Leveraging Textual Specifications for Automated Attack Discovery in Network Protocols

Cristina Nita-Rotaru

Khoury College of Computer Science

Northeastern University

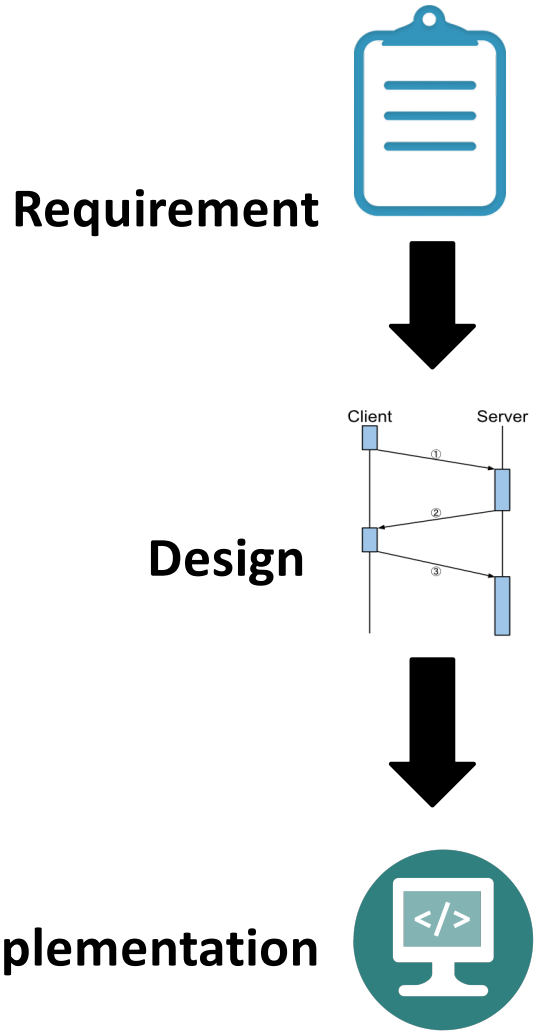
This work was supported by NSF grants CNS-1814105, CNS-1815219, and GRFP-1938052

How to get assurance about protocols?

- Proving security properties:
  Consider adversaries
  - Design
- Model checking:

  Check invariants
  - Design
  - Implementation
- Testing (fuzzying):

  Random inputs, grammar-based
  - Implementation
  - Deployement

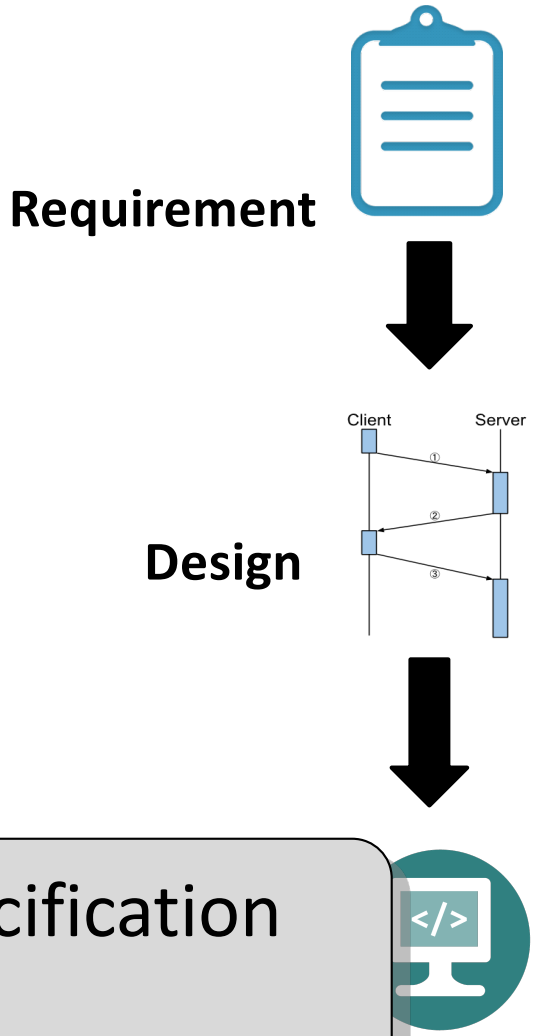


Our work in this space

- Provable security
  - QUIC [S&P 2015], TLS 1.3/TFO ESORICS 2019]
  - Secure routing [ACM CCR 2017]
  - Post-quantum [NDSS 2021]
- Compliance checking
  - IoT stacks, TLS: CHIRON [DSN 2017]
  - X509 certificates: SymCerts [S&P 2017]
  - NLP-learning of specifications: [IAAI 2019], [S&P 2022]
- Adversarial testing
  - State machine replication: Gatling [NDSS 2012], Turret [ICDCS 2013]
  - Routing: Turret-W [WiSec 2012, TON 2016]
  - SDN: BEADS [RAID 2017,USENIX SECURITY 2017]
  - TCP: Snake [DSN 2015], TCPwn [NDSS2018], BBR [RAID 2020]

Challenges

- Specification of requirements
  - Uneven culture of formally defining goals across research communities
  - Difficulty of formalizing some goals
- Assurance of designs
  - Models do not capture optimizations and other choices made at implementation time
- Assurance of implementations
  - Providing testcase coverage while addressing scalability and complexity
  - Ensuring testing in realistic scenarios similar to deployment conditions
- Machine learning will add complexity and attack vectors

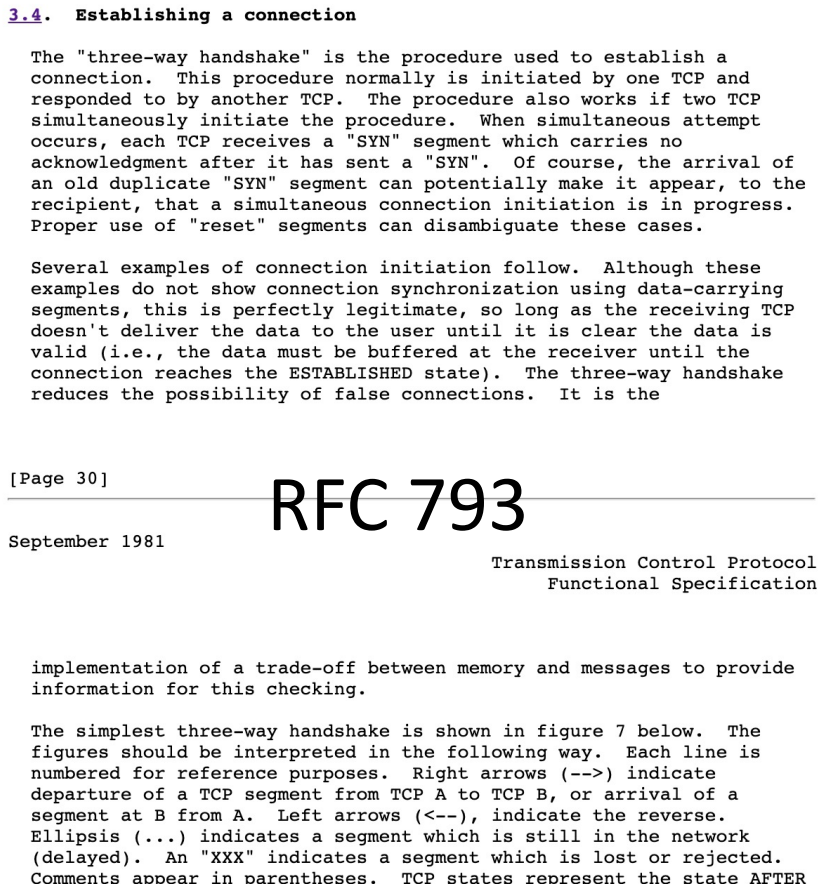


It is important to have a formal specification of the protocol to begin with

~~(Formal)~~ Specification of Internet Protocols

"RFC documents contain technical specifications and organizational notes for the Internet."

- Produced by the IETF, describe the main Internet's protocols such as addressing, routing, transport, or secure protocols such as TLS 1.3 and QUIC
- Statuses: Internet Standard, Proposed Standard, Best Current Practice, Experimental, Informational, and Historic



3.4. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the

[Page 30]

RFC 793

September 1981

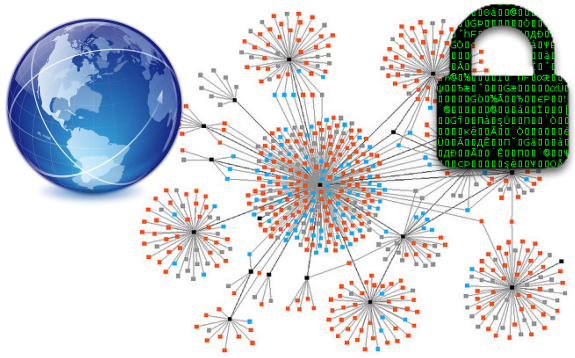
Transmission Control Protocol
Functional Specification

implementation of a trade-off between memory and messages to provide information for this checking.

The simplest three-way handshake is shown in figure 7 below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Ellipsis (...) indicates a segment which is still in the network (delayed). An "XXX" indicates a segment which is lost or rejected. Comments appear in parentheses. TCP states represent the state AFTER

TCP

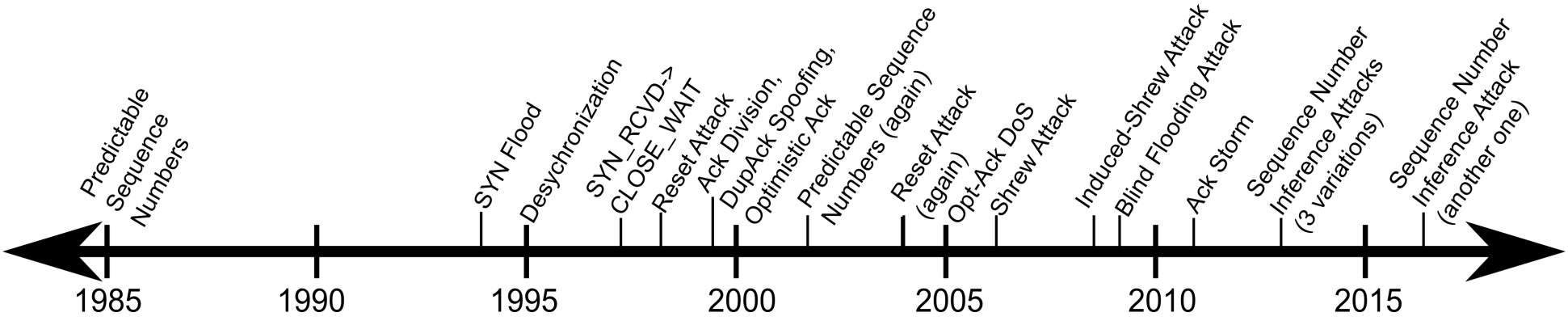
- Transport protocol used by vast majority of Internet traffic
  - Including traffic encrypted with TLS
  - Including network infrastructure protocols like BGP
- Thousands of implementations
  - Over 5,000 implementation variants detectable by nmap
- Provides:
  - Reliability
  - In-order delivery
  - Flow control
  - Congestion control



TCP functionality is described in over 20 RFCs

RFC 2861
RFC 5827
RFC 6937
RFC 3708
RFC 4653
RFC 5682
RFC 793
RFC 5681
RFC 2581
RFC 2001
RFC 6298
RFC 6582
RFC 2883
RFC 6528
RFC 7323
RFC 3390
RFC 3465
RFC 2018
RFC 3042
RFC 6675
RFC 4015

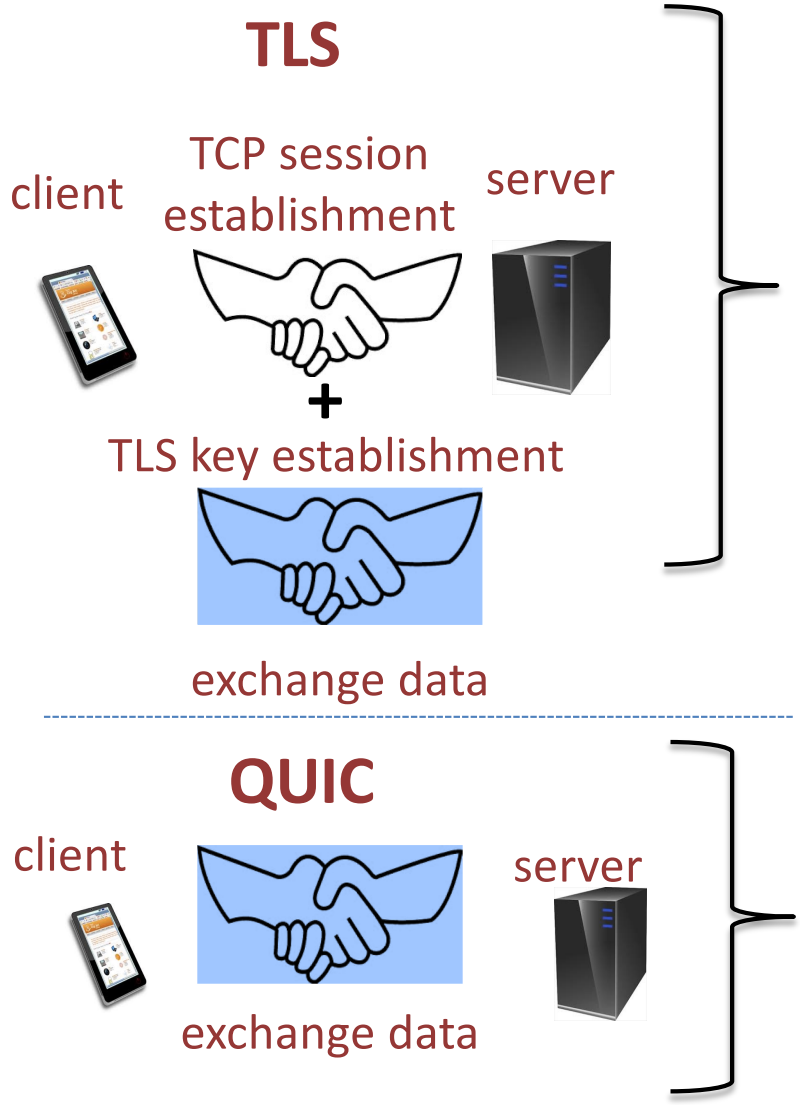
… attacked for over 35 years!



Leveraging State Information for Automated Attack Discovery in Transport Protocol Implementations. S. Jero, H. Lee, and C. Nita-Rotaru. In the 45th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), June 2015. **Best paper award.**

QUIC

- Protocol developed by Google and implemented in Chrome since 2013
- Main goal is reducing latency
- Design goals
  - Provide security protection comparable to TLS
  - Reduce connection latency by collapsing TCP and TLS functionality in one layer: requires UDP
  - Lists performance of connection establishment (0-RTT) as a goal

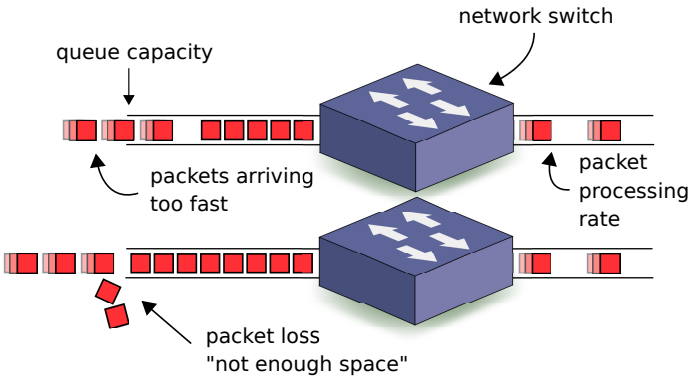


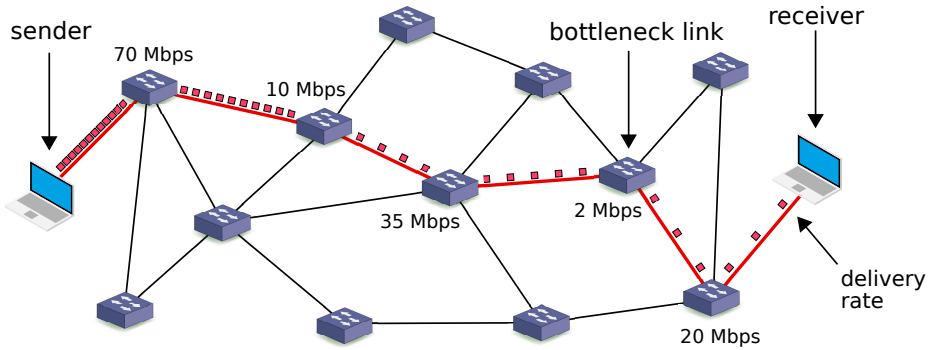
… deployed without security analysis

- The initial document provided some informal arguments
  - Specification was incomplete
  - Did not include any formal security analysis
- Security proofs were done by the research community based on analyzing the code
  - Fischlin & Günther, *ACM CCS* 2014
  - **How Secure and Quick is QUIC? Provable Security and Performance Analyses.** R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru. 36th IEEE Symposium on Security and Privacy (Oakland), May 2015. **Awarded IETF/IRTF Applied Networking Research Prize, 2016.**
- QUIC was standardized in May 2021, as RFC 9000

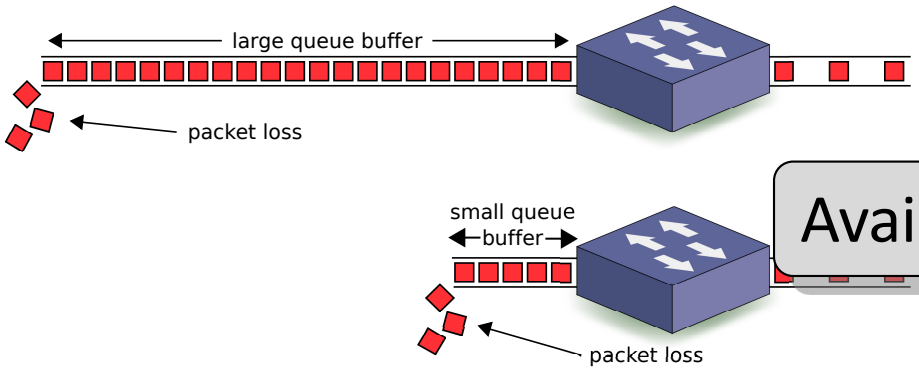
BBR Congestion Control

Packet loss as congestion signal

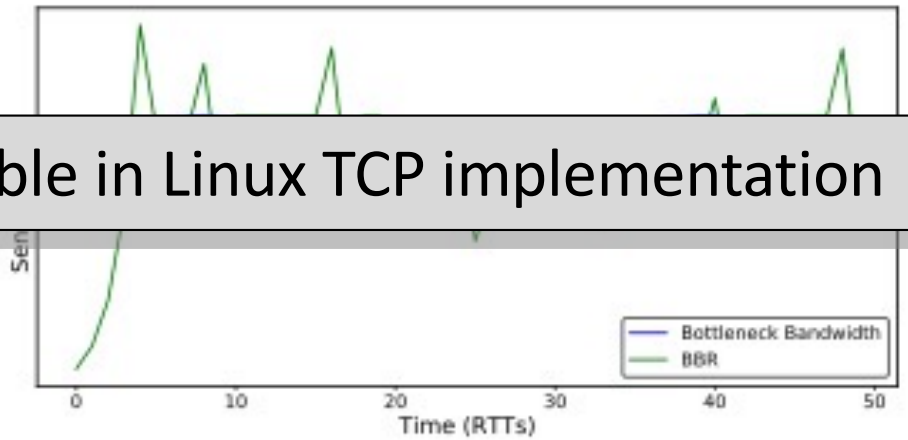




How it works: Some queue along path overflowed due to congestion



What is actually needed is to estimate the bottleneck link and not send faster than that

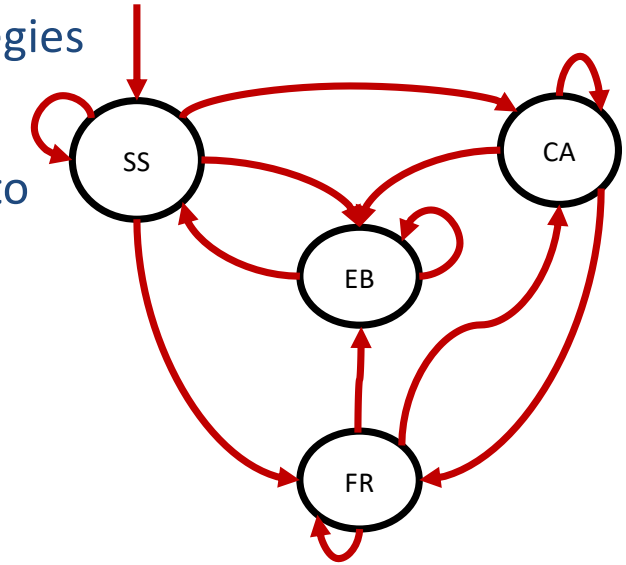


Available in Linux TCP implementation

In modern networks: less effective

TCPwn: Automated attacks against congestion control

- Use **model-based testing** to identify all possible attacks in a *scalable* manner
  - Use an abstract model to generate abstract strategies
  - Map abstract strategies to concrete strategies
  - Execute concrete strategies on implementations to find attacks causing:
    - Decreased throughput
    - Increased throughput
    - Connection stall



Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach. Samuel Jero, Endadul Hoque, David Choffnes, Alan Mislove, Cristina Nita-Rotaru. NDSS 2018, Feb. 2018. **CISCO Network Security Distinguished Paper Award.**

… TCPwn applied on BBR

Attack class		Impact
Optimistic acknowledgments		Faster
Delayed acknowledgments		Slower
Repeated Re-transmission timeout		Slower
Re-transmission timeout stall		Stalled
Sequence number de-sync stall		Stalled

aBBRate: Automating BBR Attack Exploration Using a Model-Based Approach Anthony Peterson, Samuel Jero, Endadul Hoque, Dave Choffnes, Cristina Nita-Rotaru. RAID 2020.

We had to derive the model from the code ….

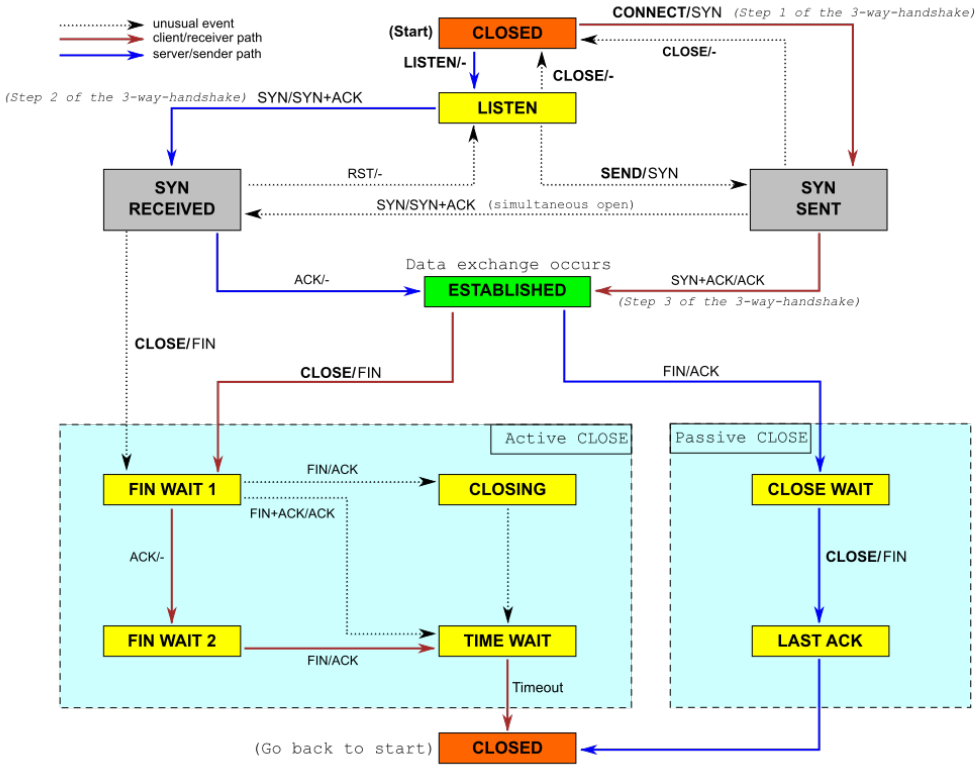
BBR is currently an Internet Draft not standardized yet

Why so many attacks?

- Most of these protocols have complex goals
- Many designs and implementations
- Written in low level languages, that are highly efficient, but error-prone
- Heavily optimized
- Specifications incomplete, have ambiguities or contradictory requirements
- Often implemented and deployed before specified, i.e. the code is the specification

Modeling protocols with FSMs

- Model of computation defined by a list of states, the initial state, and the inputs that trigger each transition
  - Change from one state to another is called a *transition*
  - For protocols events are sending , receiving messages and timeouts
- Protocol fuzzying, model checking, attack synthesis rely or may benefit from FSM



Obtaining an FSM

3.4. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

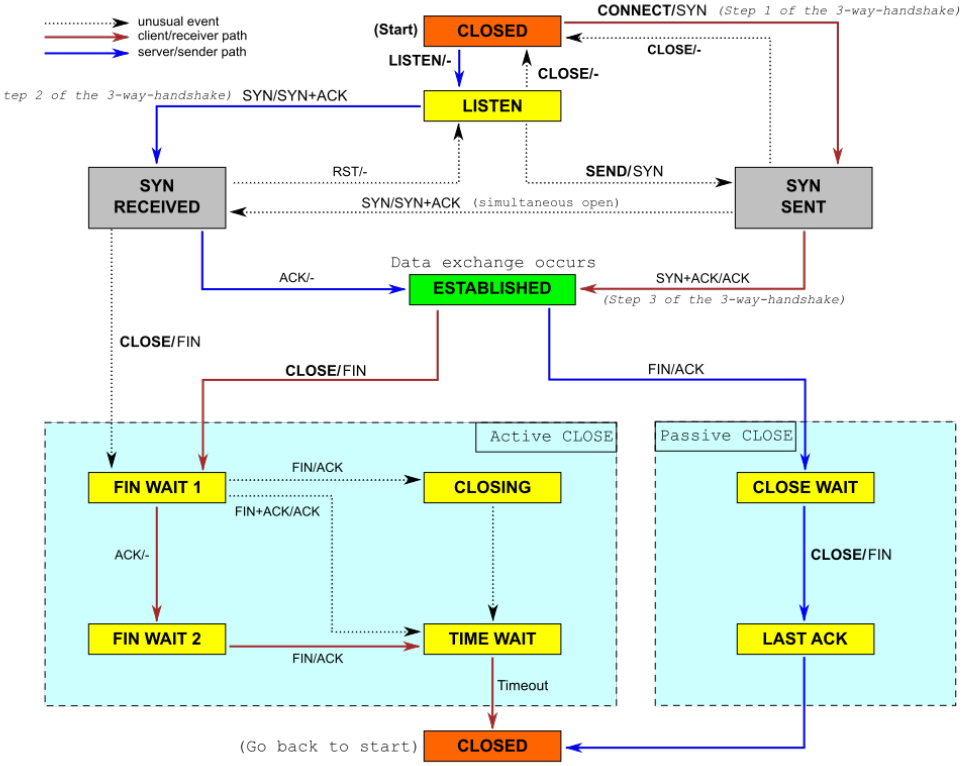
Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the

[Page 30]

```
1  // SPDX-License-Identifier: GPL-2.0-or-later
2  /*
3   * INET         An implementation of the TCP/IP protocol suite for the LINUX
4   *              operating system.  INET is implemented using the  BSD Socket
5   *              interface as the means of communication with the user level.
6   *
7   *              Implementation of the Transmission Control Protocol(TCP).
8   *
9   * Authors:     Ross Biro
10  *              Fred N. van Kempen, <waltje@uWalt.NL.Mugnet.ORG>
11  *              Mark Evans, <evansmp@uhura.aston.ac.uk>
12  *              Corey Minyard <wf-rch!minyard@relay.EU.net>
13  *              Florian La Roche, <flla@stud.uni-sb.de>
14  *              Charles Hedrick, <hedrick@klinzhai.rutgers.edu>
15  *              Linus Torvalds, <torvalds@cs.helsinki.fi>
16  *              Alan Cox, <gw4pts@gw4pts.ampr.org>
17  *              Matthew Dillon, <dillon@apollo.west.oic.com>
18  *              Arnt Gulbrandsen, <agulbra@nvg.unit.no>
```

Manual analysis

How to automatically derive FSM ???



Manual analysis

Automated FSM construction from traces

This talk

Can we automatically extract formal specifications of protocols (FSM) from RFC?

Can we synthesize attacks based on the extracted FSM?

Automated Attack Synthesis by Extracting Finite State Machines from Protocol Specification Documents Maria Leonor Pacheco, Max von Hippel, Ben Weintraub Dan Goldwasser Cristina Nita-Rotaru. IEEE Security and Privacy, 2022.

https://github.com/RFCNLP

What's in an RFC?

3.4. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the

[Page 30]

Challenges: How to define the NLP problem

- NLP semantic parsing studies methods for translating natural language into a complete formal representation
- RFCs do not contain canonical/reference FSMs, they have mistakes, omissions, ambiguities solved by human experts
- Unlike traditional NLP semantic parsing problems in our setting there is not a complete one-to-one translation between the text and the FSM

How to define a problem we could measure if we succeeded or not?

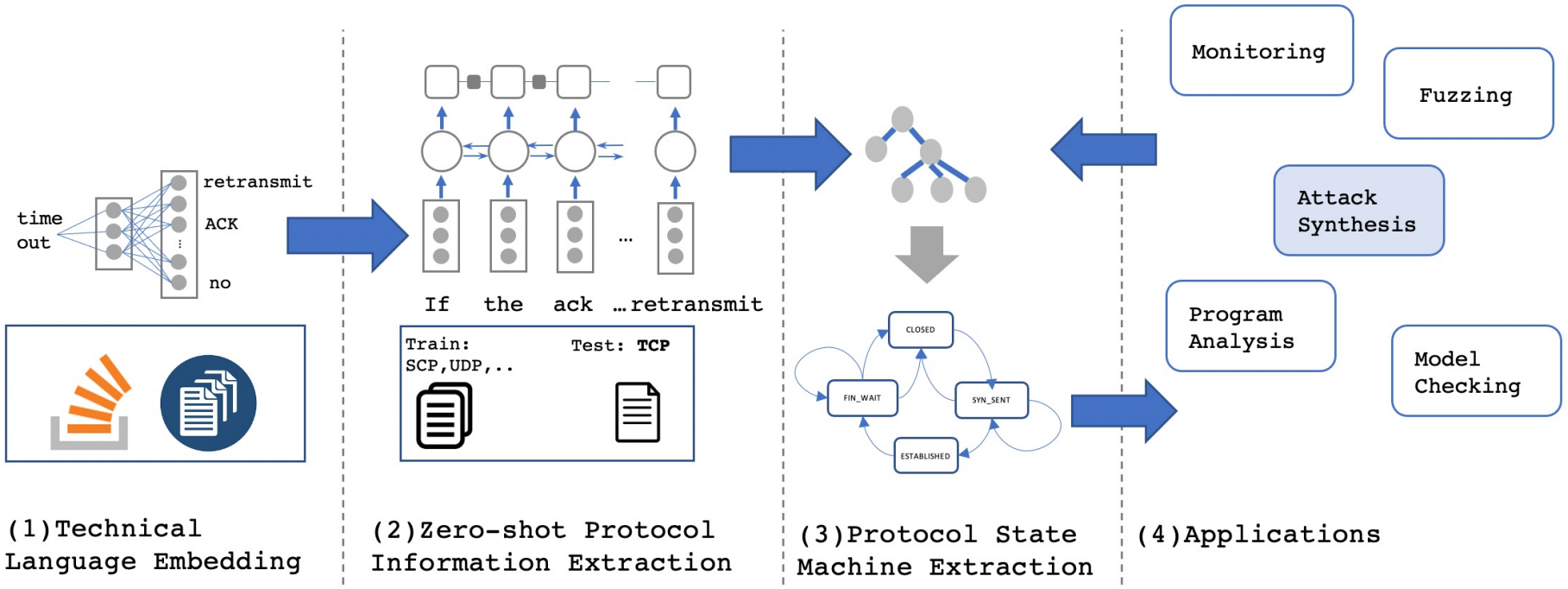
Challenges: What NLP approach to choose?

- **Off-the-shelf NLP tools**: Already available typically trained over news documents
  - When applied to technical documents that include many out-of-vocabulary words (i.e. technical terms), their performance degrades substantially
- **Rule-based systems:** Developed to support information extraction based on the specific format of the textual input.
  - Different RFC documents define variables, constraints, and temporal behaviors totally differently
  - Training such systems from scratch requires significant human effort annotating data with the relevant labels, which could be different fo

Which approach to choose?

Our approach

1. Learn large-scale word-representation for technical language with off-the-shelf tools
2. Define and learn protocol-independent information language from RFC with focused zero-shot learning to adapt to new, unobserved protocols without re-training
3. Use rule-based mapping from protocol-independent information to a protocol FSM

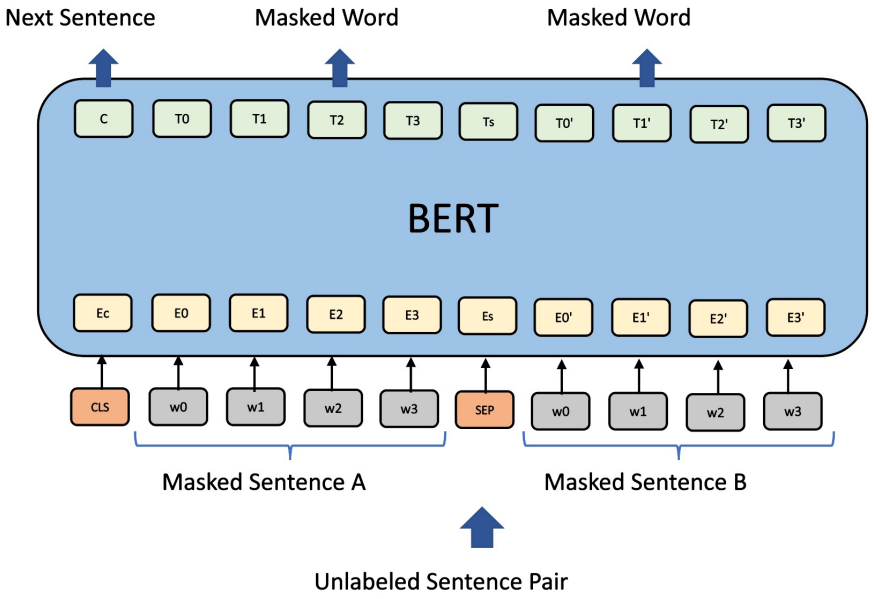


Learn distributed word representations

- **Static word representations:** learn a single vector for each word form
- **Contextualized representations:** allow the same word form to take different meanings in different context; compute different vectors for each mention
  - Example: "The connection is in error and should be reset with Reset Code 5"
- State-of-the art: BERT is an neural architecture based on a Transformer that computes a contextualized representation
- BERT models were pre-trained on the Books Corpus (800M words) and English Wikipedia (2,500M words) and are publicly available

BERT uses two learning strategies:

- *masked language modeling*: masks 15% of the words in each sentence and attempts to predict them
- *next sentence prediction*: uses pairs of sentences as input, and learns to predict whether the second sentence is the subsequent sentence



Use BERT for technical language embedding

- Goal is to learn distributed word representations for technical language
- It was shown that further training on technical language improves performance
- We pre-train BERT using the masked language model and the next sentence prediction objective using networking data
- **No supervision needed for this step**

Dataset

- Full set of RFC documents publicly available in ietf.org and rfc-editor.org
- Documents cover different aspects of computer networking, including protocols, procedures, programs, concepts, meeting notes and opinions.
- 8,858 documents and about 475M words

Our general Protocol Grammar

- We use this grammar to define the protocol semantic we want to extract from RFCs
- Captures the semantic of a protocol FSM
- General enough that is applies to many protocols
- We use it to annotate the RFC
- Four types of annotation tags:
  - Definition tags
  - Reference tags
  - State machine tags
  - Control flow tags

```
bool       ::= true | false
type       ::= send | receive | issue
def-tag    ::= def_state | def_var | def_event
ref-state  ::= ref_state id="##"
ref-event  ::= ref_event id="##" type="type"
ref-tag    ::= ref-event | ref-state
def-atom   ::= <def-tag>engl</def-tag>
sm-atom    ::= <ref-tag>engl</ref-tag> | engl
sm-tag     ::= trigger | variable | error | timer
act-atom   ::= <arg>sm-atom</arg> | sm-atom
act-struct::= act-struct | act-struct act-atom
trn-arg    ::= arg_source | arg_target | arg_inter
trn-atom   ::= <trn-arg>sm-atom<trn-arg> | sm-atom
trn-struct::= trn-struct | trn-struct trn-atom
ctl-atom   ::= <sm-tag>sm-atom</sm-tag>
             | <action type="type">act-struct</action>
             | <transition>trn-struct</transition>
             | sm-atom
ctl-struct::= ctl-atom | ctl-struct ctl-atom
ctl-rel    ::= relevant=bool
control    ::= <control ctl-rel>ctl-struct</control>
e          ::= control | ctl-atom | def-atom
             |  e_0 e_1
```

Definition tags

- *Whey we need them?* Main entities related to a protocol, used to annotate the names of **states, events, and variables**
- **State definition.** We annotate it when the name of a state is introduced in the text

  <def_state id="##">IDLE</def_state>, where ## is replaced by the identifier

- **Event definition.** Same annotation conventions as states

  <def_event id="##">

- **Variable definition.** Defined in a similar way to events and states, but they do not include identifiers because they are not explicitly referenced by annotation in the rest of the text

Reference tags

- *Why we need them?* When an event or state occurs in the text, it must be linked to an event or state which was tagged
  - RFC may formally refer to one event as “ACK”, but throughout the text these ACKs may also be referred to as “acknowledgments”
- **State reference.** States are referenced by surrounding the state’s name with the <ref_state id="##"> tag, where ## corresponds to the appropriate SID that was included with the state’s <def_state> tag

  enter <ref_state id="2">SYN-SENT</ref_state> state
- **Event reference.** Same convention as state references. The event reference must also include the *type* of event, where the three possible types are: send, receive, and compute

  a <ref_event type="send" id="10">SYN</ref_event> segment

State machine tags

- **Transition**. Denotes a state change. We use argument tags <arg_source>, <arg_target> and <arg_intermediate> to specify the segment in the text

  <transition>The server moves from the <arg_source>OPEN state</arg_source>, possibly through the <arg_inter>CLOSEREQ state</arg_inter>, to <arg_target>CLOSED</arg_target></transition>

  "OPEN", "CLOSEREQ" and "CLOSED" would also be enclosed in a <ref_state> tag.

- **Variable**. Certain variables may be tracked as part of the state machine.

  <variable>SND.UP <- SND.NXT-1</variable>

- **Timer**. This tag is used if a timer value is changed or set

  <timer>start the time-wait timer</timer>

- **Error**. If a context results in an error or warning being thrown

  <error>signal the user error: connection aborted due to user timeout</error>

- **Action**. Three types of actions: *send*, *receive* and *issue*. We use an argument tag <arg> to specify the argument in the text being sent, received or computed.

  <action type="send">Send <arg>a SYN segment</arg></action>.

  SYN" would also be enclosed in a <ref_event> tag: <ref_eventid="10">SYN</ref_event>

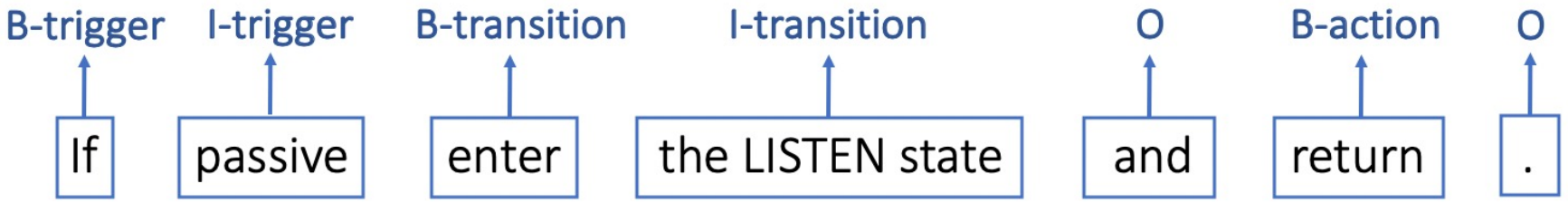
Control flow tags

- *Why we need them?* To indicate that some flow control or conditional logic is about to follow
- Flow control logic contains a <trigger> tag, which captures the event that triggers some action in the state machine, followed by one or more of the state machine tags
- A single block of control tags may contain multiple state machine tags
  - Multiple state machine tags organized in a list. In this case, the implication is that the state machine tags should all be executed if the initial trigger condition is true

```
<control>
  <trigger>
    if active and the foreign socket is
    specified,
  </trigger>
  <action type="issue">
    issue <arg>a <ref_event id="10">SYN</ref_event>
        segment</arg>.
  </action>
  <variable>
    An initial send sequence number (ISS) is
    selected.
  </variable>
  <action type="send">
    A <arg><ref_event id="10">SYN</ref_event>
        segment of the form
    <SEQ=ISS><CTL=SYN></arg> is sent.
  </action>
  <variable>
    Set SND.UNA to ISS, SND.NXT to ISS+1,
  </variable>
  <transition>
    enter the <arg_target><ref_state
        id="2">SYN-SENT</ref_state>
        state<arg_target>
  </transition>
</control>
```

Protocol information extraction

- **Zero-shot approach**: Have a system that can adapt to new, unobserved protocols without re-training the system
- We build on our general protocol grammar and the technical language embedding learnt with BERT and further trained on protocols RFC dataset (about 9000 documents).
- We identify a list of protocols and annotate them with our grammar
- *Our goal is to parse the document according to our grammar,* using a *sequence-to-sequence model*
- We segment paragraphs into smaller units (individual words, chunks or phrases). Then, we map each unit to a particular tag



Linear-Chain Conditional Random Fields (LINEARCRF)

- Uses a set of extracted features over each chunk
- Conditional Random Fields model the prediction as a probabilistic graphical model
- Chain Conditional Random Fields specifically consider sequential dependencies in the predictions
- Let y be a tag sequence and x an input sequence of textual units, maximize the conditional probability:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{p(\boldsymbol{y}, \boldsymbol{x})}{\sum_{\boldsymbol{y'}} p(\boldsymbol{y'}, \boldsymbol{x})}$$

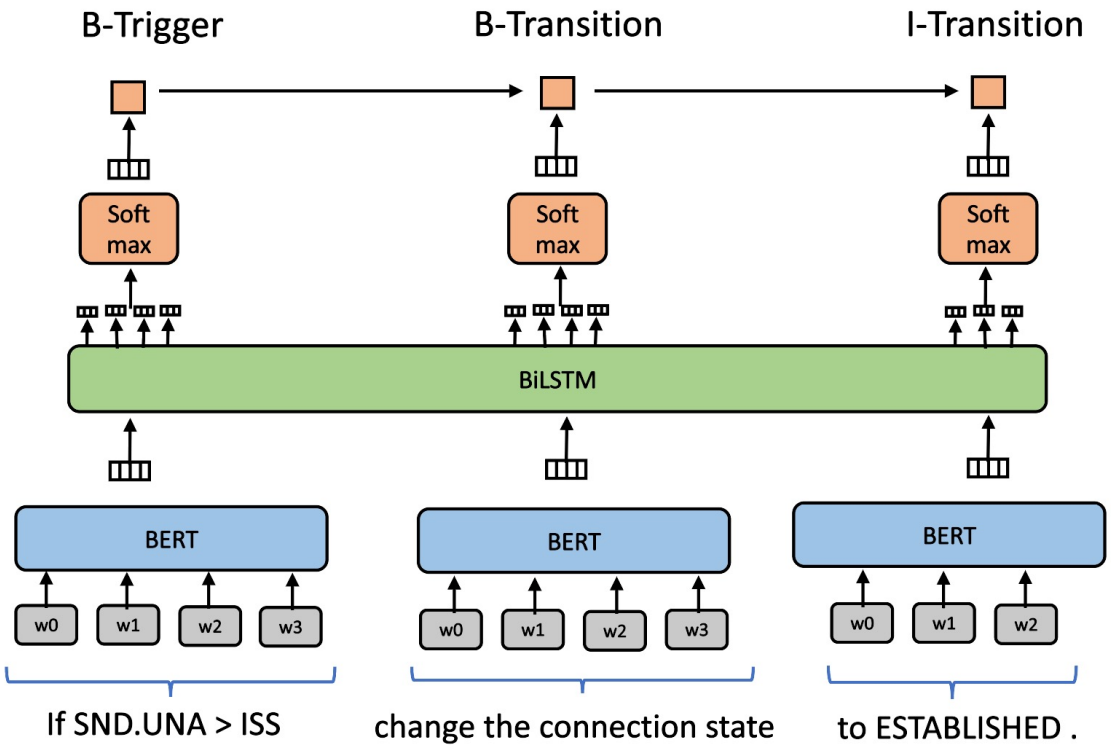
$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{t=1}^{T} \exp(f(y_t, y_{t-1}, \boldsymbol{x_t}; \boldsymbol{\theta}))$$

f is a linear scoring function learned with parameter vector θ over a feature vector x_t

- We use the LINEARCRF provided by the pystruct library

Bidirectional LSTM CRF layer (NEURALCRF)

- LSTMs are recurrent neural networks, a class of neural network that can learn long-term dependencies
- Architecture:
  - BERT encoder is used to create chunk-level representations from word sequences
  - Resulting sequence of chunks is then processed using a Bidirectional LSTM (BiLSTM)
  - A softmax activation is used to obtain scores for the labels
  - CRF on top to leverage the sequential dependencies also in the output space



Features (all are standard in NLP pipelines)

- **Vocabulary***:* Bag-of-word features for all stemmed forms of the words in the training data
- **Capitalization Patterns:** all letters are in lower case, all letters are capitalized, the first letter is capitalized, the word is in camel case, the word has only symbols, the word has only numeric characters, or the word has any other form of alpha-numeric capitalization
- **Logical and Mathematical Expression Patterns***:* Different patterns corresponding to logical and mathematical expressions
- **Dictionary Features***:* Indicator features for a held-out dictionary of reserved state and variable names
- **Part-of-Speech Tags***:* Part-of-speech (POS) tags for all observed words (e.g. noun, verb, adjective). We use an off-the-shelf tagger
- **Position Features***: P*osition and relative position for each word in a chunk
- How we use them:
  - LINEARCRF: features vector used as the input x_t for each textual unit t
  - NEURALCRF: features vector concatenated to the resulting vector u_t from the BERT encoder, before being inputted to the BiLSTM layer

From segmentation to grammar semantics

- Use exact lexical matching to identify explicit mentions to states and events in the predicted sequences using a dictionary built on the definition tags
- For triggers, transitions, actions, variables, and errors we use an off-the-shelf Semantic Role Labeler (SRL) to identify predicted actions as either send, receive, or issue, depending on the verb used, as well as to identify the segment in the text being sent, received, or issued
  - Example: given a sentence like “Send a SYN segment”, an SRL model would identify the verb “to send” as the predicate, and “SYN segment” as the argument
- (We experiment with a simple set of *rules* to correct some easy cases that the prediction models fail to identify)
  - We refer to these models as LINEARCRF+R and NEURALCRF+R

FSM extraction

```
<control relevant="true">
<trigger><def_state id="3">REQUEST</def_state></trigger>
  <transition>A client socket enters this state, from <arg_source><ref_state
id="1">CLOSED</ref_state></arg_source>,</transition> after <action type=send>sending a
  <arg><ref_event type="send" id="1">DCCP-Request</ref_event> packet</arg> to try to
initiate a connection.</action>
</control>
```

- We need a procedure to extract an FSM from the intermediary representation
- Contain pointers for *where to look* in the intermediary representation in order to guess the source and target states, and labels, for the FSM transitions.
  - Might describe no transitions at all
  - Might describe multiple transitions at once
  - Might describe only part of a transition

Evaluation of NLP task

- **Gold**: FSM obtained with the entire annotation; documents annotated using the grammar we defined;
- **LinearCRF**: FSM obtained with the linear model
- **NeuralCRF**: FSM obtained with the neural networks model
- **LinearCRF+R**: same but with some rules
- **NeuralCRF+R**: same but with some rules

- 6 protocols: BGP, DCCP, LTP, PPTP, SCTP, TCP

Protocol semantic extraction

Model	Token-level			Span-level	
	Acc	Weighted F1	Macro F1	Strict	Exact
Rule-based	31.08	25.94	29.37	41.58	41.78
BERT-base	58.93	56.72	51.33	60.77	84.18
BERT-technical	62.38	60.31	52.50	62.84	83.81
LINEARCRF	58.95	56.61	49.58	63.98	85.65
LINEARCRF+R	58.60	56.79	50.62	63.52	85.18
NEURALCRF	**64.42**	**64.18**	**54.95**	**68.81**	**86.83**
NEURALCRF+R	62.79	62.50	53.64	66.22	86.10

Protocol	LINEARCRF		NEURALCRF		# Control Statements
	Strict	Exact	Strict	Exact	
BGPv4	52.99	82.56	**57.34**	**86.86**	6
DCCP	69.74	92.73	**75.60**	**93.25**	150
LTP	67.25	**94.44**	**74.22**	94.41	65
PPTP	84.21	96.05	**87.34**	**98.73**	25
SCTP	52.21	65.49	**58.54**	**65.85**	19
TCP	57.46	**82.64**	**59.82**	81.90	31

FSM extraction: Transitions

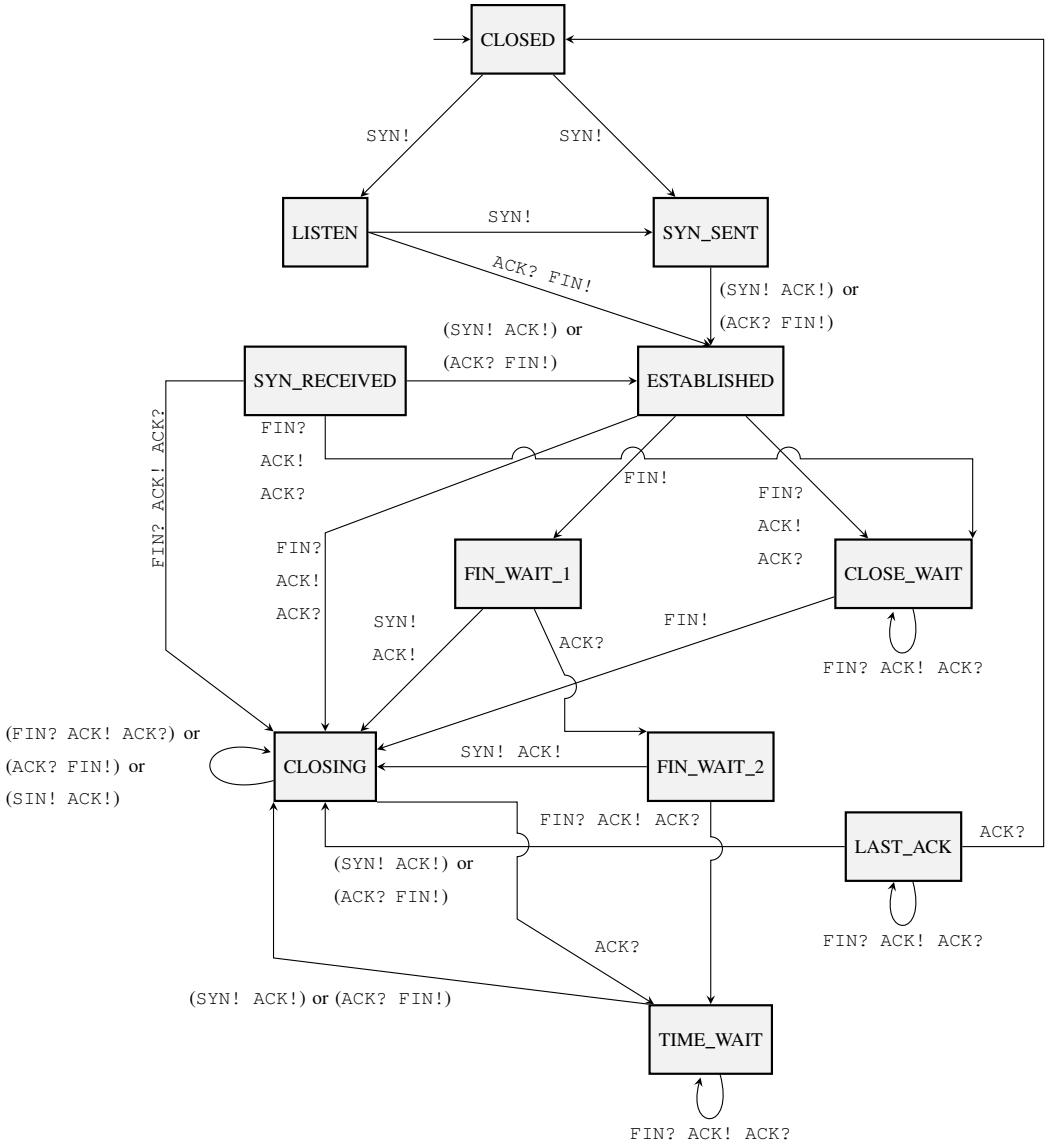
- We extract all states

TCP FSM	Canonical	Extracted	Correct	Partially Correct	Incorrect	Not Found
Gold	20	18	8	8	2	4
LINEARCRF		28	2	3	23	15
LINEARCRF+R		30	7	10	13	3
NEURALCRF		11	2	3	6	15
NEURALCRF+R		30	7	10	13	3
DCCP FSM	Canonical	Extracted	Correct	Partially Correct	Incorrect	Not Found
Gold	34	24	15	1	8	18
LINEARCRF		8	1	5	2	28
LINEARCRF+R		17	6	3	8	25
NEURALCRF		20	9	1	10	24
NEURALCRF+R		19	8	3	8	23

Transitions extraction errors

FSM	Transition	Error Type	Reason	Text Excerpt
Gold TCP	FIN_WAIT_1 $\xrightarrow{\text{FIN!}}$ LAST_ACK	Not Found	Target state not explicit	CLOSE-WAIT STATE: Since the remote side has already sent FIN, RECEIVEs must be satisfied by text already on hand, but not yet delivered to the user.
Gold DCCP	PARTOPEN $\xrightarrow{\text{DCCP-CLOSE?}}$ OPEN	Incorrect	Text is ambiguous	The client leaves the PARTOPEN state for OPEN when it receives a valid packet other than DCCP-Response, DCCP-Reset, or DCCP-Sync from the server.
LINEARCRF+R and NEURALCRF+R	SYN_SENT $\xrightarrow{\text{SYN!ACK!}}$ SYN_RECEIVED	Partially Recovered (expected SYN?ACK!)	Receive action is not explicit	If the state is SYN-SENT then enter SYN-RECEIVED, form a SYN,ACK segment and send it.

TCP FSM extracted by our models



This talk

Can we automatically extract formal specifications of protocols (FSM) from RFC?

Can we synthesize attacks based on the extracted FSM?

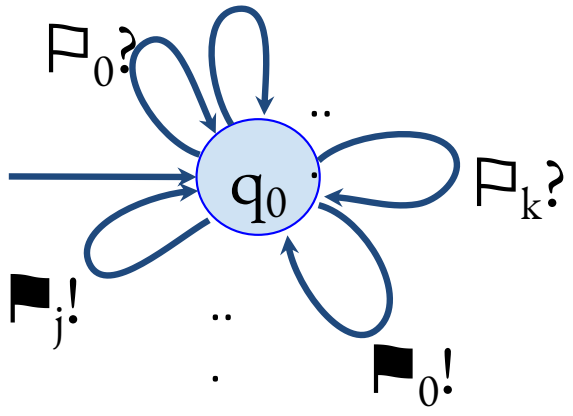
Attack synthesis

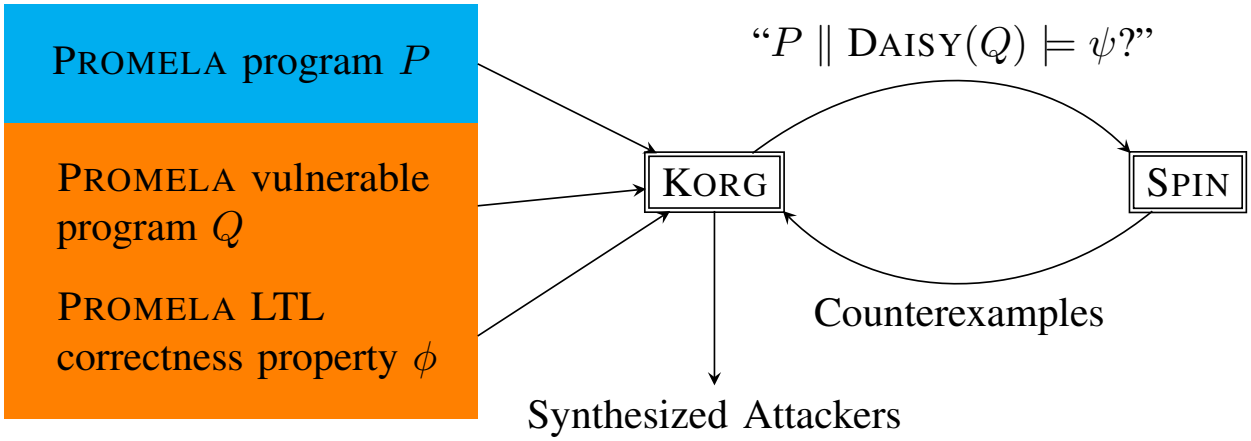
- Look at attacks through a formal methods lens
- Given a program with a specification, an attack is a counterexample violating some (security) property
- Approach
  - Define an attacker as a process that when composed with target system, results in a protocol property violation
  - Look for counterexamples on the composed system

What is the attacker program?

KORG

- Model the attacker as a Daisy process that nondeterministically exhausts the space of input and output events of a vulnerable process





Automated Attacker Synthesis for Distributed Protocols. Max von Hippel, Cole Vick, Stavros Tripakis, Cristina Nita-Rotaru, March 2020. SafeComp 2020. github.com/maxvinhippel/attackerSynthesis

Using RFC-extracted FSMs

- Apply KORG to two protocols DCCP and TCP
- Several state machines:
  - Canonical: FSMs from books, manually derived by experts
  - Gold: obtained with the grammar annotation
  - LINEARCRF+R: predicted by the linear model
  - NEURALCRF+R: predicted by the neural network model
- Select several properties by reading the RFC, hand-write them in LTL, and use them to find violations on each of the corresponding extracted FSMs
- Once we obtain a violation on an extracted model, we call them **candidate** attacks, we need to check them against the canonical FSM to see if they are **confirmed** or not

Protocol properties we used

- TCP:

$\phi 1$ =“No half-open connections.”

$\phi 2$ =“Passive/active establishment eventually succeeds.”

ϕ =“Peers don’t get stuck.”

$\phi 4$ =“SYN_RECEIVED is eventually followed by ESTABLISHED, FIN_WAIT_1, or CLOSED.”

- DCCP:

$\theta 1$ =“The peers don’t both loop into being stuck or infinitely looping.”

$\theta 2$ =“The peers are never both in TIME_WAIT.”

$\theta 3$ =“The first peer doesn’t loop into being stuck or infinitely looping.”

$\theta 4$ =“The peers are never both in CLOSE_REQ.”

Evaluation: Properties supported

- First we check what properties are supported by the extracted models

TCP PROMELA program	$\models \phi_1$	$\models \phi_2$	$\models \phi_3$	$\models \phi_4$
Canonical	✓	✓	✓	✓
Gold	✓	✓	✗	✗
LINEARCRF+R	✓	✓	✗	✓
NEURALCRF+R	✓	✓	✗	✓
DCCP PROMELA program	$\models \theta_1$	$\models \theta_2$	$\models \theta_3$	$\models \theta_4$
Canonical	✓	✓	✓	✓
Gold	✓	✓	✓	✓
LINEARCRF+R	✓	✓	✓	✓
NEURALCRF+R	✓	✓	✓	✓

Why weren't these properties supported

- TCP Gold does not support two properties
  - Why: Remember that Gold is not finding all 20 states of the canonical, due to ambiguities and not explicit mentioning of both states of the transition
- TCP Gold P program does not support property $\phi 4$, while the TCP LINEARCRF+R and NEURAL-CRF+R PROMELA programs do
  - Why: The same erroneous transition manifests in all three TCP programs, but in the TCP Gold program the code is reachable, while in the TCP LINEARCRF+R and NEURALCRF+R programs it is unreachable

Evaluation: Candidate attacks

	Candidates Guided by φ.				Unconfirmed Candidates Guided by φ.			
TCP PROMELA program	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_1	ϕ_2	ϕ_3	ϕ_4
Canonical	1	9	36	17	0	0	0	0
Gold	2	0	0	0	0	0	0	0
LINEARCRF+R	1	0	0	0	0	0	0	0
NEURALCRF+R	1	0	0	0	0	0	0	0
DCCP PROMELA program	θ_1	θ_2	θ_3	θ_4	θ_1	θ_2	θ_3	θ_4
Canonical	0	12	0	1	0	0	0	0
Gold	0	1	0	1	0	0	0	0
LINEARCRF+R	8	2	13	1	2	0	13	0
NEURALCRF+R	5	2	9	1	2	0	9	0

Examples

- *TCP.*NEURALCRF+R.ϕ1.*32* injects a single ACK to Peer 2, causing a desynchronization between the peers which can eventually cause a half-open connection, violating ϕ1

- *DCCP.*LINEARCRF+R.θ4.*32* injects and drops messages to and from each peer to first (unnecessarily) start and abort numerous connection routines, then guide both peers at once into CLOSE_REQ, violating θ4

Why we missed attacks for TCP

- Properties ϕ3 and ϕ4 not supported so we could not find violations against them
- Property ϕ2 is supported but did not not yield candidate attackers
  - ϕ2 says: “if the two peers infinitely often revisit the configuration where the first is in LISTEN while the second is in SYN_SENT, then eventually the first peer will reach ESTABLISHED”
  - TCP Gold, LINEARCRF+R, and NEURALCRF+R PROMELA programs, the tear-down routine is incomplete, so a connection cannot be fully closed
  - *Timeout* transitions needed to abort a connection establishment are missing

Limitations

- Our NLP models could not extract Canonical FSMs from RFCs, ambiguities and not explicit mentioning for Gold, difference between Gold and models due to errors in prediction
- Attacker synthesis with partial or incorrect FSM can result in missing attacks or unconfirmed attacks
- Attacks we found are dependent on property selection

MORE WORK TO BE DONE !!!!

Beyond attack synthesis

- Textual specifications will not go away, they are part of human interactions
- How to write better textual specifications, more amendable to automation
- The automation can be used as a grammar check for textual specification
- Check that design changes do not introduce new problems
- Check compatibility between modules
- Apply to other domains

Summary

- Show that it is possible to automate attacker synthesis against protocols by using textual specifications such as RFCs
- Show how to automatically extract FSMs from RFCs
- Apply the extracted FSMs to attack synthesis

Check out the code!
https://github.com/RFCNLP

ISEC Building

Relevant publications

- Leveraging State Information for Automated Attack Discovery in Transport Protocol Implementations. S. Jero, H. Lee, and C. Nita-Rotaru. IEEE/IFIP DSN 2015. **Best paper award.**
- How Secure and Quick is QUIC? Provable Security and Performance Analyses. R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru. IEEE Symposium on Security and Privacy 2015. **Awarded IETF/IRTF Applied Networking Research Prize, 2016.**
- Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach. Samuel Jero, Endadul Hoque, David Choffnes, Alan Mislove, Cristina Nita-Rotaru. NDSS 2018. **CISCO Network Security Distinguished Paper Award.**
- aBBRate: Automating BBR Attack Exploration Using a Model-Based Approach. Anthony Peterson, Samuel Jero, Endadul Hoque, Dave Choffnes, Cristina Nita-Rotaru. RAID 2020.
- Automated Attacker Synthesis for Distributed Protocols**.** Max von Hippel, Cole Vick, Stavros Tripakis, Cristina Nita-Rotaru, March 2020. SafeComp 2020.

- **Automated Attack Synthesis by Extracting Finite State Machines from Protocol Specification Documents** Maria Leonor Pacheco, Max von Hippel, Ben Weintraub Dan Goldwasser Cristina Nita-Rotaru. IEEE Security and Privacy, 2022.

https://github.com/RFCNLP