Designing a Multipath Transport Protocol

Costin Raiciu

Joint work with Mark Handley

Part of the Trilogy EU Project

Problem

- Design a multipath version of TCP
  - In order, reliable delivery
  - Byte addressed
- Why TCP?
  - Biggest chance of getting deployed
  - Any new protocol must interoperate with TCP anyway
- Why Multipath?

Why Multipath?

- It is possible today
  - A lot of multihoming - x% of Ases
  - More devices with multiple internet connections
    - Mobile phones
    - Just share your wireless with your neighbour
  - Theoretical results for stable algorithms
  - Applications are doing it anyway!
  - No changes required in the core
- For the endpoints
  - Better robustness
  - Better throughput
  - Increased competition among ISPs

Why Multipath? [2]

- For the Internet
  - A natural solution to multi homing
  - Smaller routing table - fewer “more specifics”
  - Less need for fast convergence
  - Better address aggregatibility
  - Less routing table churn
- For the operators
  - Higher link utilization
  - Reduced operational costs
  - New services offered to clients

Path Selection & API

- Multiple addresses per endpoint for path diversity
  - Signaled at transport layer
- Unmodified sockets API with semantic changes
  - bind
    - If address is INADDR_ANY bind all non-local addresses - multi homed client apps unchanged (servers too?)
    - Allow multiple binds with specific local addresses & ports - modified apps can control which addresses are used
  - getsockopt
    - New option to find out connected subflows
  - setsockopt
    - New option to unbind an address (ugly)

Multipath TCP Design

- Connection Management
- Sequence Numbers and Acks
- Data Striping
- Security

Multipath TCP Design

- Connection Management
- Sequence Numbers and Acks
- Data Striping
- Security

High Level Design Goal

- **Path Isolation** - events on one path should not affect other paths
  - Delay variation
  - Packet drops
  - Failures
- We’d like to avoid
  - A congested path stalling other paths
  - A failed path stalling other paths

Splitting TCP Functionality

- Single path TCP - loss detection, congestion detection, flow control, in-order delivery, reliability
- Multipath TCP
  - Each path detects losses and congestion
    - Isolate response to proper path
    - Reduce traffic on congested paths only
  - The connection implements
    - In-order delivery - reorder scattered app data
    - Flow control - a single receive window, advertised on all paths
    - Reliability - retransmissions may be sent on different paths

Sequence Numbers

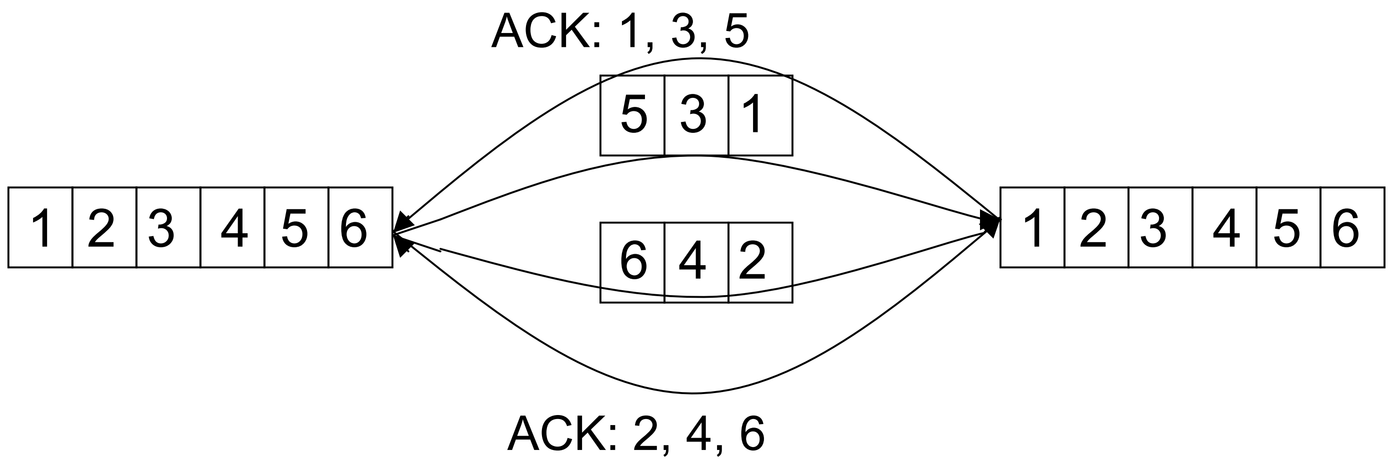
- We have:
  - Multiple subflows
  - Single data sequence space
- What sequence numbers should the subflows use?
  - Data sequence numbers?
  - Independent sequence numbers + mapping to data sequence numbers?

Unique Sequence Space

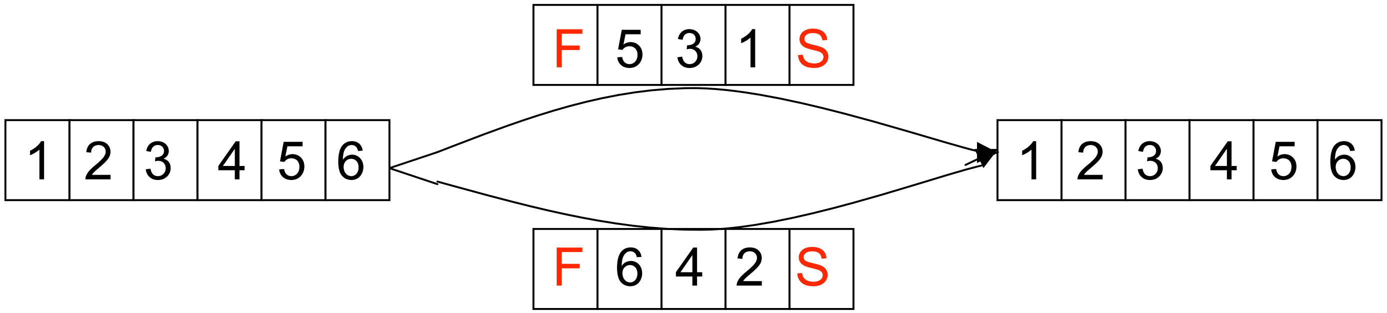
- Stripe the data sequence numbers across subflows
- Use data cumulative ack

Unique Sequence Space

- Stripe the data sequence numbers across subflows
- Use data cumulative ack

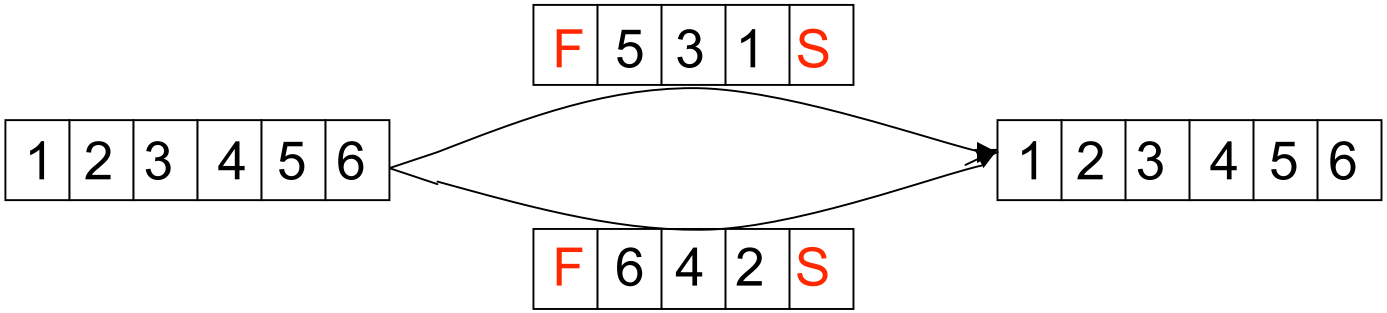


Subflow Signaling



- TCP assigns sequence numbers to SYNs and FINs

Subflow Signaling

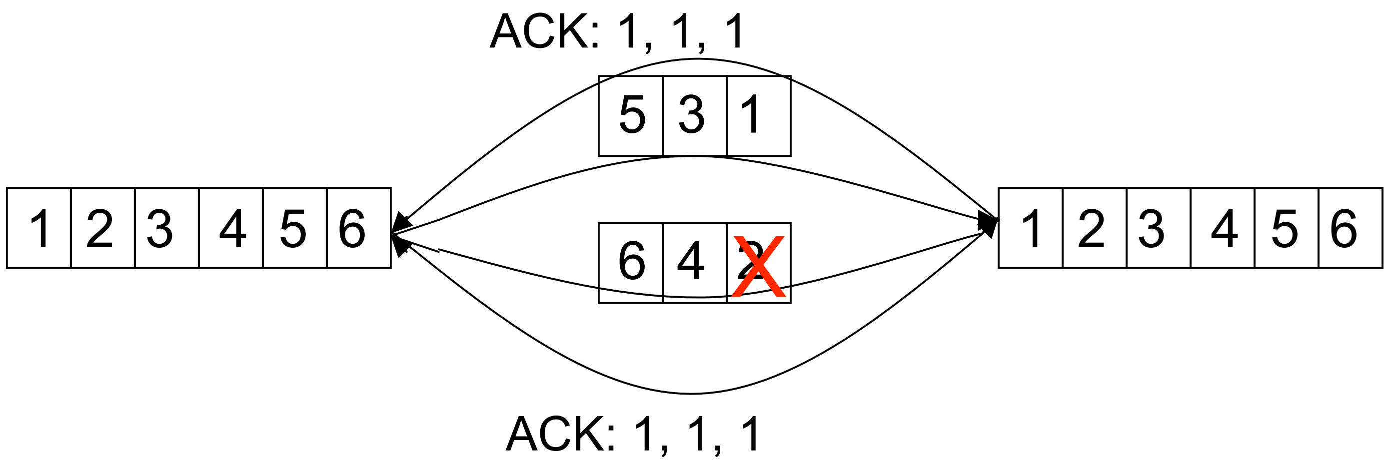


- TCP assigns sequence numbers to SYNs and FINs
- Single sequence space can't

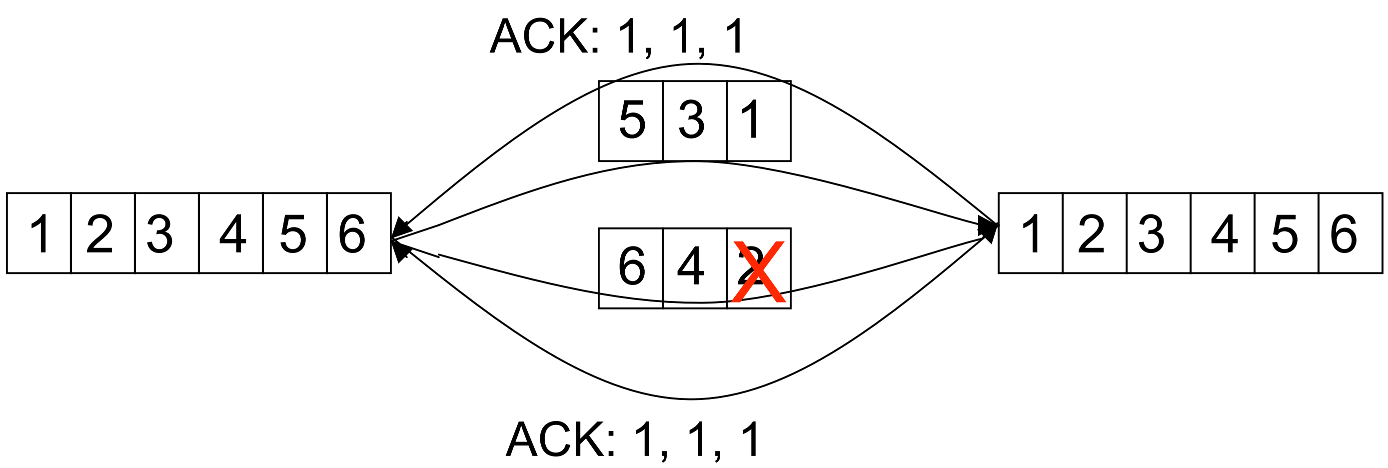
Problem: difficult to ack subflow-related messages

Fix: must be acked out of band

Lost Data Packet



Lost Data Packet



Problem: cannot tell which subflows lost data

Fix: use per flow SACK + data cumulative ack

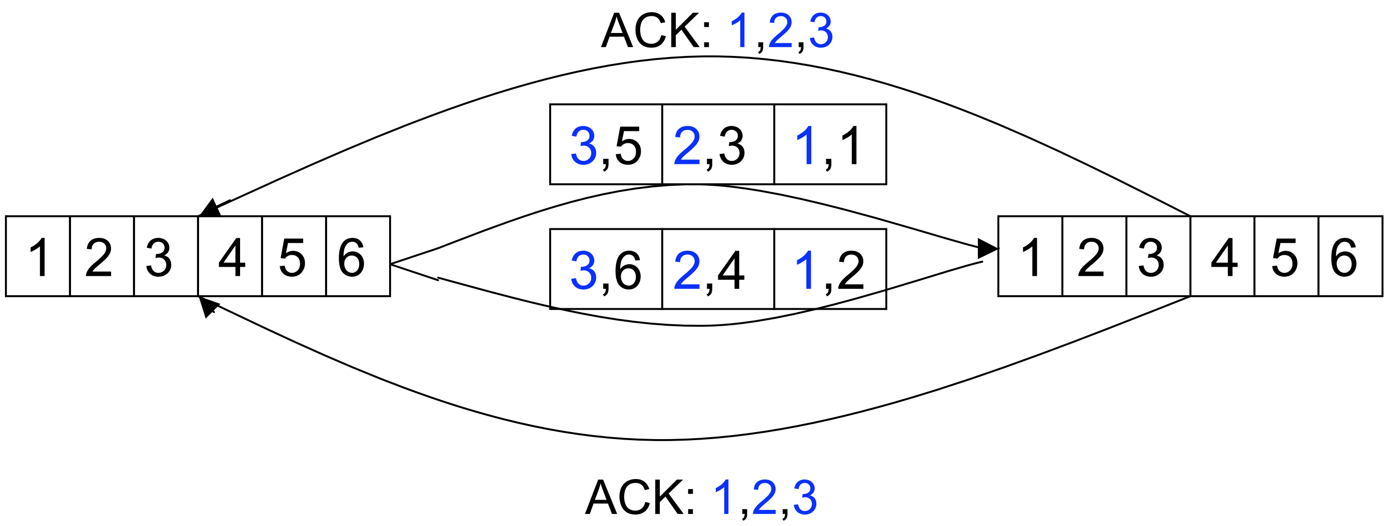
Possible issues if cumulative ack falls behind:

- SACKs get big
- Receiver has to hold more state

Multiple Sequence Spaces

- Each subflow has its own sequence number space
- Data sequence numbers are mapped on the subflow that sends them
  - Simply insert the data sequence number too
- Use cumulative ack on each subflow
  - Subflow seq nos are gapless
  - Data cumulative ack, SACK not mandatory
    - Could use as optimization
    - Or for security

Multiple Sequence Spaces



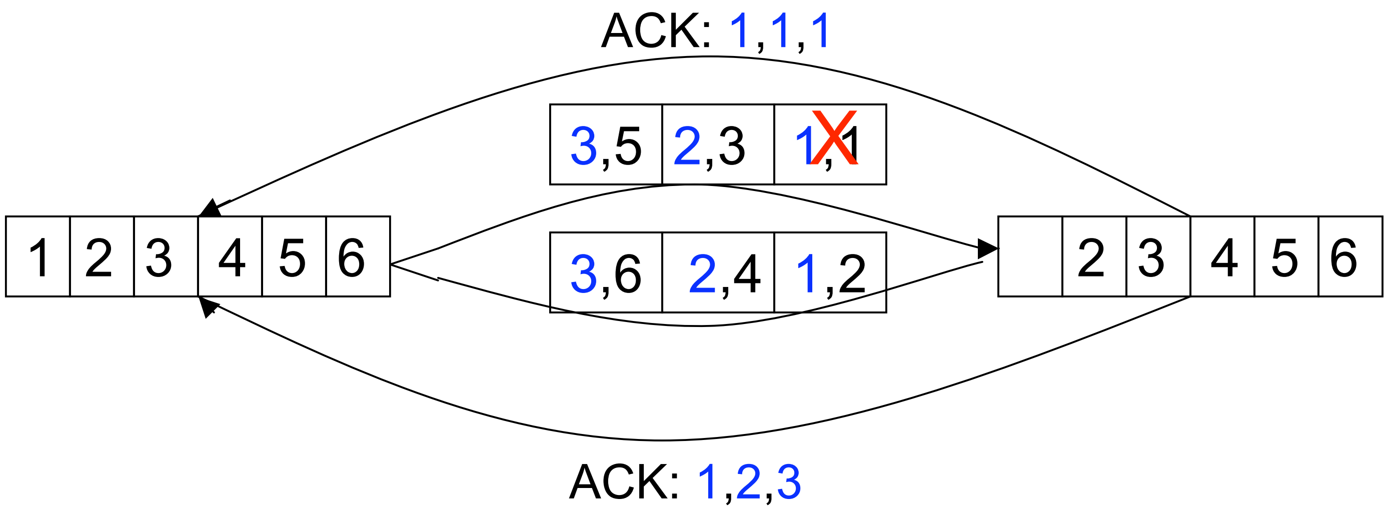
Subflow Sequence Number

Data Sequence Number

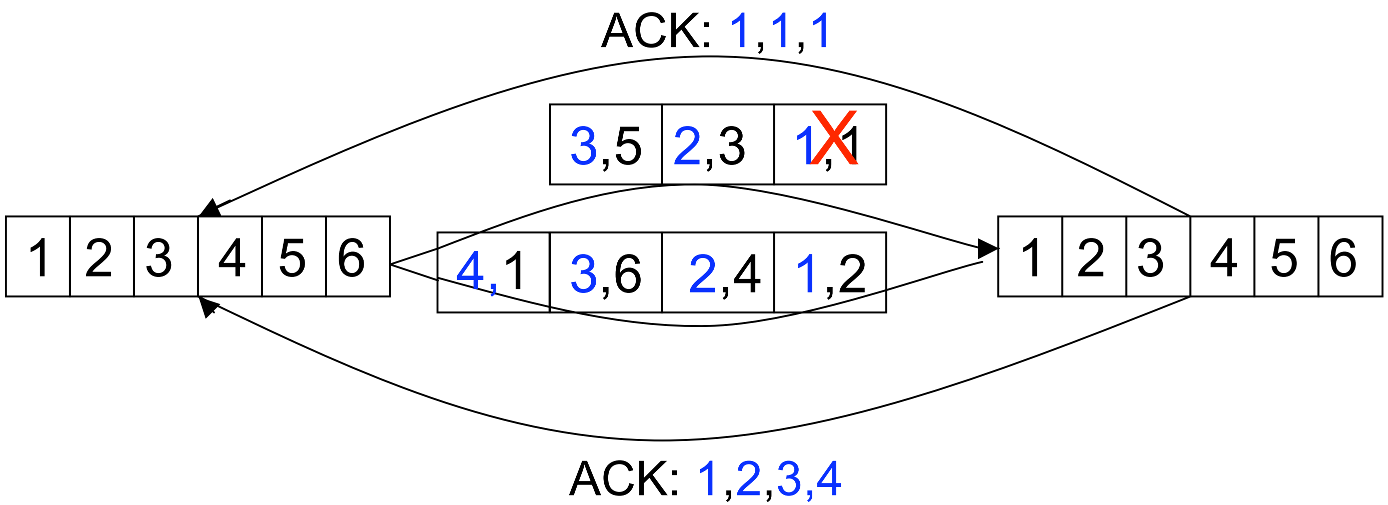
Multiple Sequence Spaces (2)

- Advantages
  - Cumulative acks for each subflow summarize connection state succinctly
  - Ack SYNs and FINs elegantly - no data mapping
  - Each subflow looks like TCP over the wire
  - Bandwidth - half of that of a single SACK block
- Disadvantages
  - Uses a bit more bw on the forward path
  - Retransmissions

Retransmissions



Retransmissions



Gaps in subflow seq nos

- What about the initial subflow (1,1) mapping?
  - Could re-map the subflow seqno to other data seqno
    - Confuses traffic normalizers
    - Sender unsure which data arrived when gets ACK
  - Or never use the subflow seqno again
    - Re-send the initial packet always, or
    - Send **“get-over-it”** packets
    - One of the above needed for correctness!

Data Striping Policy

- Which subflow gets which data?
- If we have lots of data in the send buffer
  - Want max throughput?
    - Use large receive window
    - Just keep cwnd full at all times for all paths
  - Want small packet delay too?
    - Delay dictated by reordering time
    - Compute expected arrival time for each segment on each subflow
    - Send on min
- If there’s little data in the send buffer
  - send on fastest subflow with open cwnd

Data Striping Policy [2]

- Small Flows
  - Just send everything on all paths - ensures minimum delay and robustness
- Redundancy is useful when
  - Paths lose packets
  - The connection starts
  - Probing bad paths
- Redundancy hurts!
  - Useless resending of the same data
- No incentive to use redundancy in normal operation

Related Work

- pTCP - most mature
  - Separate seq space
  - Each flow congestion controlled, three way handshake
  - Minuses:
    - Retransmit different data
    - Independent TCP flows - not TCP friendly on bneck links
    - Ns only evaluation
- mTCP
  - Single seq space, single ack path
  - RON for multipath
- R-MTP - targets wireless links
  - Probes bandwidth periodically and adjusts rate
- CMT - changes to SCTP for multipath

Summary

- Designing a multipath TCP is not as straightforward as we initially thought
- Still working on it
- Implementation under way

Connection Management

- Interoperation with TCP => need three way handshake for initial subflow
- First handshake must signal
  - Multipath availability
  - IP list for the endpoint - to avoid extra RTTs
- Additional subflows
  - Three way handshake for each?
    - Probes path characteristics
    - Is the same as the initial flow
    - Should send data on subsequent SYNs
  - Or use signaling on existing subflows?
    - Muddy semantics for seq nos, slower

Connection Initiation

- Initial SYN/SYN ACK special
  - Include connection token
  - Include IP address used to connect
  - Optionally include IP list for multipath
    - By convention, on SYN ACK (multi-homed server)
- Subsequent SYNs include the token
  - Used for connection demuxing
- Who opens additional subflows?
  - Usually the client
  - Mobile Server: pick a reachable client path:
    - When SYNs are received, a check is made to see if the packet's source address matches the IP provided.

Connection Initiation (2)

- TCP options space is small (40B) and already crowded: window scale, mss, sack, etc.
  - IPv4 addresses: max 9 addresses
  - IPv6 addresses: max 2 addresses.
- If client is multihomed, no need to send IP List
- Possible workarounds
  - Extend options space using options?
    - Needs one RTT in the general case
    - Fast if server is multi-homed, so IP list is sent on SYN ACK
      - Bad if middle boxes remove options on SYN/ACK
  - Send IP lists on additional data packets?

Connection Termination

- Subflows can be destroyed by sending FINs
- If a path fails, how do we terminate the associated subflow?
  - TCP style timeouts - KEEP ALIVE, etc.?
    - Keeps state at servers, very slow
  - Send metadata on any working subflow, indicating remote end to drop the subflow?
    - Muddies sequence number semantics, introduces security issues
  - When application executes shutdown/close send **DATA FIN**, which tells the remote host to close all subflows
    - Less flexibility than above. Challenge: reduce delay on path fail
- May need to revoke advertised addresses
  - Helps mobile hosts

Security

- Goal: at least the security offered by TCP
  - Isolate subflows: subverting one subflow should not affect the entire connection
- We use
  - MACs in every packet - instead of TCP checksum
  - Subflow IDs
  - ECN nonces

Message Authentication Codes

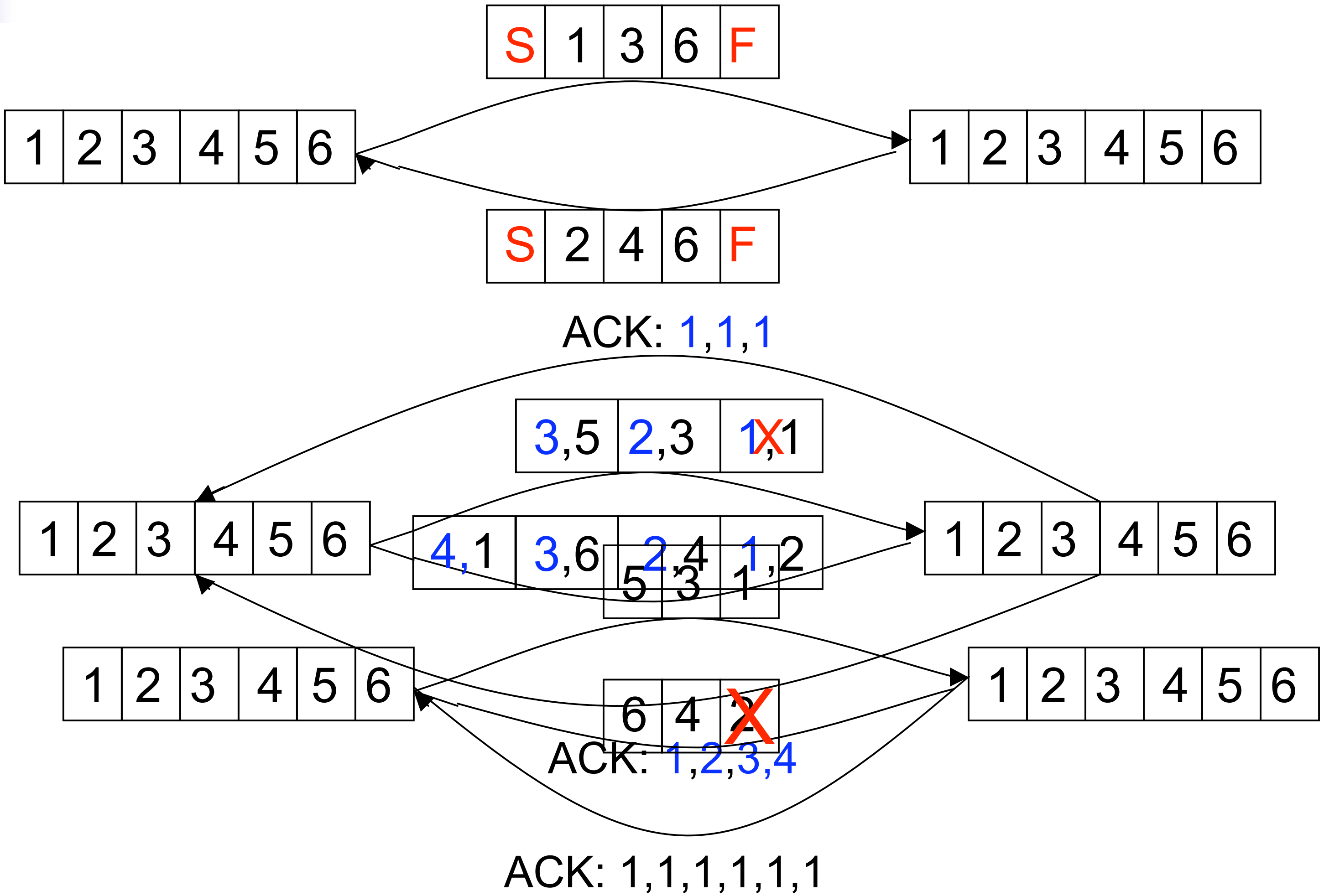
- In every packet besides the initial SYN exchange
- Computed on transport header and data
- Also serves as checksum
- Keyed with the tokens
  - Other key authentication schemes could be used
- Two modes
  - Cheap: replaces the 16 bit TCP checksum
  - Full: complete output of a collision resistant hash function, such as SHA

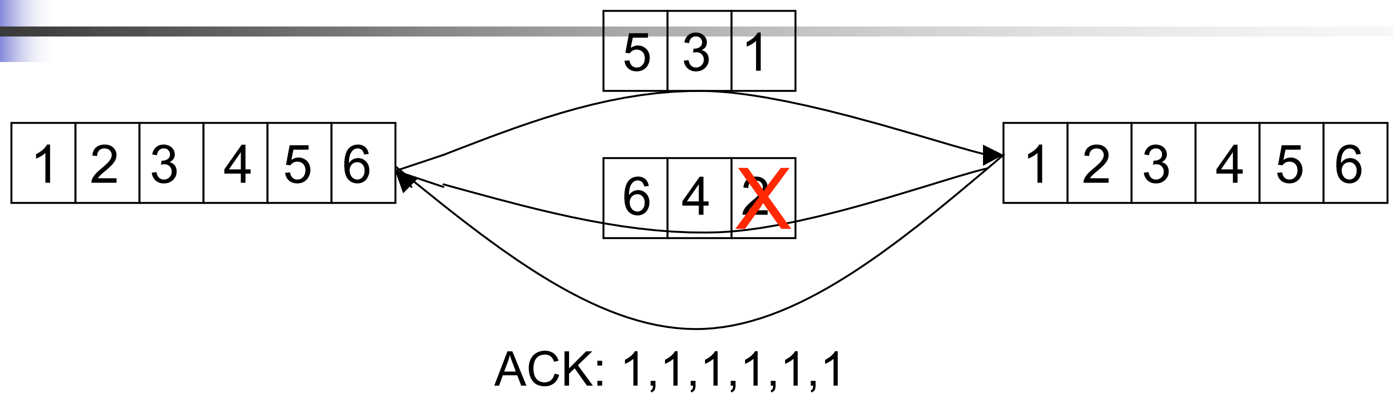
Other Security Mechanisms

- Subflow IDs
  - Each subflow gets an ID from an always increasing sequence (4 or 8 bytes)
    - Sequences maintained per endpoint
    - Included in every SYN besides the initial one
  - Prevents against SYN replay attacks
- Use ECN Nonce
  - Protects against misbehaving receivers

Security Analysis

Attacker	Attack	Works	Defences
Blind	Inject Data	No	MAC, Seq Nos and subflow IDs
	Replay Data	No	MAC, Seq Nos and IDs
Eavesdropper	Close Data Receive Window	No	Data cumulative ack
	Get Over It Packet	Yes	Expensive packets
	Create Subflows	Yes	?
	Close Subflows	Yes	Avoid DATA FIN
Man in the Middle	*	Yes	PKI + interlock protocol


S 1 3 6 F
1 2 3 4 5 6
1 2 3 4 5 6
S 2 4 6 F
ACK: 1,1,1
3,5 2,3 1,1
1 2 3 4 5 6
4,1 3,6 2,4 1,2
5 3 1
1 2 3 4 5 6
1 2 3 4 5 6
6 4 2
1 2 3 4 5 6
ACK: 1,2,3,4
ACK: 1,1,1,1,1,1


5 3 1
1 2 3 4 5 6
6 4 2
1 2 3 4 5 6
ACK: 1,1,1,1,1,1