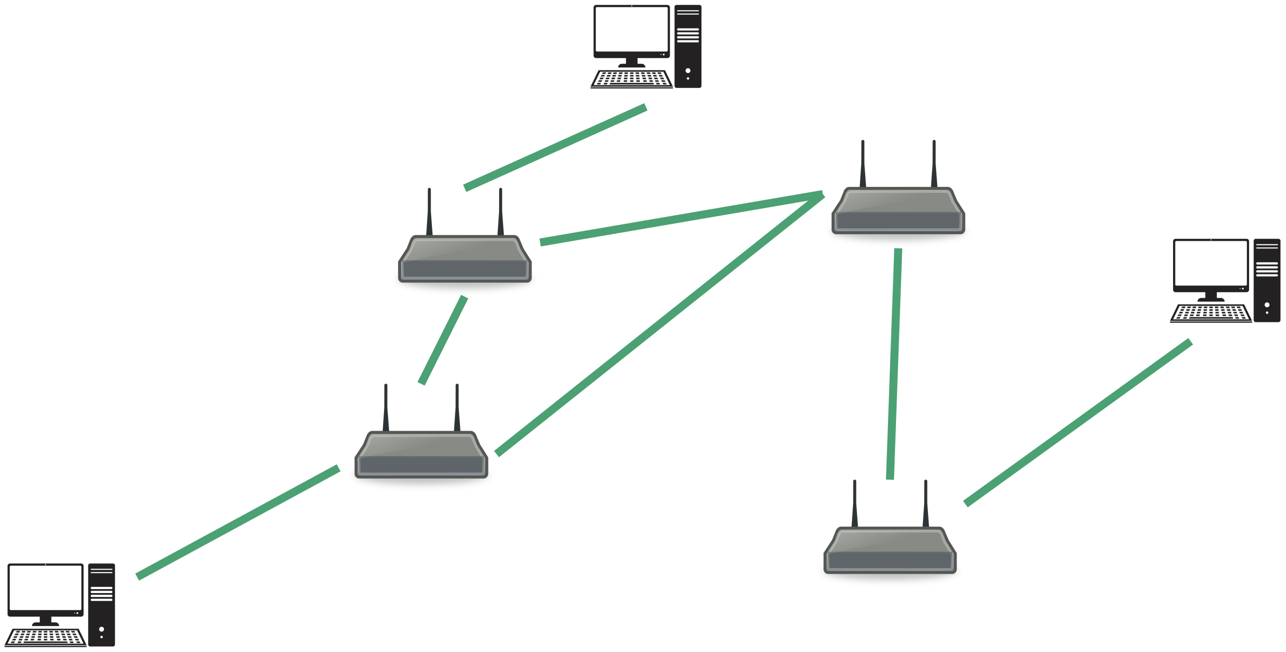
Specifying and Verifying a Real-World Packet Error-Correction System

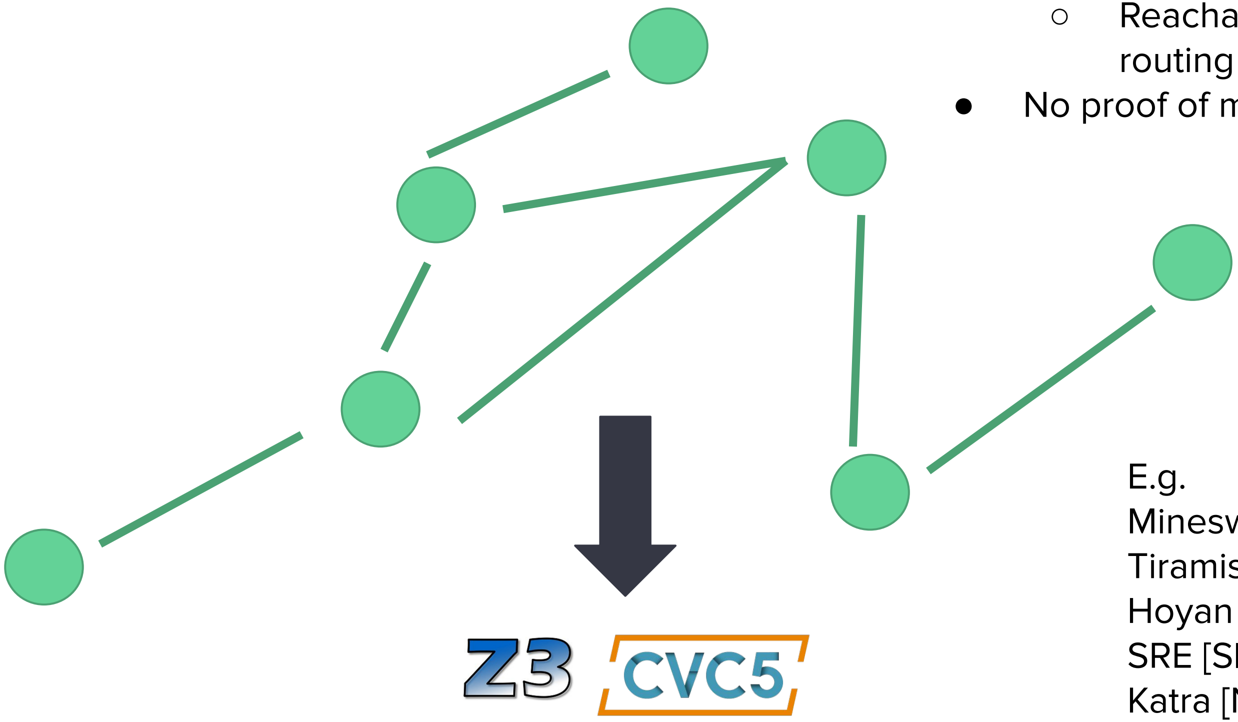
Josh Cohen
Princeton University
VSTTE 2023
10/23/23
With Andrew Appel

Network Verification



Network Verification

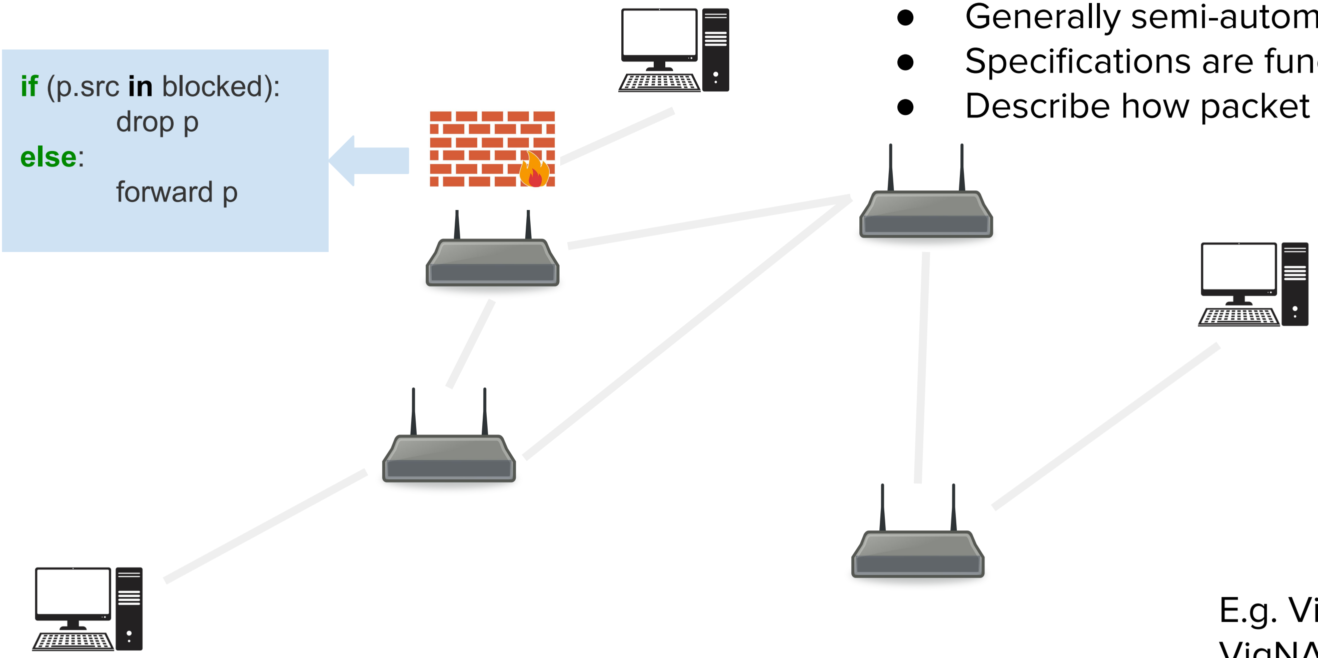
- Model network as simpler abstraction (e.g. graph)
- Use SMT solvers to prove network properties
  - Reachability, fault tolerance, routing convergence
- No proof of model ↔code



E.g.
Minesweeper [SIGCOMM 17],
Tiramisu [NSDI 20],
Hoyan [SIGCOMM 20],
SRE [SIGCOMM 22],
Katra [NSDI 22],
Flash [SIGCOMM 22]

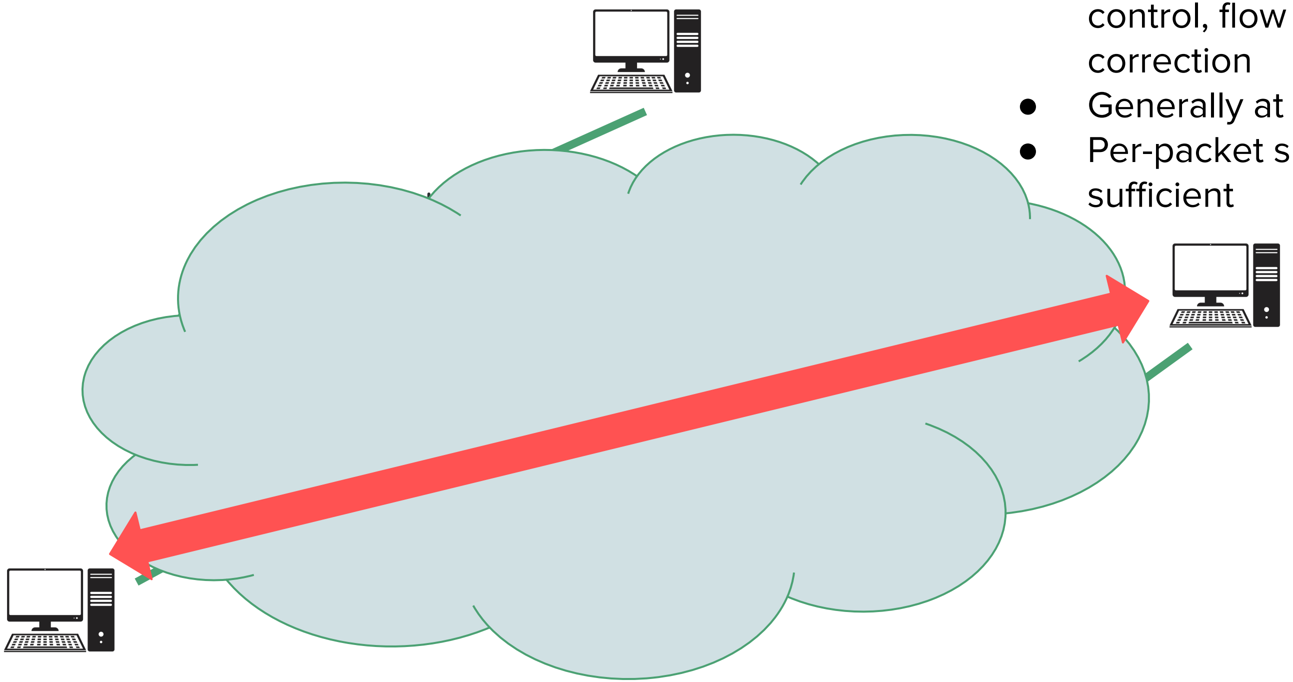
Network Verification

- Verify implementation of per-packet network functions
  - e.g. NAT, firewall, load balancer
- Generally semi-automated
- Specifications are functional programs
- Describe how packet headers change



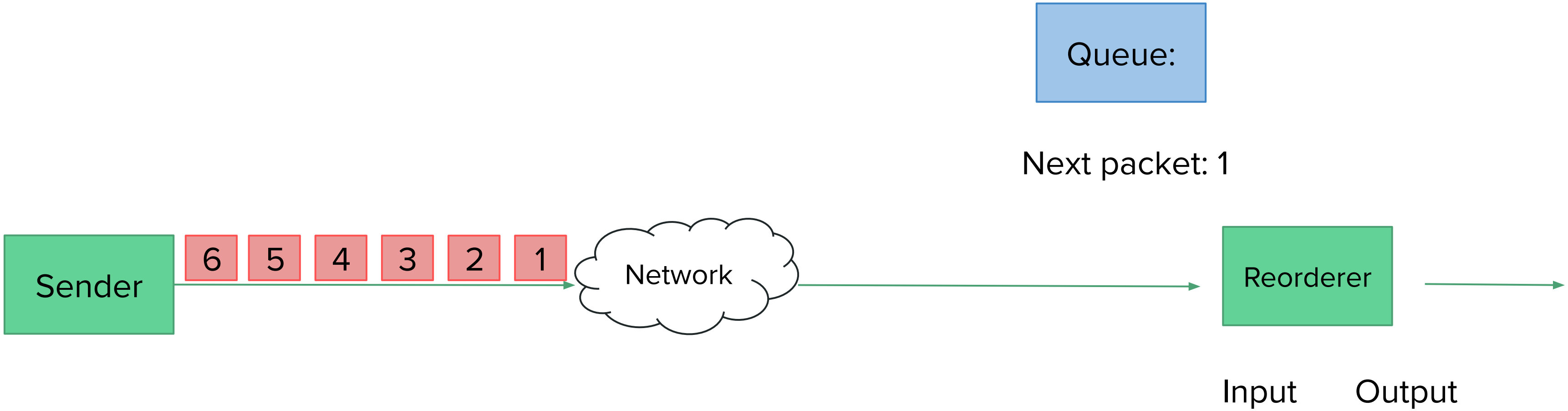
E.g. Vigor [SOSP 19],
VigNAT [SIGCOMM 17],
Klint [NSDI 22],
Gravel [NSDI 20],
Verifiable P4 [ITP 23]

Network Verification



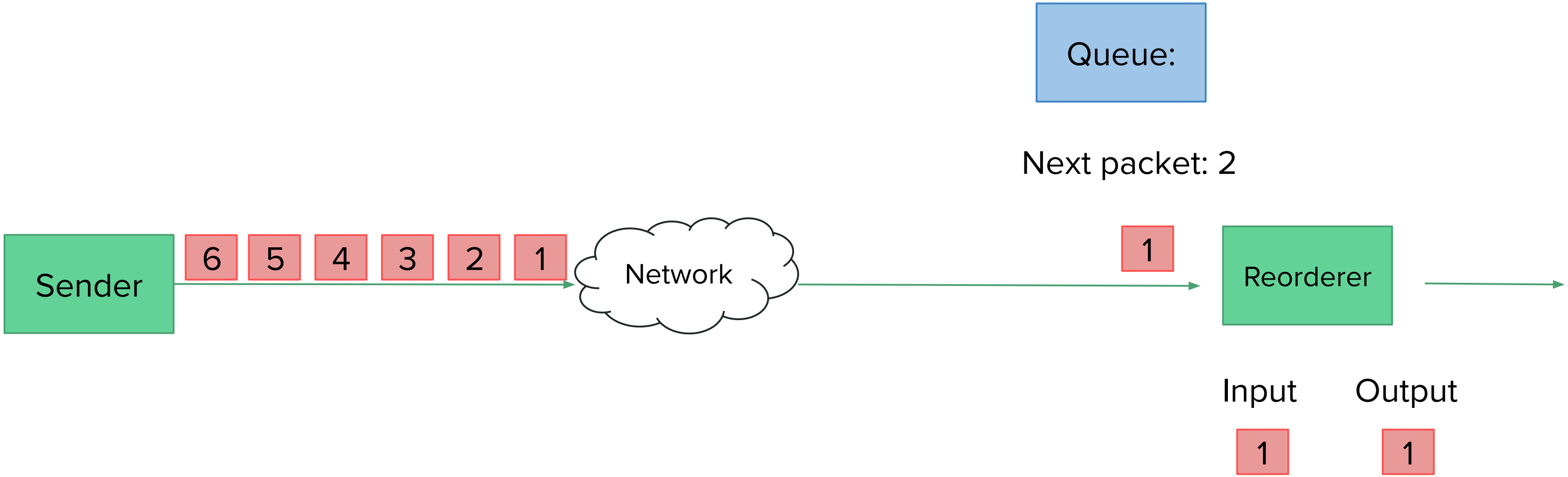
- What about end-to-end network functions?
- E.g. packet reordering, congestion control, flow control, error correction
- Generally at transport layer
- Per-packet specifications not sufficient

Example - Packet Reordering

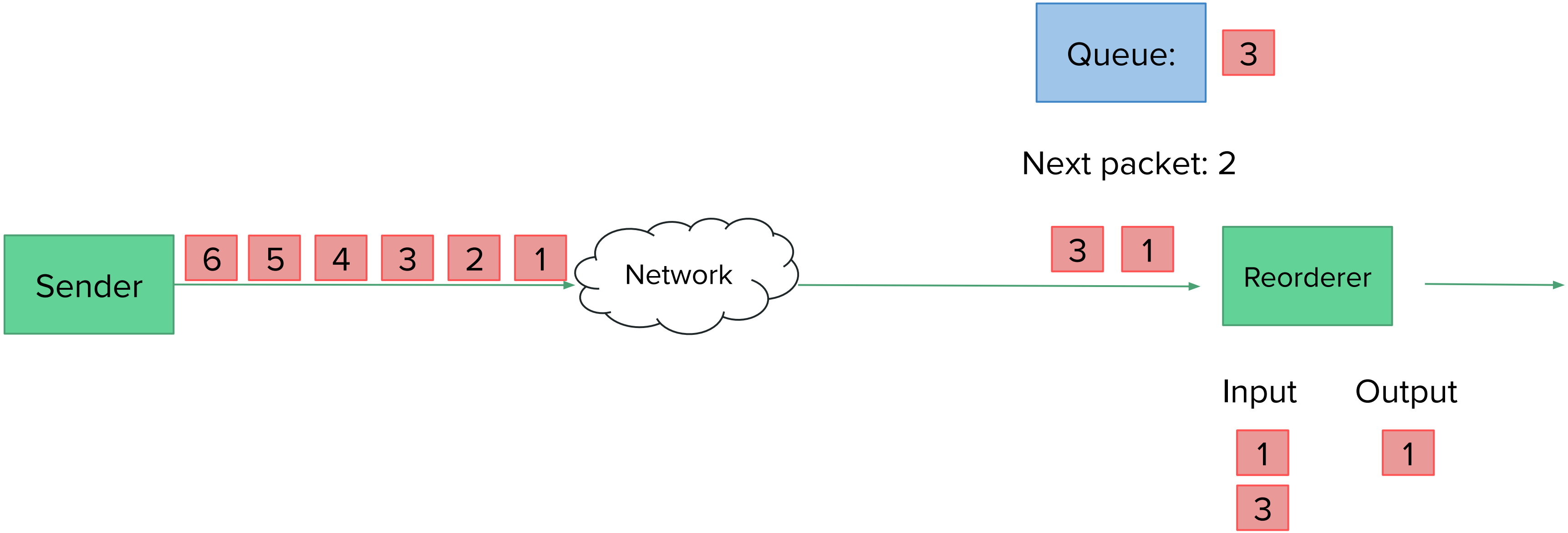


- Want illusion of in-order delivery
- Reorderer maintains queue, outputs next packets in order
- Also outputs packets if they have timed out

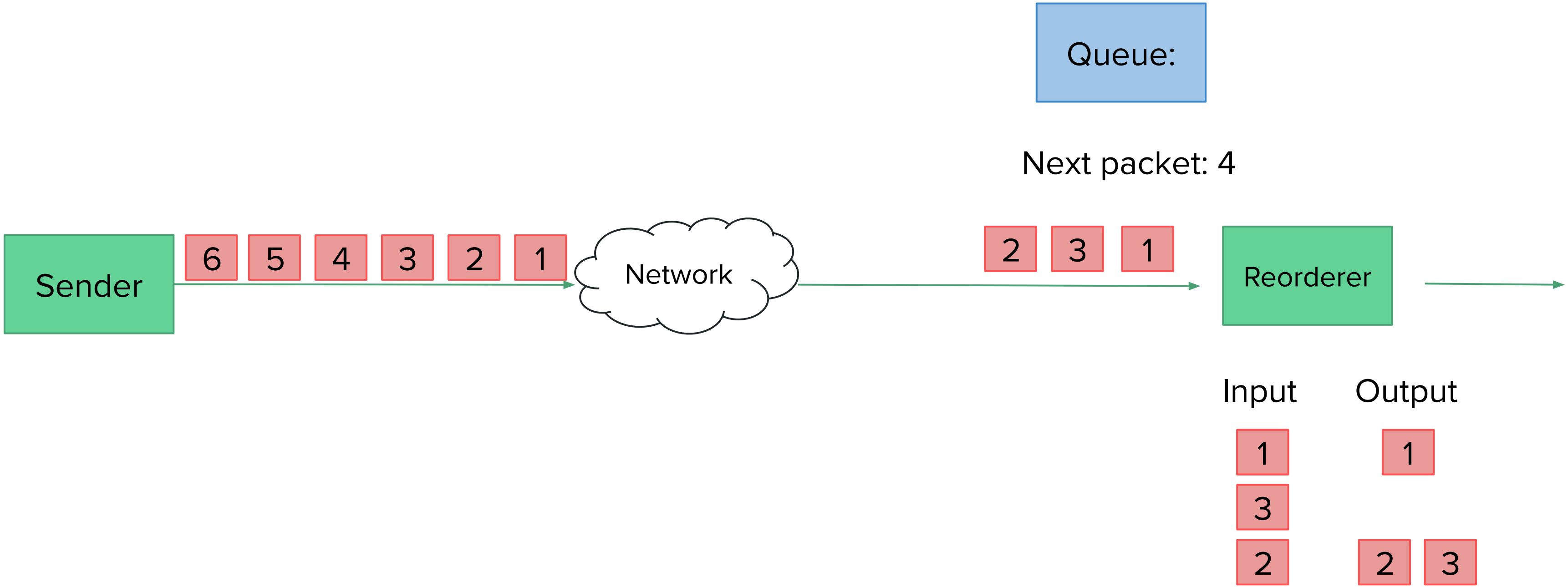
Example - Packet Reordering



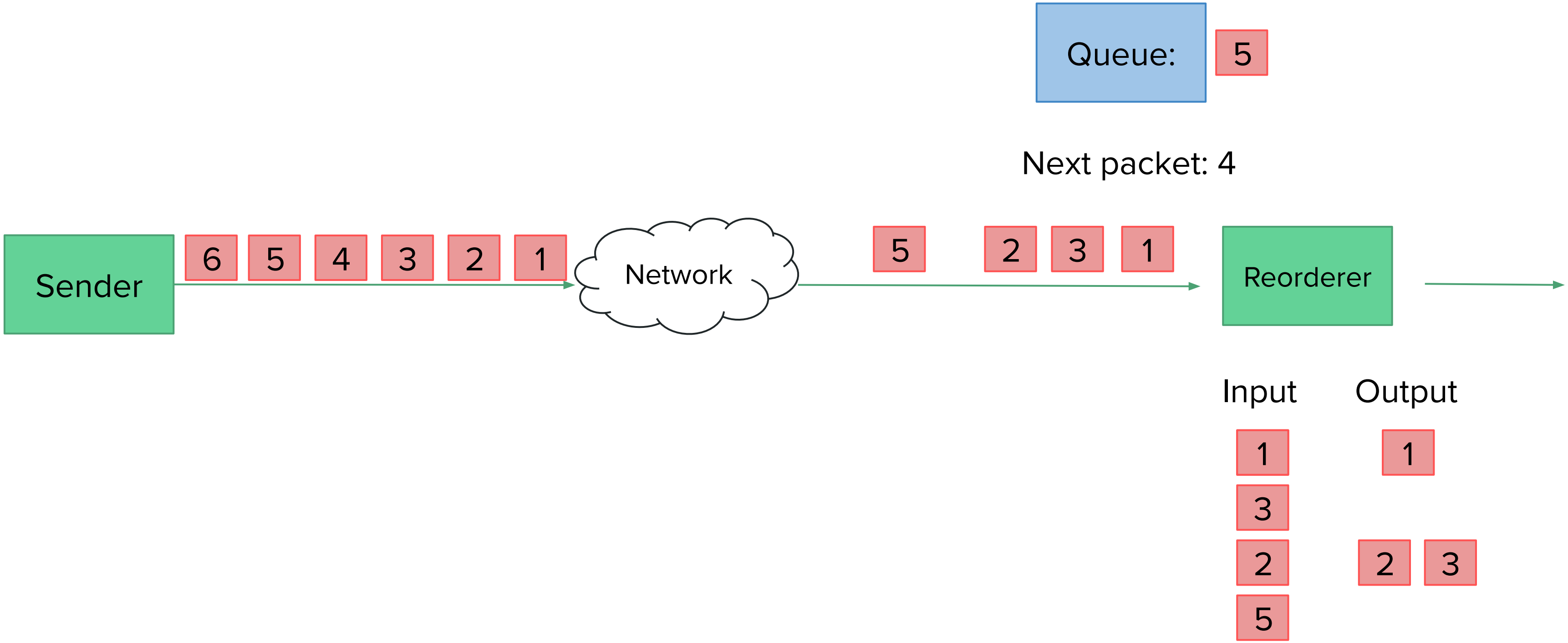
Example - Packet Reordering



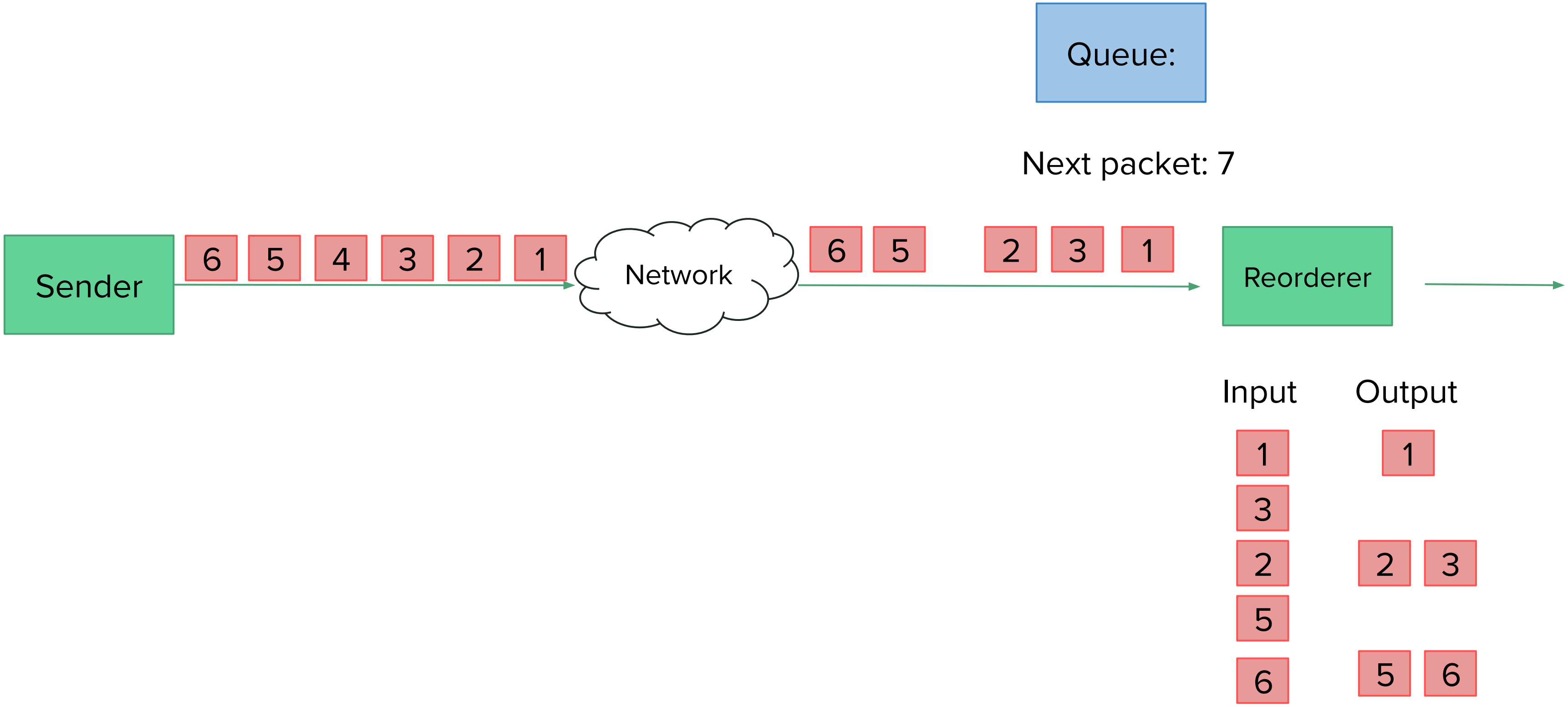
Example - Packet Reordering



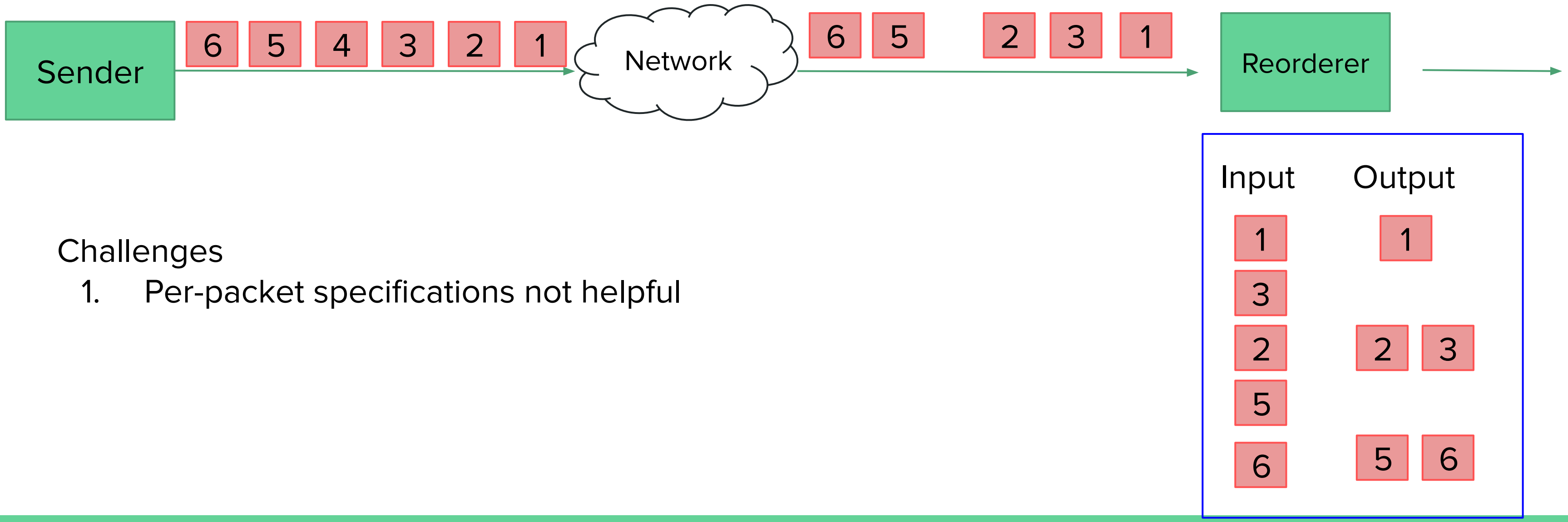
Example - Packet Reordering



Example - Packet Reordering



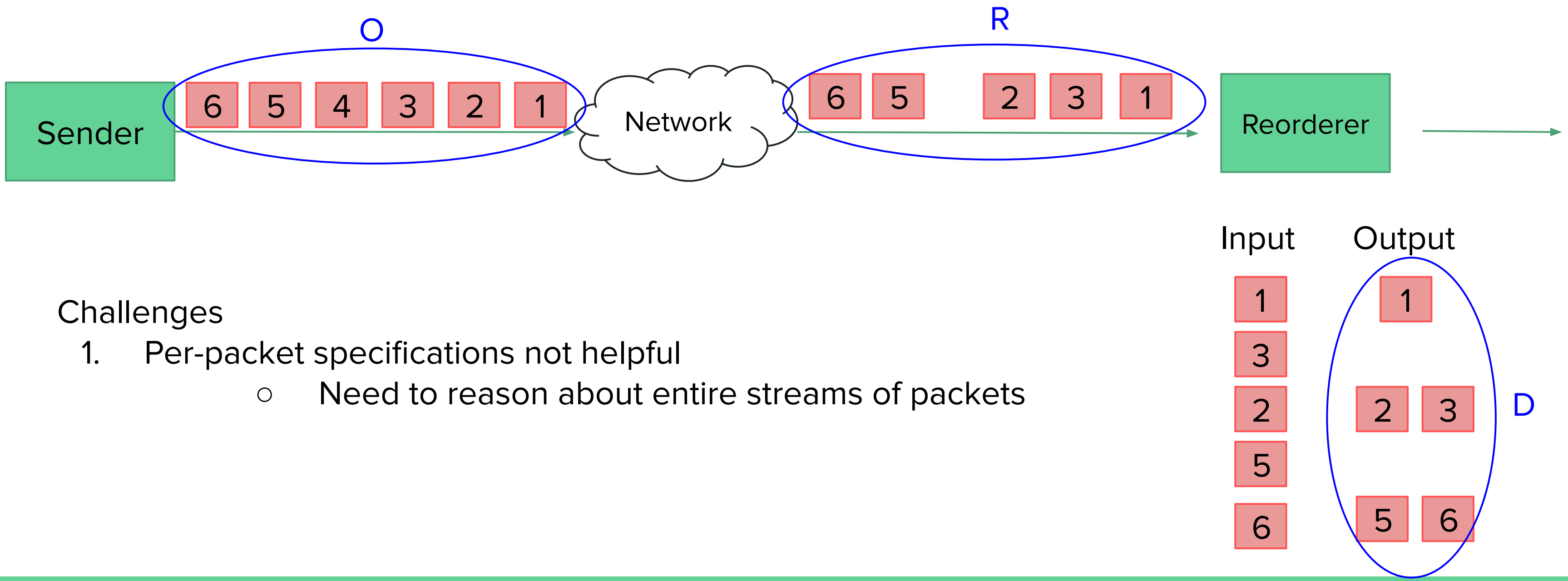
Example - Packet Reordering



Challenges

1. Per-packet specifications not helpful

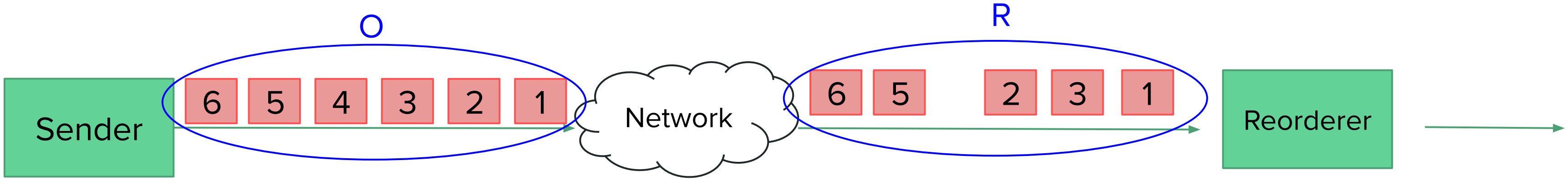
Example - Packet Reordering



Challenges

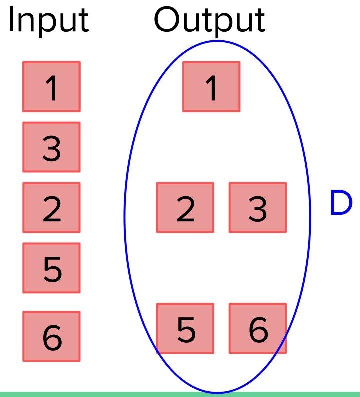
1. Per-packet specifications not helpful
    - Need to reason about entire streams of packets

Example - Packet Reordering



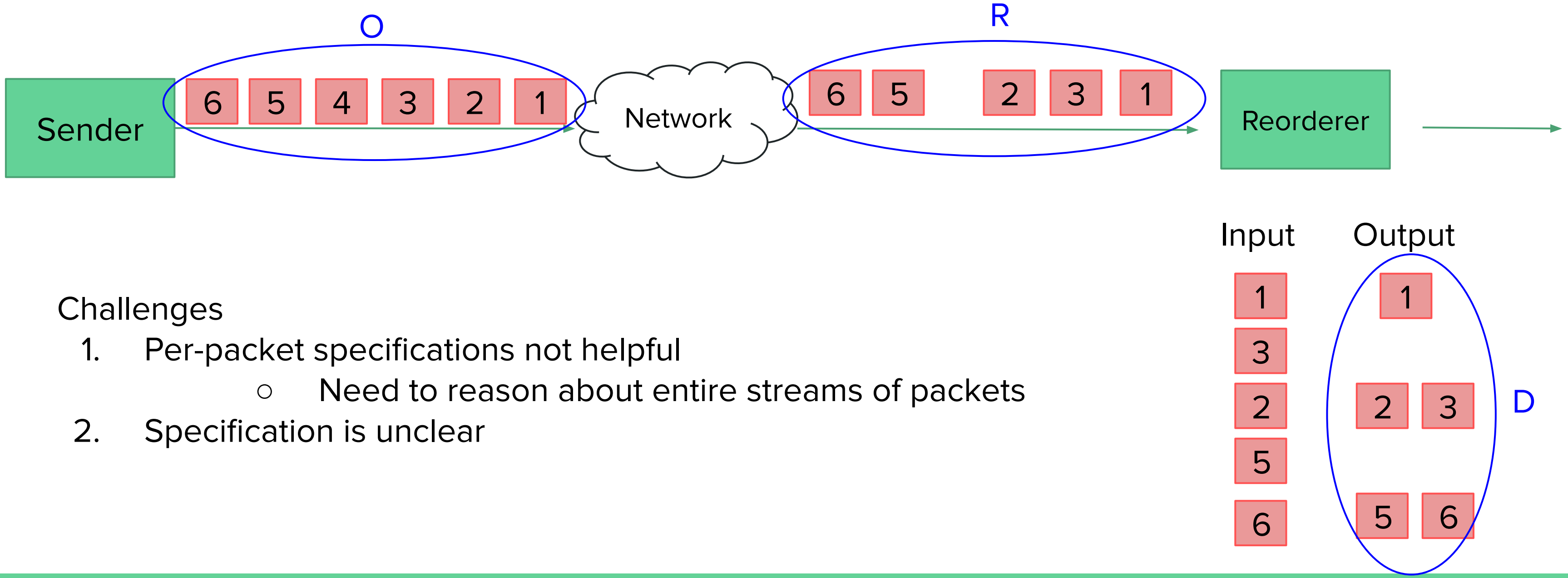
Challenges

1. Per-packet specifications not helpful
   - Need to reason about entire streams of packets
2. Specification is unclear



Example - Packet Reordering

Spec: Stream D is sorted
- Spec does not hold
- Spec very weak

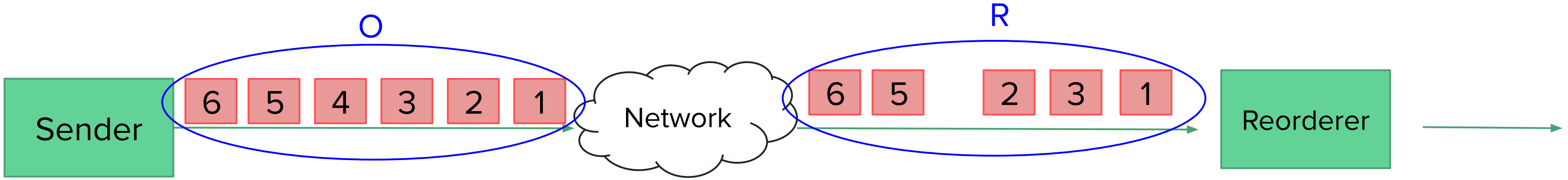


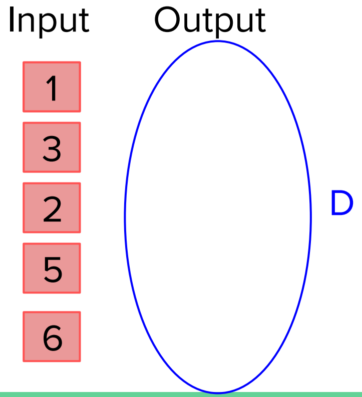
Challenges
1. Per-packet specifications not helpful
    - Need to reason about entire streams of packets
2. Specification is unclear

Example - Packet Reordering

Spec: Stream D is sorted

- Spec does not hold
- Spec very weak





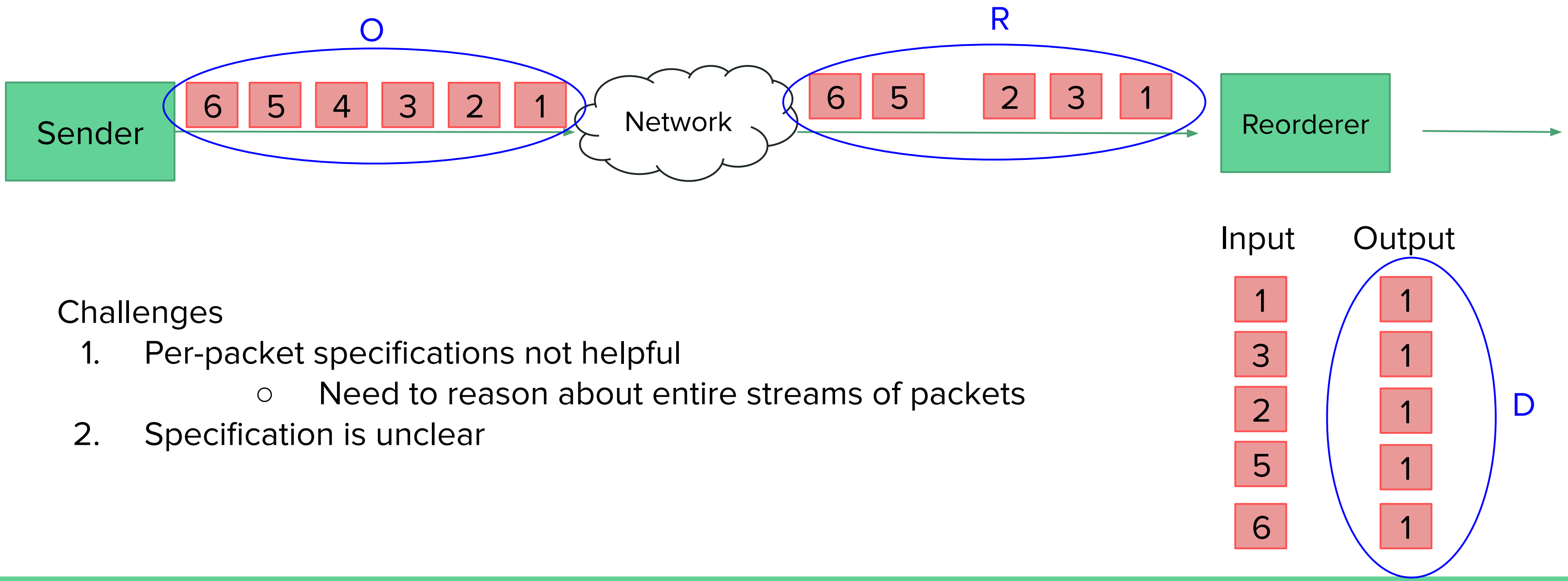
Challenges

1. Per-packet specifications not helpful
   - Need to reason about entire streams of packets
2. Specification is unclear

Example - Packet Reordering

Spec: Stream D is sorted

- Spec does not hold
- Spec very weak



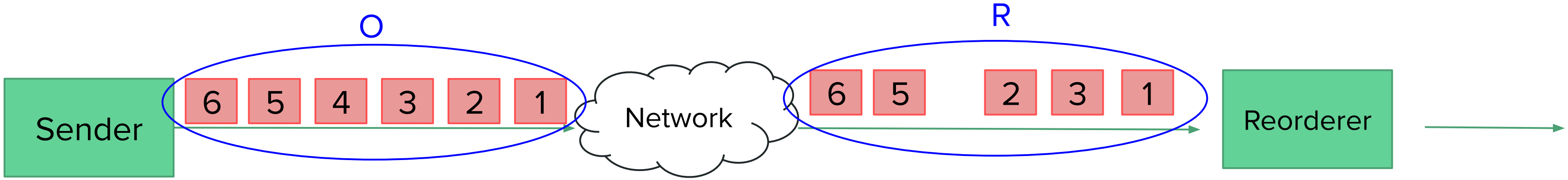
Challenges

1. Per-packet specifications not helpful
   - Need to reason about entire streams of packets
2. Specification is unclear

Example - Packet Reordering

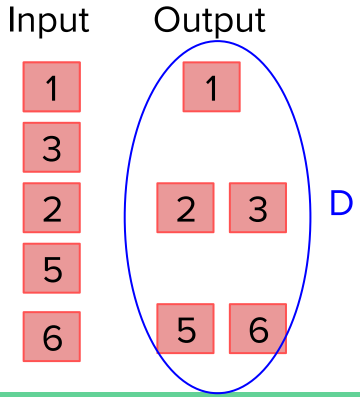
Spec: Stream D is O

- Stronger but does not hold due to network delay/loss



Challenges

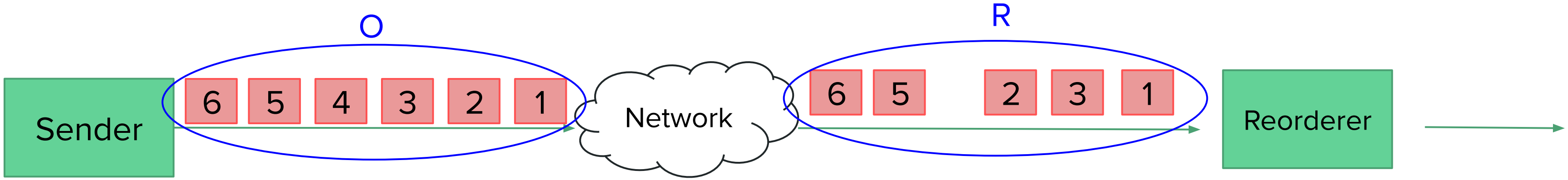
1. Per-packet specifications not helpful
    - Need to reason about entire streams of packets
2. Specification is unclear

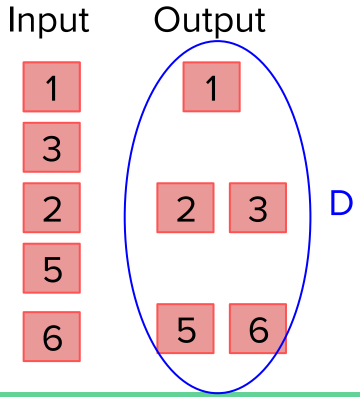


Example - Packet Reordering

Spec: Stream D is O

- Stronger but does not hold due to network delay/loss





Challenges

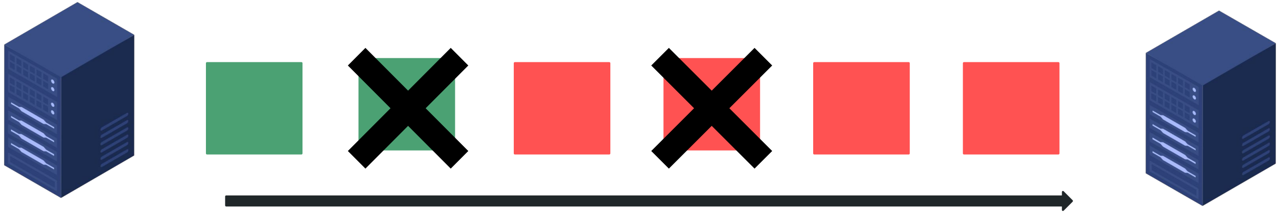
1. Per-packet specifications not helpful
    - Need to reason about entire streams of packets
2. Specification is unclear
3. Strong specification needs to reason about network conditions: reordering, duplication, delay, loss

Goals

- Develop a methodology for specifying and verifying these kinds of end-to-end network functions
- Extended case study - real world packet error-correction system
- All done using machine-checked proofs in Coq proof assistant

Forward Error Correction

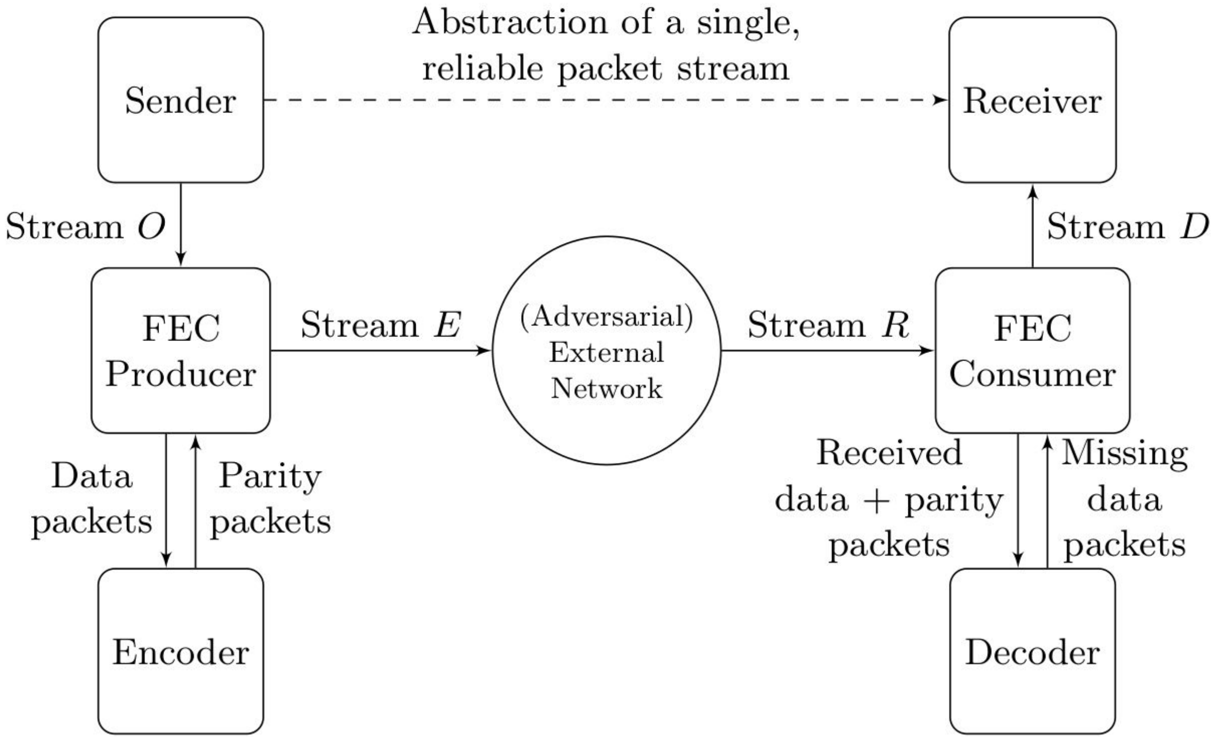
- Send data over network (or any noisy channel) - some may not arrive
- Usual solution, retransmit missing data
- In many cases, infeasible or impossible due to latency requirements, storage at sender
- Solution: use Error-Correcting code to create additional parity packets, enabling recovery of lost data



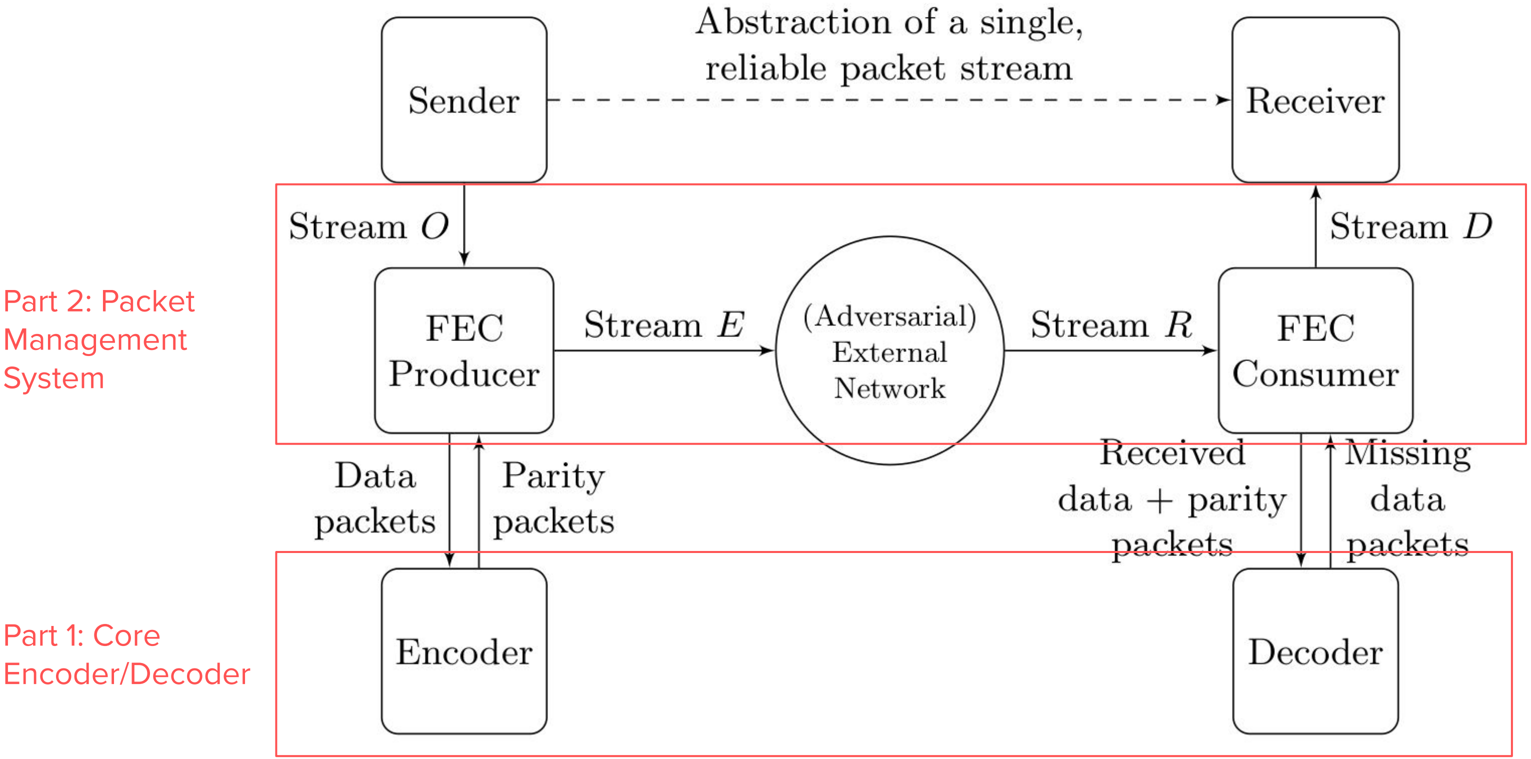
A Real-World FEC System

- C implementation originally written by Anthony McAuley of Bellcore in ‘90s, in active use since
- Algorithm is modified Reed-Solomon, developed by Rabin [Journal of the ACM 1989], McAuley [SIGCOMM 90], and others
  - Block code: *k* data packets + *h* parity packets, can correct if at most *h* total losses
- 2 parts: core encoder/decoder and larger packet/buffer management system
- Core encoder/decoder verified [CAV 2022], larger system more difficult to specify

FEC System Architecture

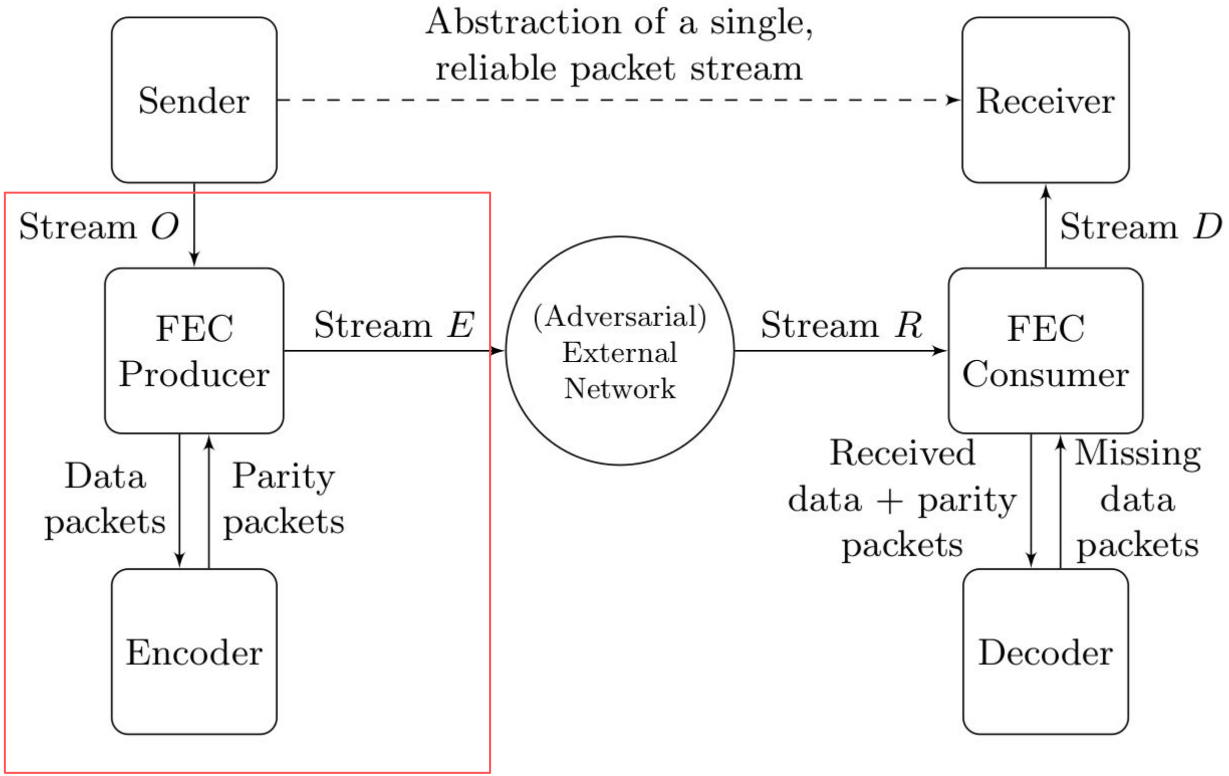


FEC System Architecture

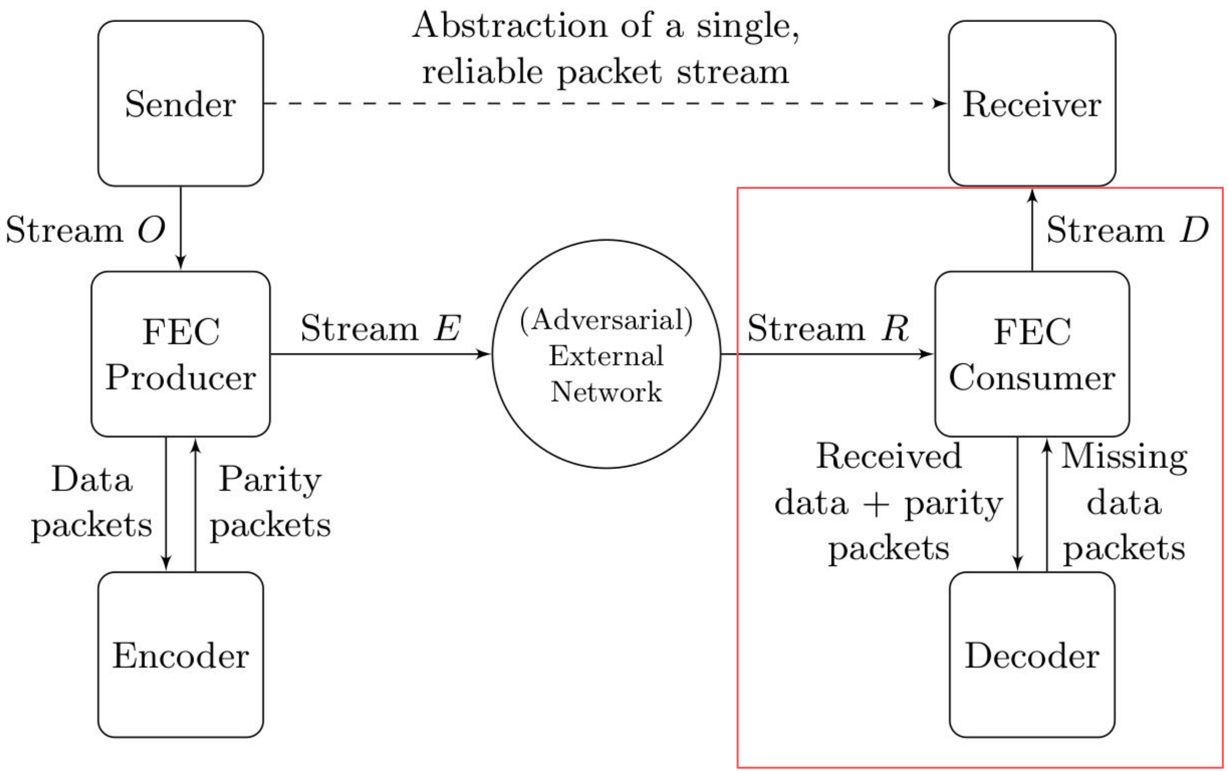


FEC System Architecture

Group packets into batches of *k*, generate *h* parities, send all as stream *E*

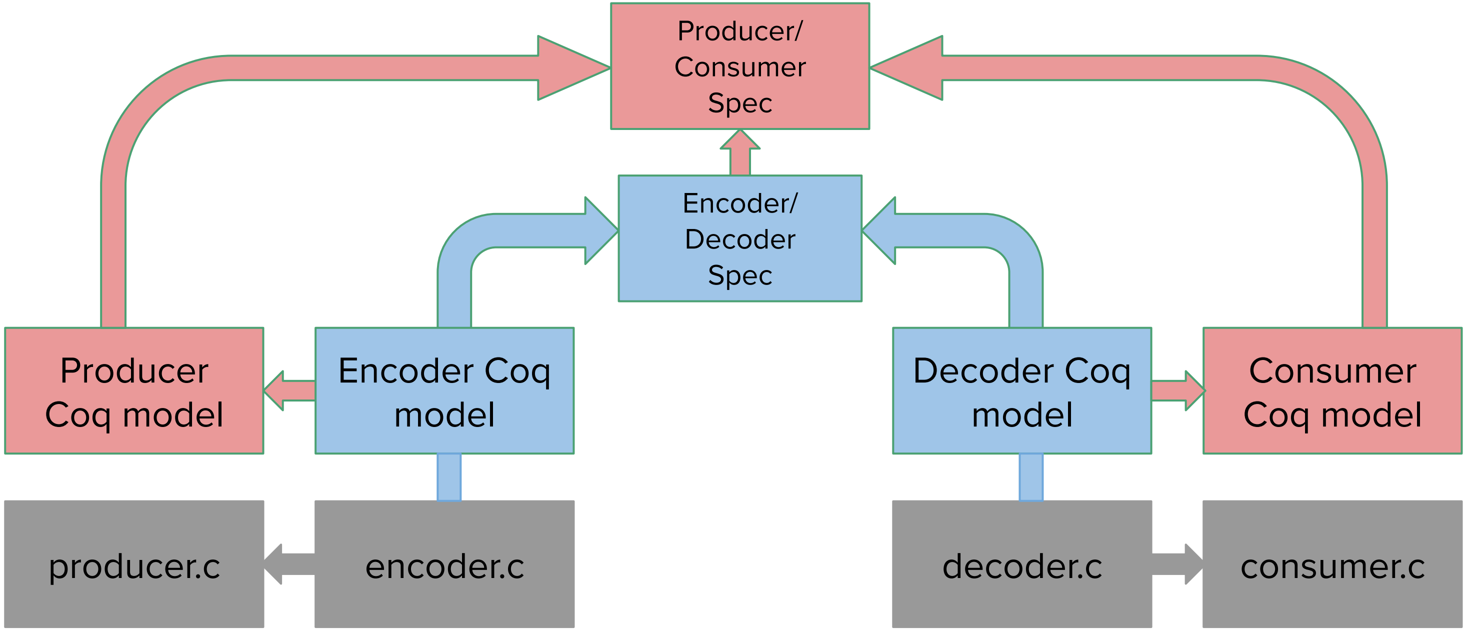


FEC System Architecture



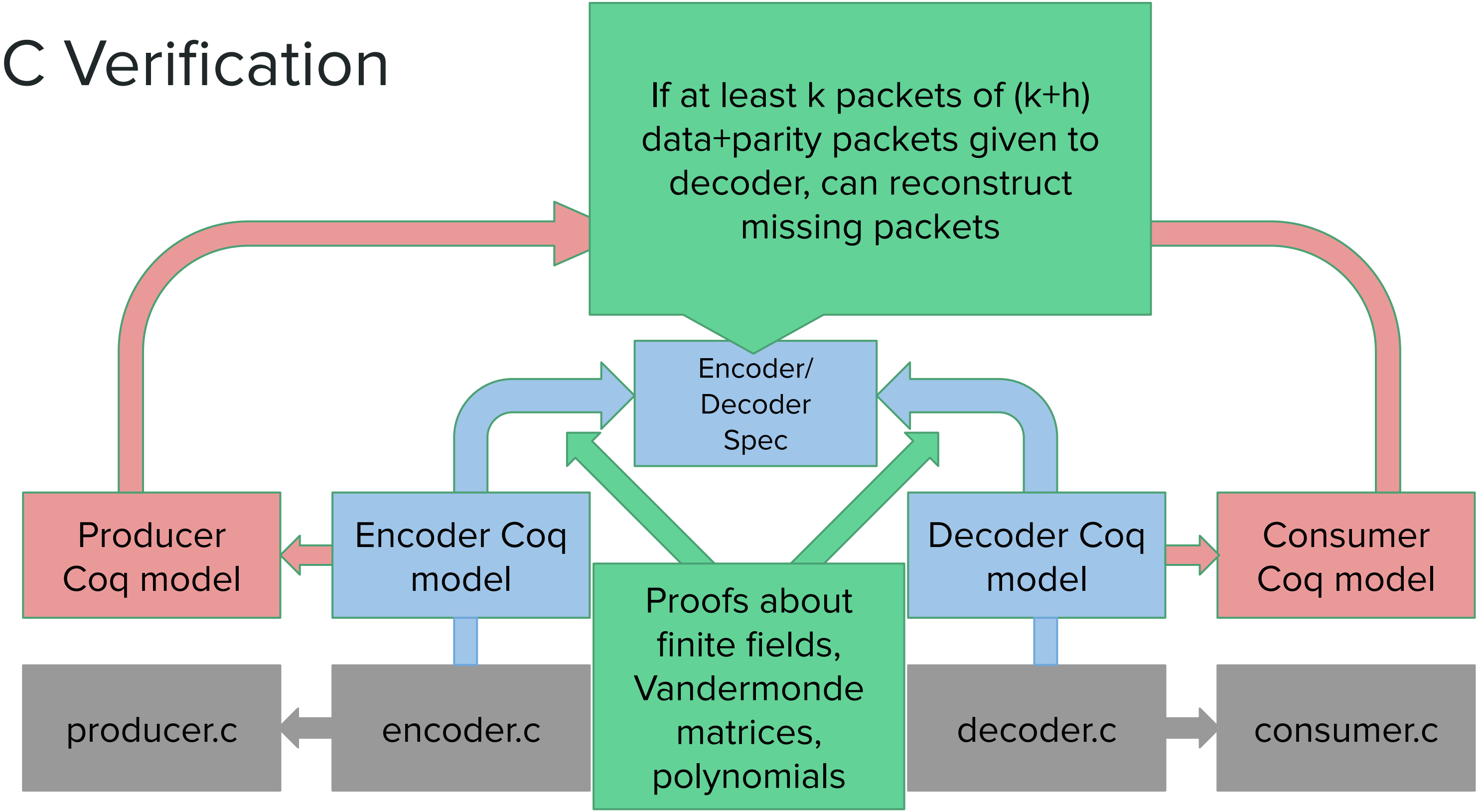
Store packets in assigned batches, when enough packets received in a batch, regenerate original packets

FEC Verification

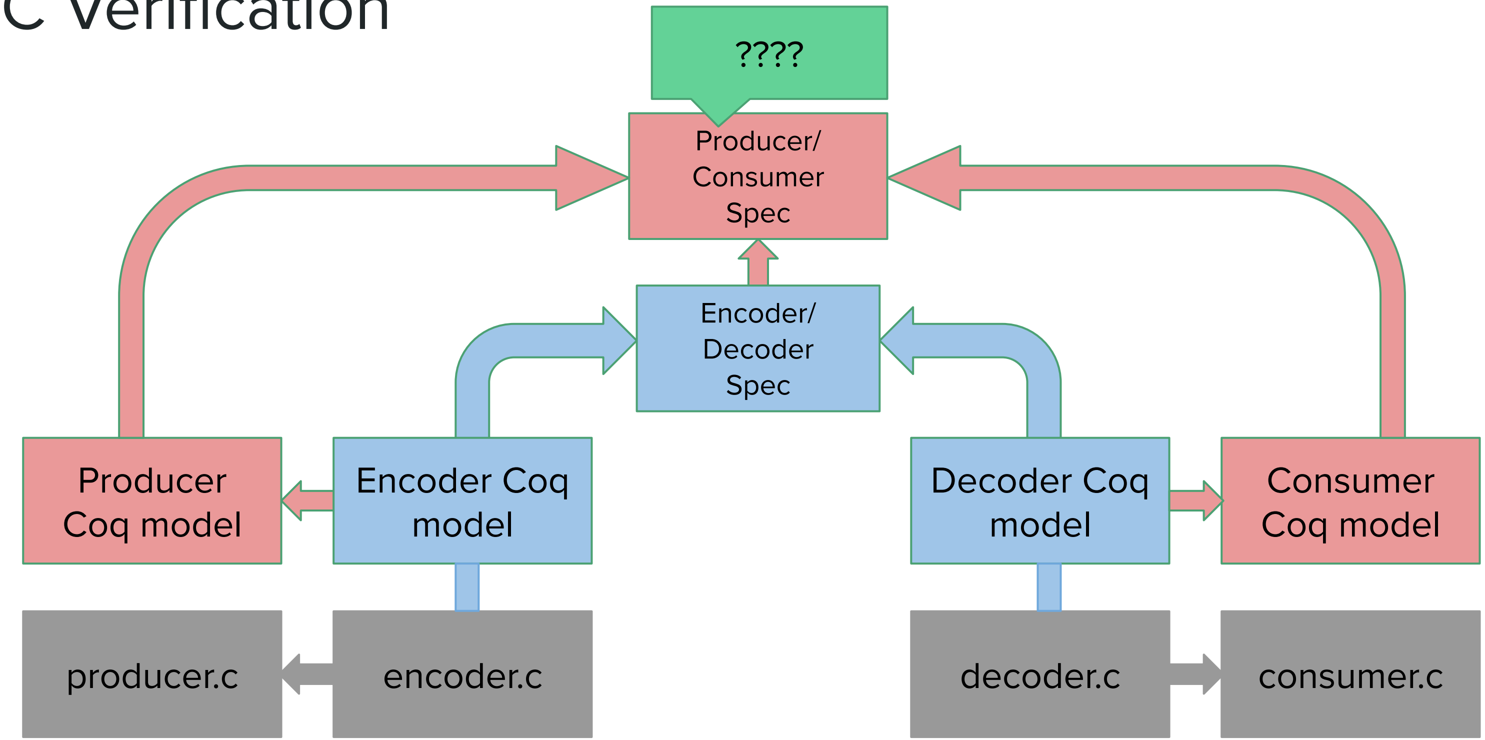


Cohen, J.M., Wang, Q., Appel, A.W. (2022). Verified Erasure Correction in Coq with MathComp and VST. In: Shoham, S., Vizel, Y. (eds) Computer Aided Verification. CAV 2022. Lecture Notes in Computer Science, vol 13372. Springer, Cham.

FEC Verification



FEC Verification



Surprises in the Implementation

> A program without a specification cannot be right or wrong, it can only be surprising.
> - Paraphrase of J. J. Horning, 1982

Current implementation satisfies no reasonable spec in 3 ways:

1. Memory leaks, implicit casting between signed and unsigned ints
2. Does not handle sequence number wraparound
3. Timeout mechanism causes unrelated packets to be dropped, can affect behavior of packets in other batches, some packets dropped unnecessarily
    - Violates *locality* - behavior should be per-batch

Surprises in the Implementation

> A program without a specification cannot be right or wrong, it can only be surprising.
> - Paraphrase of J. J. Horning, 1982

Current implementation satisfies no reasonable spec in 3 ways:

1. Memory leaks, implicit casting between signed and unsigned ints
2. Do ot handle sequence number wraparound
3. sm causes unrelated packets to be dropped, can affect ets in other batches, some packets dropped unnecessarily
   cality - behavior should be per-batch

Bugs under any possible specification

Surprises in the Implementation

> A program without a specification cannot be right or wrong, it can only be surprising.
> - Paraphrase of J. J. Horning, 1982

Current implementation satisfies no reasonable spec in 3 ways:

1. Memory leaks, implicit casting between signed and unsigned ints
2. Does not handle sequence number wraparound
3. Tim t mechanism causes unrelated packets to be dropped, can affect b ts in other batches, some packets dropped unnecessarily
    - *ality* - behavior should be per-batch

Bugs depending on the program's environment (i.e. how many packets are expected)

Surprises in the Implementation

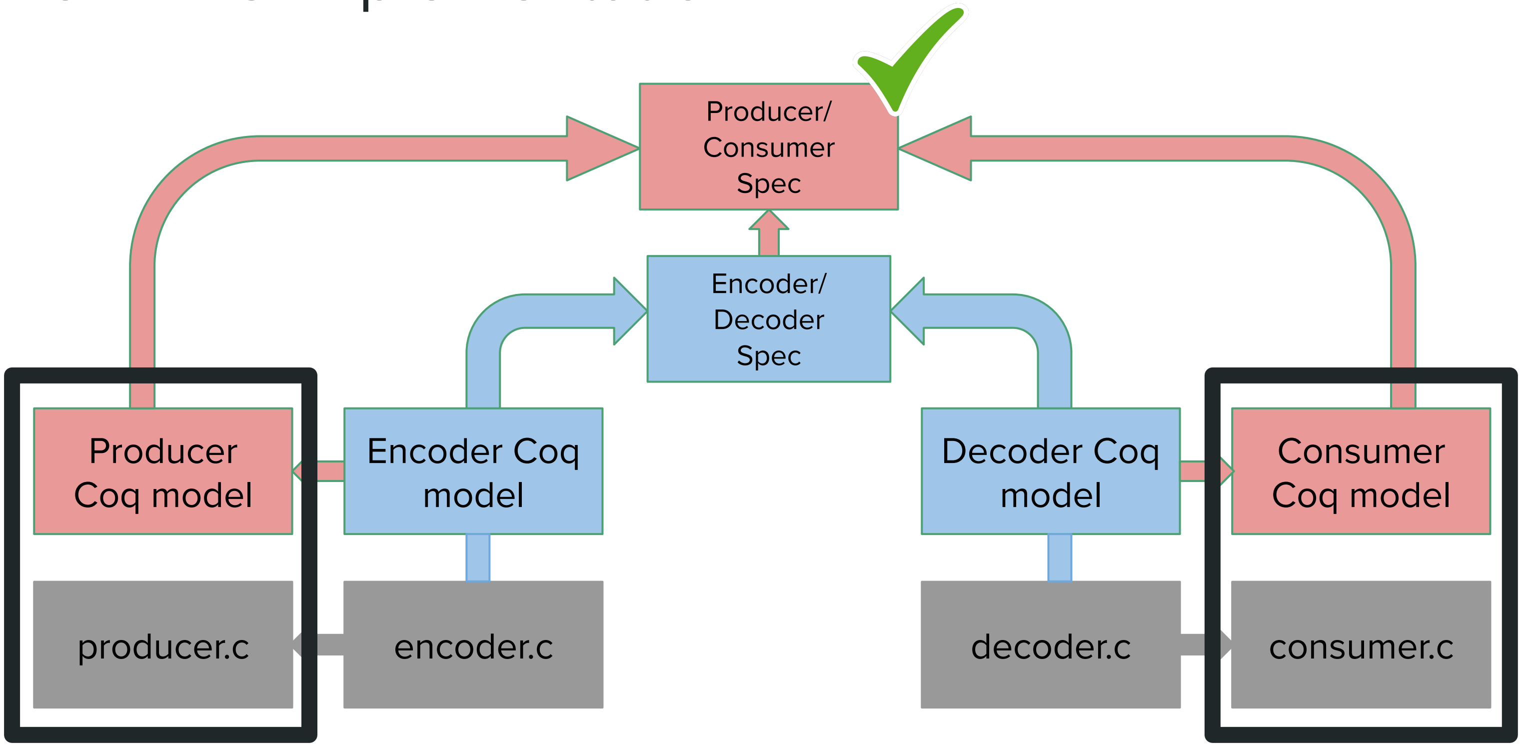
> A program without a specification cannot be right or wrong, it can only be surprising.
> - Paraphrase of J. J. Horning, 1982

Bugs if the program is expected to guarantee packet recovery

Curr… on satisfies no reasonable spec in 3 ways:

1. …plicit casting between signed and unsigned ints
2. Do… not handle sequence number wraparound
3. Timeout mechanism causes unrelated packets to be dropped, can affect behavior of packets in other batches, some packets dropped unnecessarily
    - Violates *locality* - behavior should be per-batch

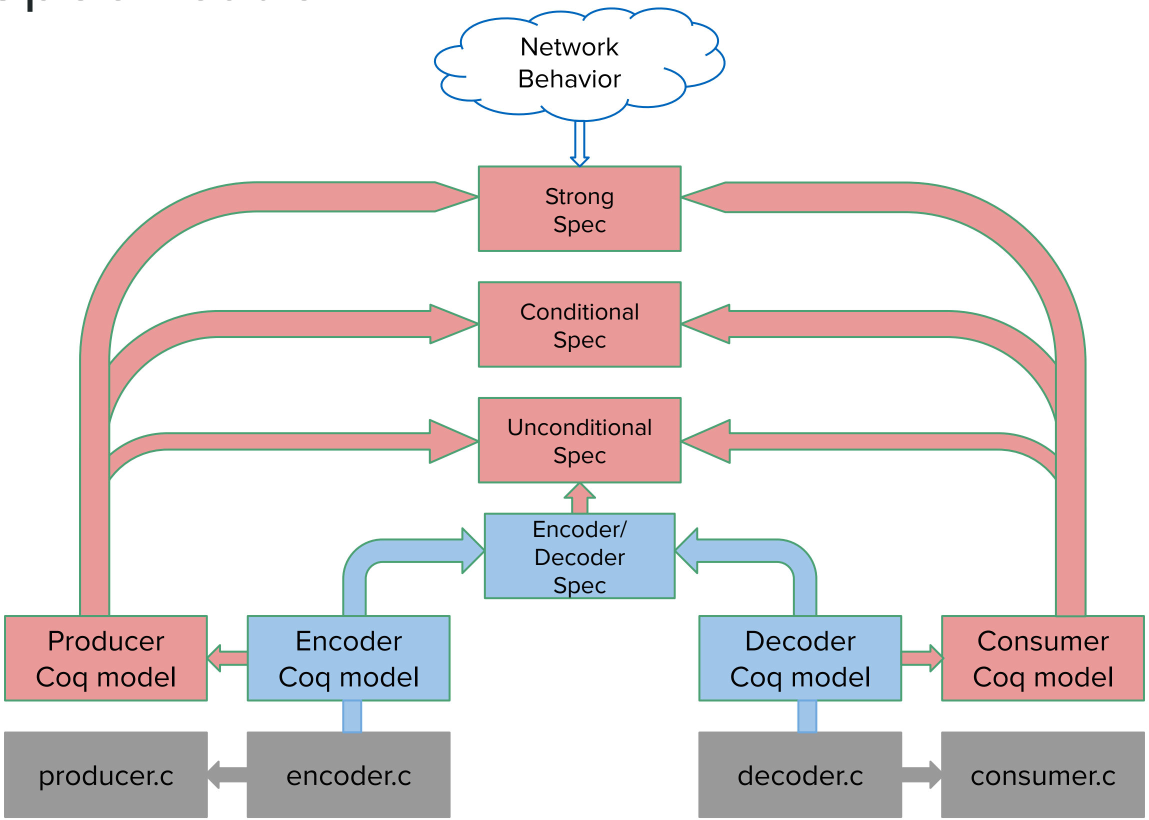
A New FEC Implementation



Layers of Specification

- Spec relies on external network conditions (reordering, duplication, delay, loss)
- One spec is not enough!
- Different behavior/guarantees based on external network conditions
- Want to know: guarantees in good/normal conditions as well as (weaker) guarantees in bad/adversarial ones

Layers of Specification



Layers of Specification

Unconditional specification:

1. The program has no memory leaks, signed integer overflow, or undefined behavior

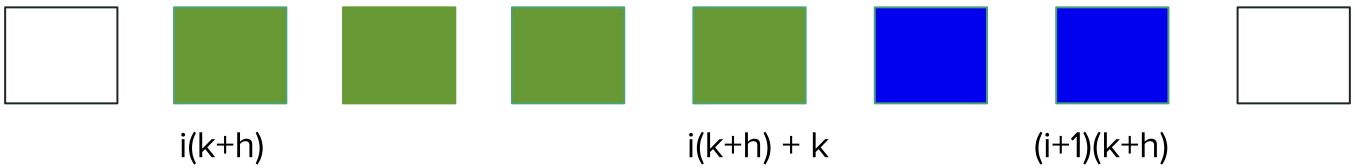
A bit stronger - FEC should not make things worse than doing nothing at all

2. If a data packet arrives in stream *R*, it appears in the outputted stream *D*
3. Every packet in *D* must have been in the original stream *O*

This is violated because of sequence number wraparound issue - we change to 64 bit sequence numbers and use serial number arithmetic [RFC 1982]

Towards a Stronger Spec

- Those specs are not enough: can satisfy by dropping all parities
- Want guarantee: with normal network conditions, FEC helps by recovering lost packets
- Simplest: if at least k packets per batch (packets *i(k+h)* to *(i+1)(k+h)* in the encoded stream) are received, all data packets in batch recovered



Towards a Stronger Spec

- Not true: timeouts caused by reordering, duplication, delay
- Would like to assume: packets “close” in *E* (encoded) are “close” in *R* (received) - then batch arrives before timing out
- We will formalize metrics for measuring these network conditions and prove (in Coq) that under reasonable bounds, this is true and so we can guarantee packet recovery
- Specifically, need to formalize bounds on reordering, duplication, and delay

Formalizing Properties of Packet Streams - Reordering

- Many existing metrics for measuring packet reordering [RFC 4737, 5236]
- We use Reorder Density (RD) [NETWORKING 2005] - comprehensive, good performance, robust [International Journal of Communication Systems 2008]
- Idea: measure displacement - difference between arrival sequence number and expected sequence number (RI)
- We will assume a global bound on the displacement
    - In measured experiments, displacement tends to be quite small (≈ 50)
- Note: intentionally ignores duplicate and missing packets

seq[i]	1	2	3	6	4	5	7
RI[i]	1	2	3	4	5	6	7
d[i]	0	0	0	-2	1	1	0

seq[i]	1	4	3	5	3	8	7	6
RI[i]	1	3	4	5	x	6	7	8
d[i]	0	-1	1	0	x	-2	0	2

Formalizing Properties of Packet Streams - Duplicates/Timeouts

- Very few existing metrics for duplicates, difficult to use with reordering metrics
    - Want: displacement bound ⇒ packets arrive close together, not true with duplicates
    - Only get weak, multiplicative bounds
- We use metric inspired by RD: every pair of duplicate packets have at most *m* packets in between them
    - If view duplicates as sent in sequence, this is essentially the difference between the displacements
- Timeouts are difficult - we need assumptions about network speeds and time between packets
- Instead, use alternate approach - measure time in *packets*, not seconds

A New Timeout Mechanism

Change implementation to count (estimate) the *number of unique packets received*, use this to measure time, and always delete expired blocks

- Keeps data structures (provably) small, size does not depend on network speeds
- No overhead: program already checks for duplicate packets
- Allows Producer to delay
- Consumer no longer needs external state (system time)
- Performance more predictable, no space leaks
- Spec becomes much cleaner: duplication and reordering both count unique packets; we get strong additive bounds

A Strong Spec

Suppose that the packet streams satisfy the following conditions:

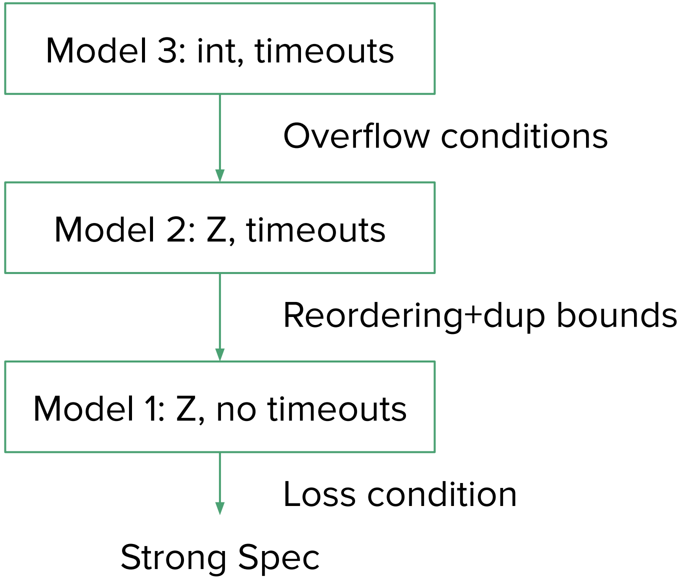
1. k and h (the FEC parameters) are fixed for all packets
2. For all packets, the magnitude of the displacement between E (encoded) and R (received) is bounded by d
3. Any two identical packets in R have at most m packets between them
4. The timeout threshold is at least $k+h+2d+m$ and less than 2^{31}
5. All sequence numbers are unique and less than 2^63, $0<k\leq 127$, $0<h\leq 128$

Let i be between 0 and $|O|/k$, and suppose that at least k packets of the $k+h$ packets between positions $i(k+h)$ and $(i+1)(k+h)$ in stream E appear in stream R

Then, all packets in batch i (packets $i*k$ to $(i+1) *k$) appear in D, the decoded stream

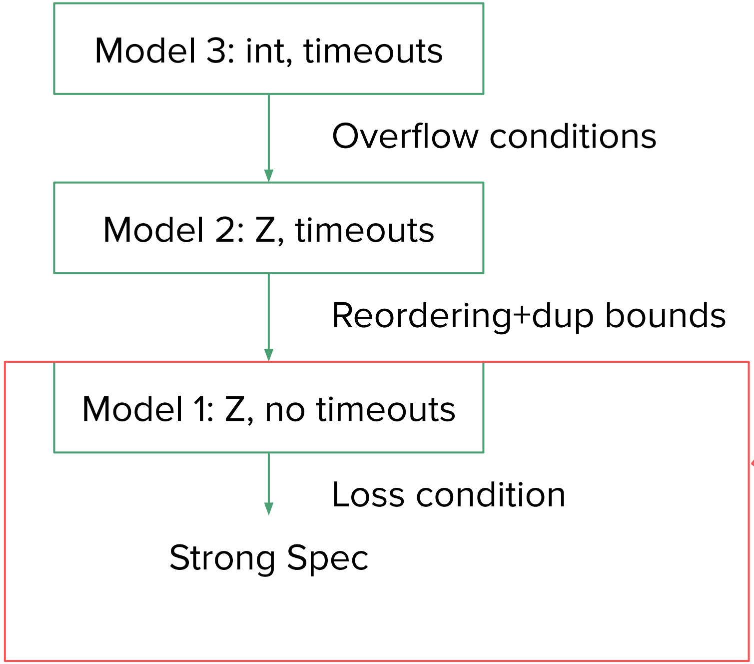
Corollary: if all of these conditions hold for all such i, streams O and D have the same packets

Proving the Program Correct



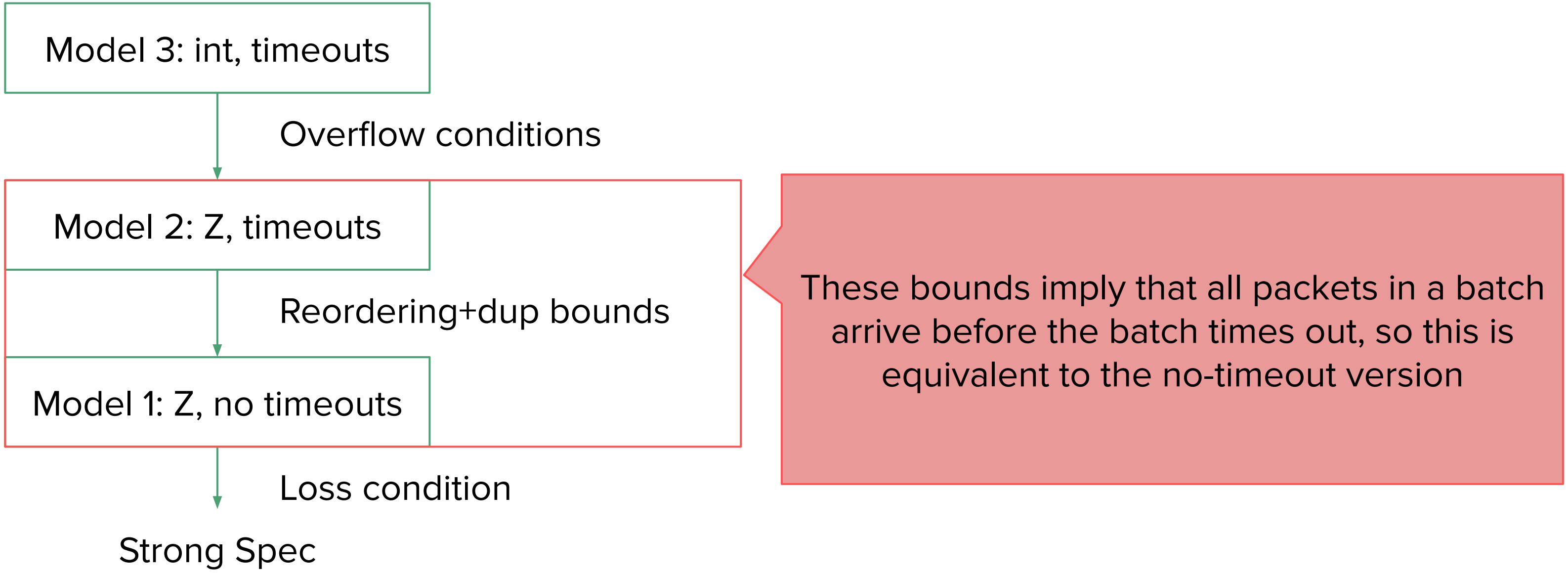
After writing new C program, we write a close functional model of the system in Coq and prove it correct according to the 3 levels of specification above

Proving the Program Correct

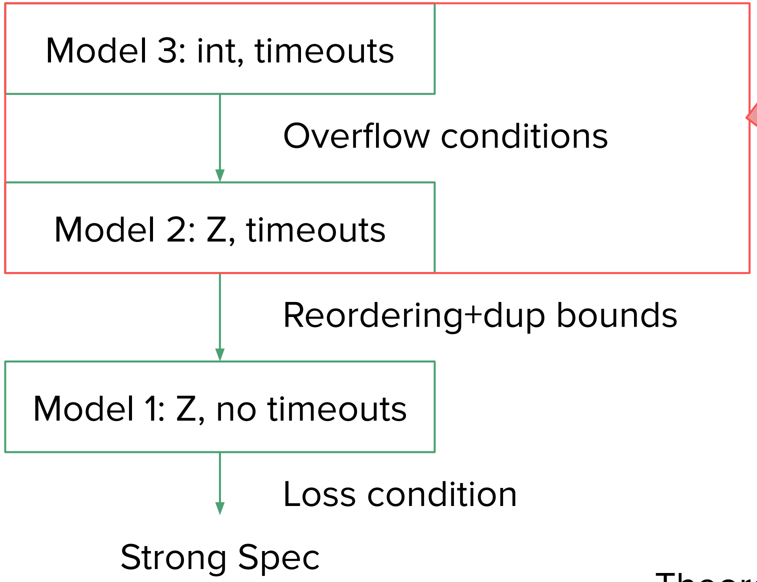


With no timeouts, eventually enough packets in batch arrive (by loss condition), so the decoder recovers missing packets and therefore all packets in batch appear in D

Proving the Program Correct



Proving the Program Correct



We formalize serial number arithmetic and prove that all integer comparisons occur between values within 2^{31}, which means that comparison is exactly mathematical integer comparison

```
int seq_cmp(unsigned int i1, unsigned int i2) {
return ((int)(i1−i2)); }
```

Theorem: Suppose $| z_1 - z_2 | < 2^{31}$.
Then, $\text{seq\_cmp}(z_1 \% 2^{32}, z_2 \% 2^{32}) < 0$ iff $z_1 < z_2$.

Conclusion

- We proved correct in Coq a close model of a real-world packet error-correction system, developing a simpler, more predictable, provably correct program that recovers more packets
- We developed a methodology for specifying and verifying such end-to-end network functions, including
    - Different layers of specifications to identify stronger guarantees in “good” scenarios and weaker ones in worst-case scenarios
    - Formalizing external network behavior (reordering, duplication, delay, and loss) and proving spec assuming bounds on this behavior
    - Formalizing and using serial number arithmetic to handle long-running programs with integer wraparound
    - Using refinement to simplify proofs and identify specific assumptions necessary for each guarantee
- Proofs available at https://github.com/verified-network-toolchain/Verified-FEC/tree/end-to-end

References

Abhashkumar, A., Gember-Jacobson, A., Akella, A.: Tiramisu: Fast multilayer network verification. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). pp. 201–219. USENIX Association, Santa Clara, CA (Feb 2020)

Bare, A.A., Jayasumana, A.P., Banka, T.: Metrics for degree of reordering in packet sequences. In: Proceedings LCN 2002. 27th Annual IEEE Conference on Local Computer Networks. p. 0333. IEEE Computer Society, Los Alamitos, CA, USA (Nov 2002).

Beckett, R., Gupta, A.: Katra: Realtime verification for multilayer networks. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). pp. 617–634. USENIX Association, Renton, WA (Apr 2022)

Beckett, R., Gupta, A., Mahajan, R., Walker, D.: A general approach to network configuration verification. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. p. 155–168. SIGCOMM '17, Association for Computing Machinery, New York, NY, USA (2017)

Bush, R., Elz, R.: Serial Number Arithmetic. RFC 1982 (Aug 1996).

Cohen, J.M., Wang, Q., Appel, A.W.: Verified erasure correction in Coq with MathComp and VST. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification. pp. 272–292. Springer International Publishing, Cham (2022)

Guo, D., Chen, S., Gao, K., Xiang, Q., Zhang, Y., Yang, Y.R.: Flash: Fast, consistent data plane verification for large-scale network settings. In: Proceedings of the ACM SIGCOMM 2022 Conference. p. 314–335. SIGCOMM '22, Association for Computing Machinery, New York, NY, USA (2022).

Jayasumana, A., Piratla, N., Banka, T., Bare, A., Whitner, R.: Improved packet reordering metrics. RFC 5236, RFC Editor (June 2008)

References

McAuley, A.J.: Reliable broadband communication using a burst erasure correcting code. In: Proceedings of the ACM Symposium on Communications Architectures & Protocols. p. 297–306. SIGCOMM ’90, New York, NY, USA (1990).

Morton, A., Ciavattone, L., Ramachandran, G., Shalunov, S., Perser, J.: Packet reordering metrics. RFC 4737, RFC Editor (November 2006)

Piratla, N.M., Jayasumana, A.P.: Metrics for packet reordering—a comparative analysis. International Journal of Communication Systems 21(1), 99–113 (2008).

Piratla, N.M., Jayasumana, A.P., Bare, A.A.: Reorder density (RD): A formal, comprehensive metric for packet reordering. In: Boutaba, R., Almeroth, K., Puigjaner, R., Shen, S., Black, J.P. (eds.) NETWORKING 2005. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems. pp. 78–89. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

Pirelli, S., Valentukonytė, A., Argyraki, K., Candea, G.: Automated verification of network function binaries. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). pp. 585–600. USENIX Association, Renton, WA (Apr 2022)

Zaostrovnykh, A., Pirelli, S., Iyer, R., Rizzo, M., Pedrosa, L., Argyraki, K., Candea, G.: Verifying software network functions with no verification expertise. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. p. 275–290. SOSP ’19, New York, NY, USA (2019).

Zaostrovnykh, A., Pirelli, S., Pedrosa, L., Argyraki, K., Candea, G.: A formally verified NAT. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. p. 141–154. SIGCOMM ’17, New York, NY, USA (2017).

Zhang, K., Zhuo, D., Akella, A., Krishnamurthy, A., Wang, X.: Automated verification of customizable middlebox properties with Gravel. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). pp. 221–239. USENIX Association, Santa Clara, CA (Feb 2020)