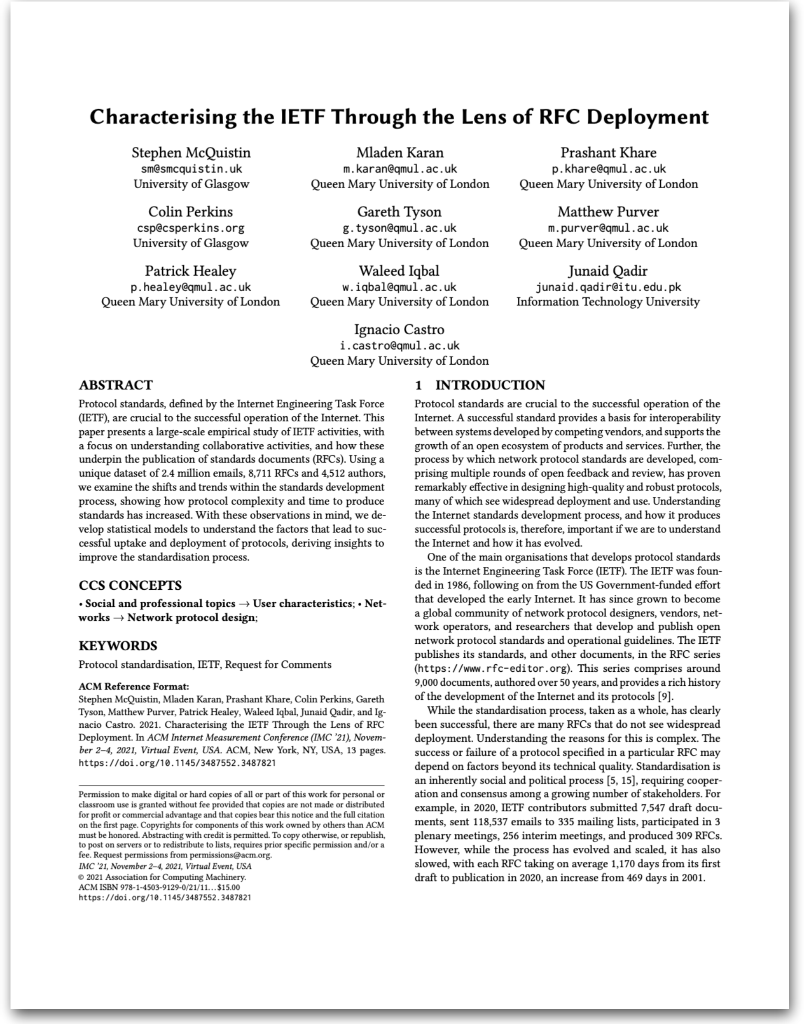
Accordingly, we consider the code generation functionality of our Network Packet Representation [2], and highlight those features of the representation that assist standards authors in writing clear, unambiguous specifications that generate well-formed code that is easy to reason about. We demonstrate the specification of the TCP packet format, show how its Network Packet Representation is derived and how code can then be generated from this representation. We show how the generated code can be integrated with an existing TCP implementation, and demonstrate its correctness and performance.

An increasing number of ad-hoc, semi-structured, protocol specification languages are seeing adoption within the standards process [5], [6]. This shows willingness within the standards community to experiment and improve their specifications. However, while adoption of these languages will lead to specifications that are more precisely written, precision on its own is not sufficient [4]. Effective protocol description languages must also limit expressiveness of the formats to those that can be safely parsed. In this paper, we demonstrate how the Network Packet Representation, in providing a common representation framework, can influence protocol design, and the architecture of generated implementations. Both of these have significant implications for the safety and trustworthiness of the documents and generated code.

We structure the remainder of this paper as follows. In Section II, we further describe the role of formal protocol description techniques and automatic code generators in determining the overall safety and trustworthiness of packet parsing code. Then, in Sections III, IV, and V, we step through the specification, representation, and code generation steps, respectively, for a description of TCP using the Network Packet Representation. In Section VI, we evaluate the generated code in terms of correctness and performance. Finally, Section VII describes the related work, and Section VIII concludes.

ISBN 978-3-903176-39-3 © 2021 IFIP

S. McQuistin, V. Band, D. Jacob, and C. S. Perkins, "Investigating Automatic Code Generation for Network Packet Parsing", IFIP Networking 2021. DOI: 10.23919/IFIPNetworking52078.2021.9472829



Characterising the IETF Through the Lens of RFC Deployment

Stephen McQuistin
sm@smcquistin.uk
University of Glasgow

Mladen Karan
m.karan@qmul.ac.uk
Queen Mary University of London

Prashant Khare
p.khare@qmul.ac.uk
Queen Mary University of London

Colin Perkins
csp@csperkins.org
University of Glasgow

Gareth Tyson
g.tyson@qmul.ac.uk
Queen Mary University of London

Matthew Purver
m.purver@qmul.ac.uk
Queen Mary University of London

Patrick Healey
p.healey@qmul.ac.uk
Queen Mary University of London

Waleed Iqbal
w.iqbal@qmul.ac.uk
Queen Mary University of London

Junaid Qadir
junaid.qadir@itu.edu.pk
Information Technology University

Ignacio Castro
i.castro@qmul.ac.uk
Queen Mary University of London

ABSTRACT

Protocol standards, defined by the Internet Engineering Task Force (IETF), are crucial to the successful operation of the Internet. This paper presents a large-scale empirical study of IETF activities, with a focus on understanding collaborative activities, and how these underpin the publication of standards documents (RFCs). Using a unique dataset of 2.4 million emails, 8,711 RFCs and 4,512 authors, we examine the shifts and trends within the standards development process, showing how protocol complexity and time to produce standards has increased. With these observations in mind, we develop statistical models to understand the factors that lead to successful uptake and deployment of protocols, deriving insights to improve the standardisation process.

CCS CONCEPTS

- **Social and professional topics → User characteristics**; - **Networks → Network protocol design**;

KEYWORDS

Protocol standardisation, IETF, Request for Comments

ACM Reference Format:
Stephen McQuistin, Mladen Karan, Prashant Khare, Colin Perkins, Gareth Tyson, Matthew Purver, Patrick Healey, Waleed Iqbal, Junaid Qadir, and Ignacio Castro. 2021. Characterising the IETF Through the Lens of RFC Deployment. In *ACM Internet Measurement Conference (IMC '21), November 2–4, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3487552.3487821

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

IMC '21, November 2–4, 2021, Virtual Event, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9129-0/21/11…$15.00
https://doi.org/10.1145/3487552.3487821

1 INTRODUCTION

Protocol standards are crucial to the successful operation of the Internet. A successful standard provides a basis for interoperability between systems developed by competing vendors, and supports the growth of an open ecosystem of products and services. Further, the process by which network protocol standards are developed, comprising multiple rounds of open feedback and review, has proven remarkably effective in designing high-quality and robust protocols, many of which see widespread deployment and use. Understanding the Internet standards development process, and how it produces successful protocols is, therefore, important if we are to understand the Internet and how it has evolved.

One of the main organisations that develops protocol standards is the Internet Engineering Task Force (IETF). The IETF was founded in 1986, following on from the US Government-funded effort that developed the early Internet. It has since grown to become a global community of network protocol designers, vendors, network operators, and researchers that develop and publish open network protocol standards and operational guidelines. The IETF publishes its standards, and other documents, in the RFC series (`https://www.rfc-editor.org`). This series comprises around 9,000 documents, authored over 50 years, and provides a rich history of the development of the Internet and its protocols [9].

While the standardisation process, taken as a whole, has clearly been successful, there are many RFCs that do not see widespread deployment. Understanding the reasons for this is complex. The success or failure of a protocol specified in a particular RFC may depend on factors beyond its technical quality. Standardisation is an inherently social and political process [5, 15], requiring cooperation and consensus among a growing number of stakeholders. For example, in 2020, IETF contributors submitted 7,547 draft documents, sent 118,537 emails to 335 mailing lists, participated in 3 plenary meetings, 256 interim meetings, and produced 309 RFCs. However, while the process has evolved and scaled, it has also slowed, with each RFC taking on average 1,170 days from its first draft to publication in 2020, an increase from 469 days in 2001.

S. McQuistin, M. Karan, P. Khare, C. S. Perkins, G. Tyson, M. Purver, P. Healey, W. Iqbal, J. Qadir, and I. Castro, "Characterising the IETF Through the Lens of RFC Deployment", ACM IMC 2021. DOI: 10.1145/3487552.3487821

Work is supported by EPSRC grants EP/X027309/1, EP/R04144X/1, EP/S036075/1, EP/S033564/1, and DTA studentships