High-performance data networking

CESE4045

Fernando Kuipers

TU Delft

1

1

Organization

- TAs: Chenxing Ji Adrian Zapletal
- Prerequisites: Networking basics + programming (Python)
- Mix of:
  - Theory (Slides)
  - Exercises (Reader)
  - Q&A sessions
- Exam (25/01/2024) covers both theory (slides) & exercises (reader)

TU Delft

2

2

Tentative schedule

15/11 Lecture*: Basics
17/11 Q&A: Exercise 1 (starting 09:30)
22/11 Lecture: Software-Defined Networking (SDN)
24/11 Q&A: Exercise 2 (starting 09:30)
29/11 Lecture*: Quality-of-Service (QoS)
01/12 No class
06/12 Q&A: Exercise 3 + Lecture*: Multicast Content Distribution
08/12 Lecture: Network resilience
13/12 Q&A: Exercise 4 + Lecture: P4 (part 1)
15/12 Lecture: P4 (part 2)
20/12 Lecture: Software-defined cellular networks
22/12 Q&A: Exercise 5 (starting 09:30)

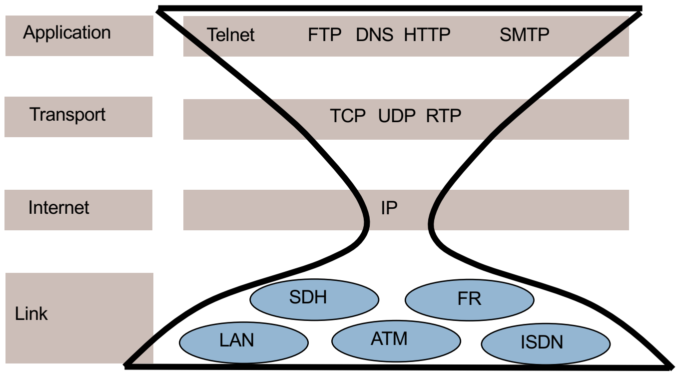
* Part of the content and slides from these lectures are from Prof. Piet Van Mieghem and his 2011 book “Data Communications Networking”.

TU Delft

3

3

TCP/IP: “Hourglass” design



TU Delft

5

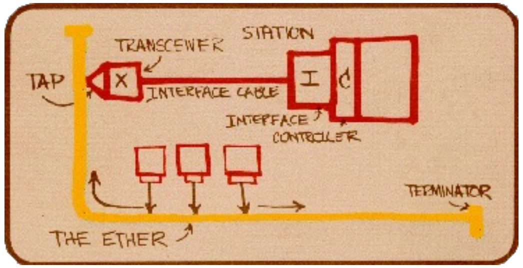
5

Ethernet
Local Area Network - LAN

6

6

Ethernet



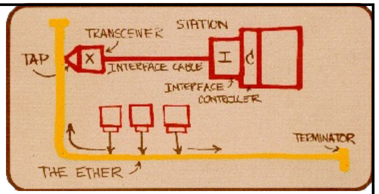
Original drawing by Robert Metcalfe (1976)

Metcalfe's ACM Turing award presentation
https://www.youtube.com/live/QRfRT19N2Rg?si=jeV8SwlJABhxRJOi

7

7

Ethernet



- *Bus (Ether)*: all stations share a single communication channel (distributed access control)
- *Broadcast*:
  - all transceivers receive every transmission
  - host interface filters among packets those intended for the corresponding computer
- *Best-effort delivery*: no notification about packet receipt

TU Delft

8

8

Ethernet Access: CSMA/CD

- Carrier Sense Multiple Access (CSMA)
  - multiple machines can access the Ethernet simultaneously
  - each machine determines whether the “ether” is free by sensing carrier wave propagation
  - each transmission is limited in duration and needs a minimum time between transmissions to prevent monopolization of the network
- Collision Detection (CD)
  - **Collision**: when two electrical waves cross, they become scrambled and meaningless.
  - **Collision detection**: each transceiver monitors the cable while transmitting to search for foreign signal interferences

TU Delft

9

9

Sending Rules

- **Non-persistent CSMA:**
  - Sensing is not continuously but repeated after random time.
  - If no collisions are sensed, the station sends a packet.
- **p-Persistent CSMA**:
  - Continuously sensing, but sending of packet probabilistically: send in current time slot with probability *p* and in another time slot with probability *1-p*

TU Delft

10

10

Binary exponential back-off

After each collision j a station chooses a random time uniformly in [0,2^j-1] timeslots (with a timeslot equal to the worst round-trip time):

IEEE 802.3:

- 1st: 0 or 1 equi-probable
- 2nd: 0 or 1 or 2 or 3
- 3rd: 0 or 1 or 2 or 3 or 4 or 5 or 6 or 7
- Increase until 10th
- Then constant until 16th
- Then failure

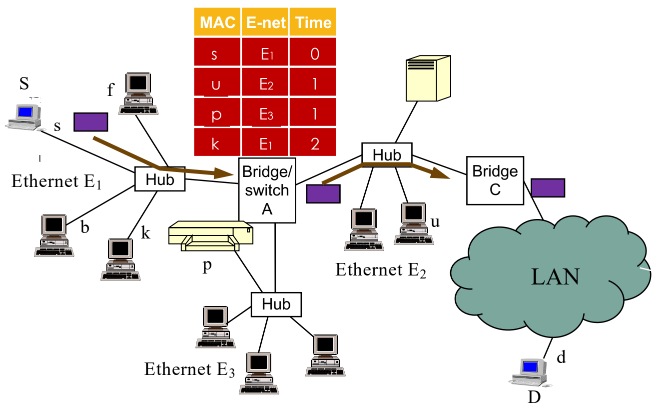
- Balances between prevention of collisions and waiting time
- Ethernet is a 1-persistent CSMA/CD scheme with binary exponential back-off

TU Delft

11

11

Example of an Ethernet LAN



TU Delft

12

12

Virtual LANs

- A LAN can be subdivided into virtual LANs. VLANs allow to simplify network design and deployment, because VLAN membership can be configured through software.
- IEEE 802.1Q supports virtual LANs (VLANs) on an Ethernet network.
- Ethernet frames get a VLAN tag.
- Frames with different tags may be treated differently by bridges and switches.

TU Delft

13

13

IP

14

IP

The Internet Protocol (**IP**) defines an unreliable, connectionless, best-effort delivery mechanism:

- specification of basic unit of transfer (packet)
- IP software handles the routing functionality
- IP includes the rules for unreliable, connectionless, best-effort delivery

15

IPv4 packet

0	4	8	16	19	24	31
vers	hlen	ToS	total length (bytes)			
identification			flags	fragment offset		
time to live		protocol	header checksum			
source IP address						
destination IP address						
IP options (if any)					padding	
user data						
...						

TU Delft

Fragmentation

IP header	data1 600 octets	data2 600 octets	data3 200 octets

fragment1 header	data1	Fragment1:	offset 0
fragment2 header	data2	Fragment2:	offset 600
fragment3 header	data3	Fragment3:	offset 1200

TU Delft

IPv6

Several major changes over IPv4:

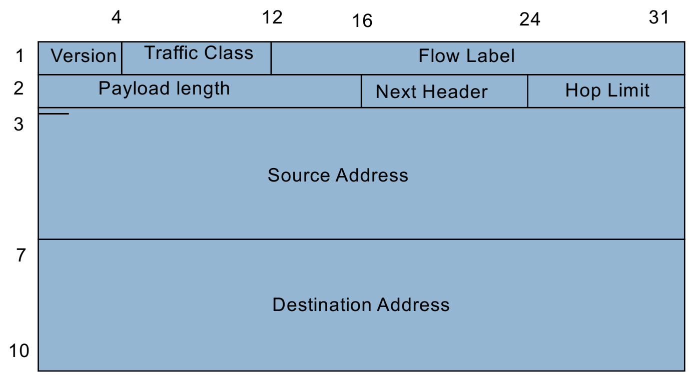
- *extended addressing capability*
  - 128 bits versus 32 bits in IPv4
- *a fixed format to all headers*
  - no option element, no header length field, but extension headers
- *no header checksum*
  - diminish cost of processing, other layers check & correct for errors
- *no hop-by-hop fragmentation*
  - unit of transmission = unit of control: use *path MTU discovery*
- *ICMPv6 and neighbour discovery*
  - support for address resolution and group management integrated into IPv6 (previously separate protocols)
- *easier management*
  - through auto-configuration (SLAAC)
  - longer addresses allow clear structuring of subnets and specific prefixes for different address types (e.g., ff00::/8 for multicast, fe80::/10 for link-local addresses, 2000::/3 for global unicast addresses)

TU Delft

18

18

IPv6 Header Format

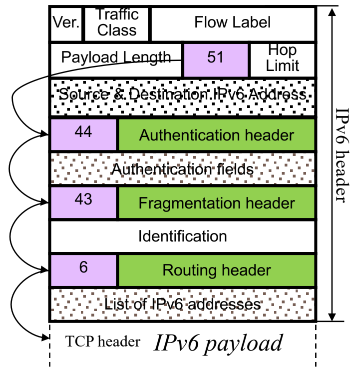


TU Delft

19

19

Extension Header



20

20

IPv6 Address Format

- 128 bit, canonically written in hexadecimal
  - `2001:0db8:0004:0a21:dead:beef:0000:1337`
  - Leading 0s can be suppressed and 16 0-bits can be shortened using ::
    (→ `2001:db8:4:a21:dead:beef::1337`)
- For global unicast addresses:
  - First 64 bits: Global Routing Prefix (n bits) + Subnet-ID (64 - n bits)
  - Last 64 bits: Interface-ID
  - Interface-ID often generated using the interface's MAC address

21

21

ICMPv6

- Integrates functions of ICMP (Internet Control Message Protocol), group management (for Multicast), and ARP (Address Resolution Protocol)
- Every IPv6 interface has a link-local address that identifies it and is used for link-local communication (e.g., for Neighbor Discovery)
- Neighbor Discovery: IPv6 nodes can find systems they are physically connected to (other nodes and routers)
  - Routers send periodic Router Advertisements (RAs)
  - Hosts can also solicit RAs by broadcasting a Router Solicitation
  - RAs contain Global Routing Prefix and Subnet-IDs that hosts need for generating global unicast IP addresses
- Duplicate Address Detection automatically resolves address conflicts

TU Delft 22

22

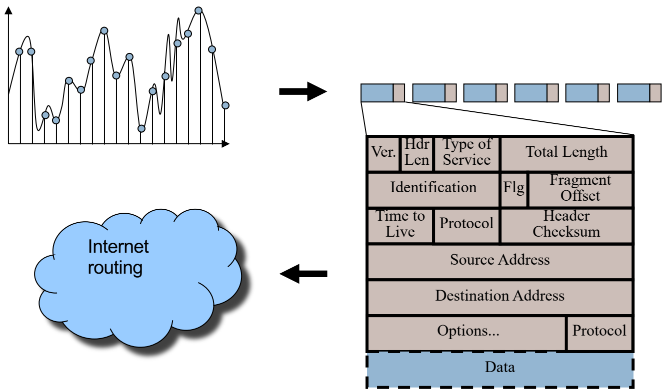
Stateless Address Auto-Configuration (SLAAC)

- Generation of link-local IP address
  - Either generate random Interface-ID (64-bit) or generate it from the hardware MAC address
  - Example MAC address (48 bit, following the IEEE 802 standard): `00:10:4b:4e:52:e4`
  - IPv6 Interface-ID (64 bit): `0210:4bff:fe4e:52e4`
    - Flip 7th bit (identifies whether address is local or universal) and insert `ff:fe` in the middle
  - IPv6 link-local address (128 bit): `fe80::0210:4bff:fe4e:52e4`
    - `fe80::` + Interface-ID
- Ask neighbors whether they own the generated address (Duplicate Address Detection)
  - If duplicate: generate new random Interface-ID
- Generation of global unicast address
  - Global Routing Prefix + Subnet-ID (from RA) + Interface-ID

TU Delft 23

23

IP packets



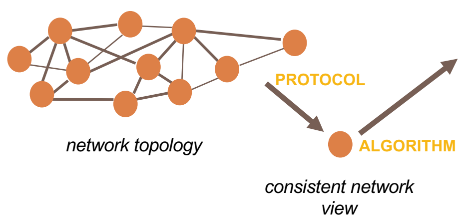
TU Delft

24

24

What is Routing?

- Finding one (or more) paths between two (or more) nodes in the network
  - often there is an optimality criterion (e.g. shortest path)
- Basic building blocks:



To	Link	Cost
A	local	0
B	1	1
C	3	1
E	3	6

forwarding table

TU Delft

25

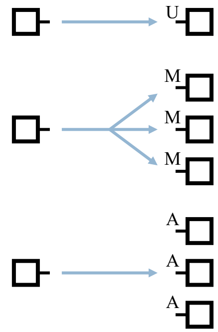
25

Basic Address Types

unicast:
for one-to-one communication

multicast:
for one/many-to-many communication

anycast:
for one-to-nearest communication

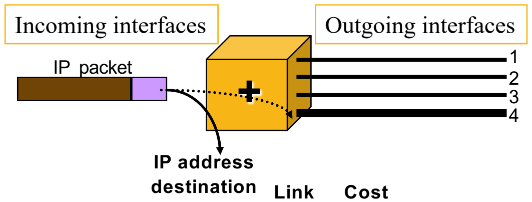


TU Delft

26

26

Forwarding



Forwarding TABLE

IP address destination	Link	Cost
IP_1	1	5
IP_2	4	2
IP_3	1	7
...	...	...

TU Delft

27

27

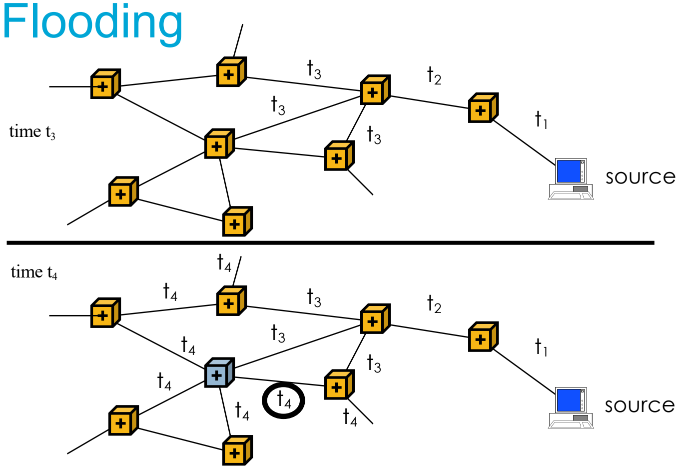
Distributed routing protocol families

- Distance-vector protocols (RIP; Bellman-Ford)
  - exchange list of distances with neighbors
  - maintain list of shortest distances
  - protocol itself constructs forwarding table
  - simple but vulnerable (e.g., count-to-infinity problem)
- Link-state protocols (OSPF; Dijkstra)
  - flood topology information
  - maintain entire map of network
  - local routing algorithm computes forwarding table
  - more robust but more complex

TU Delft 28

28

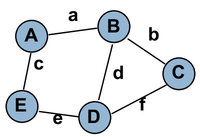
Flooding



TU Delft 29

29

OSPF: Link State Database



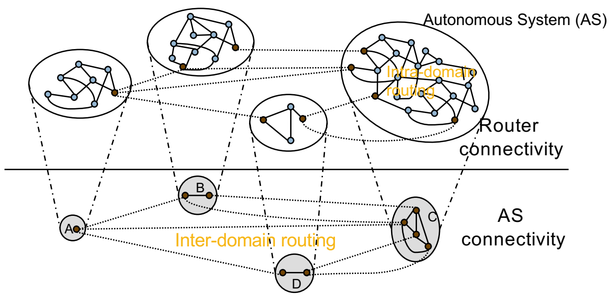
From	To	Link	Cost	Num
A	B	a	1	1
A	E	c	5	1
B	A	a	1	1
B	C	b	3	1
B	D	d	3	1
C	B	b	3	1
C	D	f	7	1
D	B	d	3	1
D	C	f	7	1
D	E	e	8	1
E	A	c	5	1
E	D	e	8	1

TU Delft

30

30

Two-level Routing Hierarchy



- Intra-domain routing (interior gateway protocols): Routing within one AS (e.g., OSPF and IS-IS)
- Inter-domain routing (exterior gateway protocols): Glues together different ASes (e.g., BGP)

TU Delft

31

31

Organization of Internet Routing

- More than 75,000 autonomous routing domains: A domain is a set of routers, links, hosts and local area networks under the same administrative control
- Domain's size: from 1 PC to millions of hosts
- Domains are interconnected in various ways

TU Delft 32

32

Types of domains: Transit

- Transit domains: A transit domain allows external domains to use its own infrastructure to send packets to other domains
- Examples: AT&T, UUNet, Level3, Opentransit, KPN,...

TU Delft 33

33

Types of domains: Stub

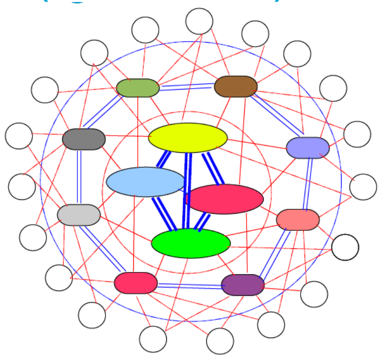
- Stub domains:
  A stub domain does not allow external domains to use its infrastructure to send packets to other domains
- A stub is connected to at least one transit domain
- Content stub domains: Google, BBC, ...
- Access stub domains: ISPs providing Internet access via cable, DSL, ...

TU Delft

34

34

Organization of the Internet (Traditional)



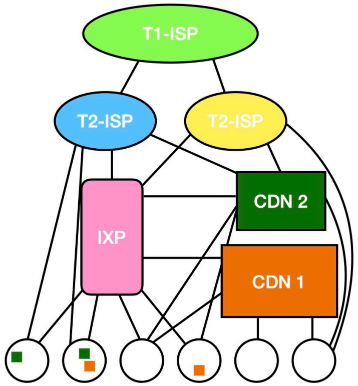
- Tier-1 ISPs
  - About 20 large ISPs
  - Provide transit service
  - E.g., Sprint
- Tier-2 ISPs
  - Regional or National ISPs
  - Customers of T1 ISP(s)
  - Providers of T3 ISP(s)
  - E.g., KPN
- Tier-3 ISPs
  - Smaller ISPs, Corporate Networks, Content providers
  - Customers of T2 or T1 ISPs

TU Delft

35

35

Organization of the Internet (Modern)



- Content Delivery Networks (CDNs)
  - Google, Meta, Netflix, IBM, Akamai, ...
- Direct peering
  - Mostly between ISPs and CDNs
- Points of Presence (PoPs)
  - CDN servers inside ISP networks
- Internet Exchange Points (IXPs)
  - Large interconnect facilities
  - AMS-IX, DE-CIX, ...

TU Delft

36

36

Border Gateway Protocol (BGP)

- Fundamental Internet routing protocol: the heart of the Internet's global connectivity
  - glue between ASs
  - complex protocol
  - only unicast
- Distance vector protocol enhanced with path vectors
  - Path vector contains entire path (list of ASs)
- 'best path': based on policies

TU Delft

37

37

Example BGP Table (RIB)

```
TIME: 08/28/01 15:02:05
TYPE: TABLE_DUMP/INET
VIEW: 0
SEQUENCE: 0
PREFIX: 3.0.0.0/8
FROM: 192.65.184.3 AS513
ORIGINATED: 08/28/01 12:42:08
ORIGIN: IGP
ASPATH: 513 209 701 80
NEXT_HOP: 192.65.184.3
STATUS: 0x1

TIME: 08/28/01 15:02:05
TYPE: TABLE_DUMP/INET
VIEW: 0
SEQUENCE: 1
PREFIX: 3.0.0.0/8
FROM: 64.211.147.146 AS3549
ORIGINATED: 08/28/01 12:41:54
ORIGIN: IGP
ASPATH: 3549 701 80
NEXT_HOP: 64.211.147.146
COMMUNITY: 3549:2256 3549:30840
STATUS: 0x1

TIME: 08/28/01 15:02:05
TYPE: TABLE_DUMP/INET
VIEW: 0
SEQUENCE: 2
PREFIX: 3.0.0.0/8
FROM: 193.148.15.85 AS3257
ORIGINATED: 08/28/01 11:30:59
ORIGIN: IGP
ASPATH: 3257 701 80
NEXT_HOP: 193.148.15.85
STATUS: 0x1
```

main body of entries skipped

```
TIME: 08/28/01 15:02:35
TYPE: TABLE_DUMP/INET
VIEW: 0
SEQUENCE: 38026
PREFIX: 218.67.0.0/17
FROM: 12.127.0.121 AS7018
ORIGINATED: 08/27/01 13:40:42
ORIGIN: INCOMPLETE
ASPATH: 7018 6453 4134
NEXT_HOP: 12.127.0.121
ATOMIC_AGGREGATE
AGGREGATOR: AS4134 202.97.32.22
STATUS: 0x1

TIME: 08/28/01 15:02:35
TYPE: TABLE_DUMP/INET
VIEW: 0
SEQUENCE: 38027
PREFIX: 218.67.0.0/17
FROM: 129.250.0.232 AS2914
ORIGINATED: 08/27/01 13:43:04
ORIGIN: IGP
ASPATH: 2914 6453 4134
NEXT_HOP: 129.250.0.232
MULTI_EXIT_DISC: 0
ATOMIC_AGGREGATE
AGGREGATOR: AS4134 202.97.32.22
COMMUNITY: 2914:420
STATUS: 0x1
```

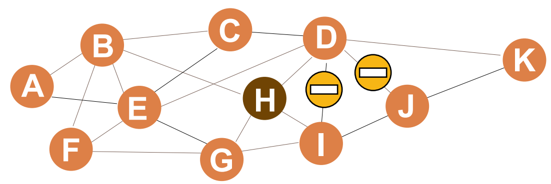
The syntax is explained in RFC 1771

TU Delft

38

38

BGP Path Vector Protocol



1. H receives the path vector from its BGP-neighbours about K:
   from B: BHIJK from G: GHDK
   from D: DK from I : IJK
2. H discards the path from B and from G that runs over itself
3. H uses a module with policy information and computes a "distance" for each remaining path from H to K
4. The path corresponding to 'best' distance is stored in H's routing table

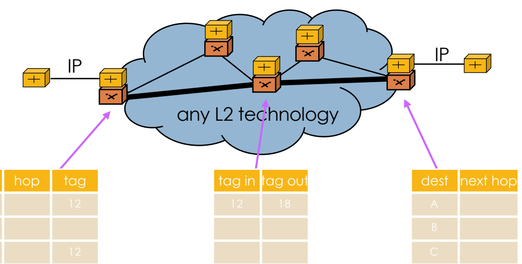
TU Delft

39

39

MPLS

Cisco's Tag Switching



Forwarding Information Base (FIB) Tag Information Base (TIB) L3 routing table

TAG BINDING TO
GROUP OF DESTINATIONS

LAYER 2 SWITCHING
NO LAYER 3 FORWARDING

TRADITIONAL
LAYER 3 FORWARDING

Multi Protocol Label Switching (MPLS) concepts

- Forwarding information (label) separate from content of IP header
- Single forwarding paradigm (label swapping) with hierarchy (label stacking) using multiple routing types (L3, L2)
- Flexibility to form forwarding equivalence classes (FEC) related to QoS or VPN
- Traffic engineering (TE): to override IP routing. TE is a powerful mechanism for current ISPs to
  - direct traffic away from congested paths
  - balance traffic across multiple paths
  - offer QoS in case of failures (back-up paths)

TU Delft

42

42

Labels

- Label:
  - short
  - fixed-length
  - local significance
  - exact match for forwarding
- Forwarding equivalency class (FEC):
  - packets that share the same next hop share the same label (locally)
- Needs label distribution mechanism

TU Delft

43

43

Label stacking

- Label stacking allows an indefinite number of labels to be used
- 3 Label operations:
  - Push
  - Pop
  - Swap
- Separates control and forwarding

TUDelft

44

44

MPLS

- Label Distribution Protocol (LDP) or RSVP?
  - LDP can support explicit routing (or QoS or constraint routing)
  - RSVP uses the routing tables of current non-QoS-aware routing protocols
- MPLS use cases
  - traffic engineering
  - scalable IP virtual private networks

TUDelft

45

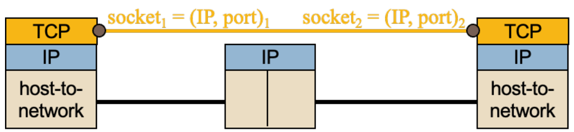
45

TCP

TU Delft

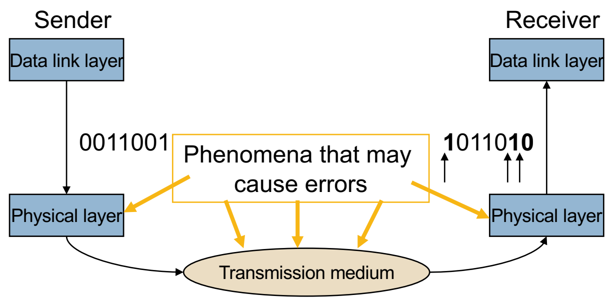
Transmission Control Protocol (TCP)

- Connection-oriented transport protocol
- Reliable transport
  - Positive acknowledgment (ACK) with retransmission
  - Principles of sliding window
- Details:
  - full duplex, ports, connections and endpoints ($socket_1$,$socket_2$)



TU Delft

Errors

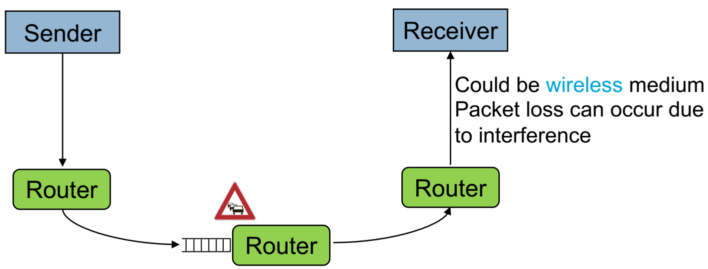


TU Delft

48

48

Packet Loss



Congestion: router buffer fills up with packets
If buffer full, router drops packets

TU Delft

49

49

Error Control

Three ways to deal with errors or packet loss after detection:

Retransmission — ARQ/TCP
- infrequent errors
- when time permits

Forward Error Correction — real-time services
- frequent errors
- when time does not permit retransmissions

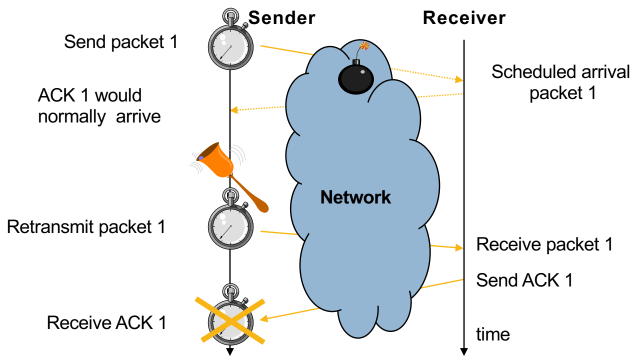
Discard — UDP
- when strict reliability is not required/too expensive

TU Delft

50

50

Time-out and Retransmission



TU Delft

51

51

TCP Retransmission Timeout

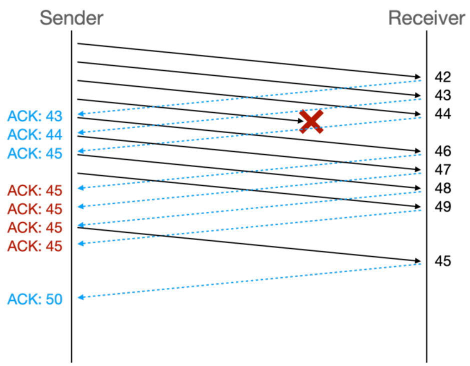
- Retransmission Timeout (RTO) after
  RTO = min{*rto_min*, *srtt* + 4 * *var*}
  rto_min originally specified as 1s; cur. default in Linux: 200ms
- Smoothed Round-Trip Time *srtt*:
  - At least once every Round-Trip Time (RTT): measure current RTT and store as *r*
    - Don't measure RTT for retransmissions
  - *srtt* = (1 - α) * *srtt* + α * *r*
  - Initial value for *srtt* (when measuring first RTT): *srtt* = *r*
- RTT Variation *var*:
  - *var* = (1 - β) * *var* + β * |*srtt* - *r*|
  - Initial value for *var* (when measuring first RTT): *var* = *r* / 2
- Typically, α = 1/8 and β = 1/4

TUDelft

52

52

Duplicate Acknowledgments



ACKs acknowledge received data by indicating the sequence number the receiver expects next

If an out-of-order packet arrives, the receiver sends duplicate ACKs

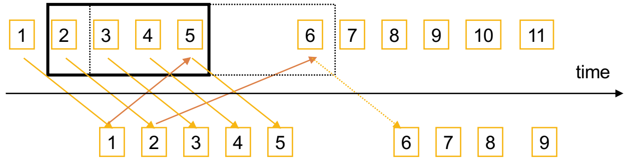
After three duplicate ACKs: sender considers the packet lost and retransmits (Fast Retransmit)

TUDelft

53

53

Sliding Window



Tuning sliding window impacts number of packets in network
Packets "inflight" = packets inside the network at a given time

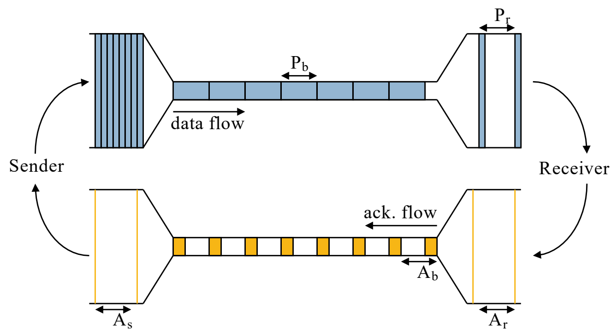
- TCP receive window (*rwnd*): upper limit on packets inflight specified by the receiver to ensure that the receiver is not overwhelmed by incoming packets (flow control)
- TCP congestion window (*cwnd*): upper limit on packets inflight determined by the sender to ensure that the network does not collapse under congestion due to too many packets (congestion control)

TU Delft

54

54

TCP is 'self-clocking'



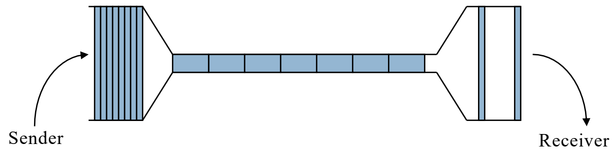
TU Delft

55

55

Congestion Control

- Problem: congestion at bottlenecks (typically router buffers)



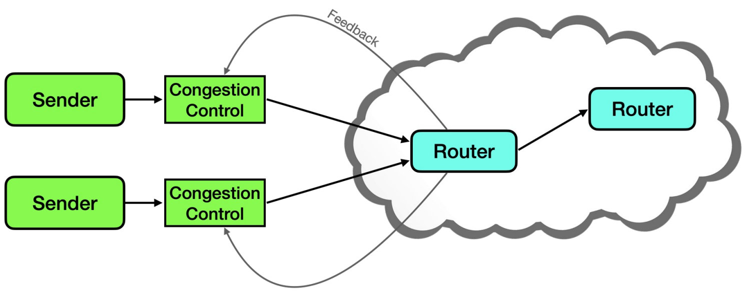
- Avoid network overload by throttling sending rate on sender side

TU Delft

56

56

Congestion Control



TU Delft

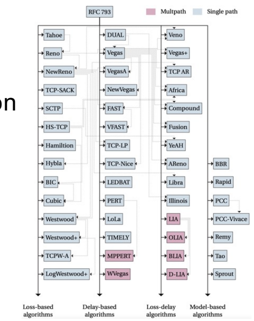
57

57

Types of Congestion Control Algorithms (CCAs)

- Loss-based CCAs
  - Use packet loss as congestion signal
  - E.g., Reno, Cubic, …
- Delay-based CCAs
  - Use increasing delay as congestion signal
  - E.g., Vegas, …
- Others
  - Hybrids between loss- and delay-based, model-based, …
  - E.g., BBR, …

- Window-based control
  - Limit inflight using congestion window (cwnd)
- Rate-based control
  - Limit sending rate by pacing packets at a certain rate



An overview of different CCAs ordered by type

TU Delft

58

58

TCP Reno: Overview

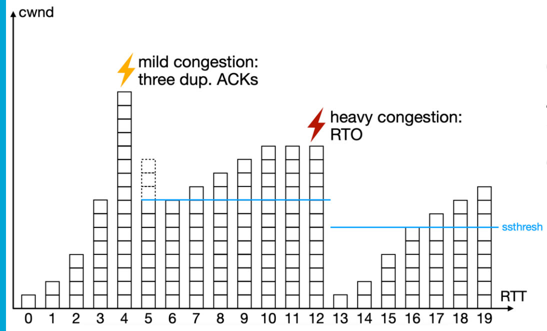
- Startup: probe available bandwidth
  - Slow Start: ramp up cwnd exponentially
- Steady state: keep up throughput and react to congestion
  - Congestion Avoidance: Additive Increase, Multiplicative Decrease (AIMD)
  - Raise cwnd linearly (additively)
  - Upon 3 duplicate ACKs:
    - decrease cwnd multiplicatively
    - Fast Retransmit of lost packet
    - Fast Recovery keeps up ACK-clocking
  - Upon RTO: reset to Slow Start

TU Delft

59

59

TCP Reno



0 – 4: Slow Start

4: *three dup. ACKs* received
5: Fast Recovery

6 – 10: Congestion Avoidance

10 – 12: no new ACKs arrive
12: *RTO* triggered

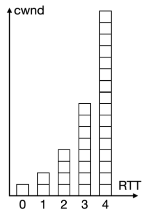
11 – 14: Slow Start
15 – 17: Congestion Avoidance

TU Delft

60

60

TCP Reno: Slow Start



Initialize *cwnd* = 1; *ssthresh* = ∞
Upon receiving an ACK:
  cwnd = *cwnd* + 1

If *cwnd* > *ssthresh*: switch to Congestion Avoidance
When *ssthresh* is not set, continue Slow Start until packet loss.

Note: nowadays, initial *cwnd* = 10 (typically)

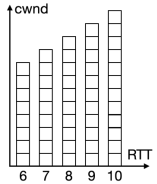
TU Delft

61

61

TCP Reno: Congestion Avoidance

Upon receiving an ACK:
    cwnd = *cwnd* + 1/*cwnd*
(Linear, Additive Increase)



Continue Congestion Avoidance until packet loss.

TU Delft

62

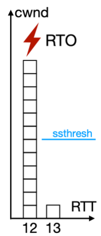
62

TCP Reno: Packet Loss

Packet loss detected if RTO or three duplicate ACKs received

Upon packet loss:
    ssthresh = max{inflight / 2, 2}
    inflight = number of not yet ACKed packets inside the network

If RTO ("heavy congestion"):
    cwnd = 1
    switch to Slow Start
If three dup. ACKs ("mild congestion"):
    switch to Fast Recovery



TU Delft

63

63

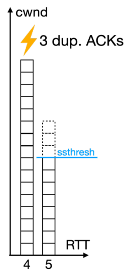
TCP Reno: Fast Recovery

Triggered upon receiving 3 dup. ACKs.

Fast Retransmit: retransmit the lost packet
Set *cwnd* = *ssthresh* + 3 (keep up the ACK clocking by sending three new packets)
(Multiplicative Decrease)

Upon receiving another dup. ACK:
cwnd = *cwnd* + 1 (keep up the ACK clocking)

Upon receiving a new ACK:
cwnd = *ssthresh*
switch to Congestion Avoidance

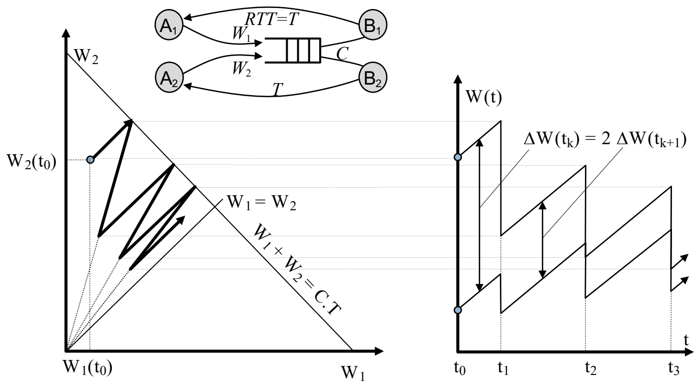


TU Delft

64

64

Add. Increase – Mult. Decrease



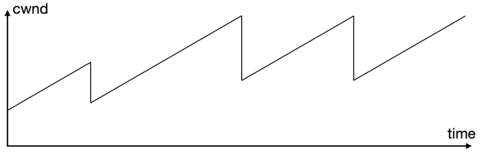
TU Delft

65

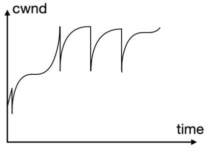
65

TCP Cubic

- Problem with Reno: linear Add. Increase is too slow in networks with high bandwidth and/or long RTTs



- Cubic: cubic increase function



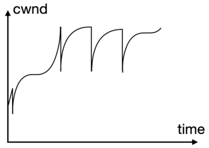
TU Delft

66

66

TCP Cubic

- When receiving ACK in Congestion Avoidance: increase *cwnd* according to a monotonic cubic function
  - Reno: linear increase
- Function is based on previous congestion event
  - Inflection point is around where the algorithm estimates congestion to occur again
- Upon 3 dup. ACKs:
  - *ssthresh* = max{0.7 * *cwnd*, 2)
  - *cwnd* = 0.7 * *cwnd*
- Upon RTO:
  - *ssthresh* = 0.7 * *cwnd*
  - *cwnd* = 1, switch to Slow Start
- Currently: Cubic is the default CCA in Linux, Windows, and MacOS and the most commonly used in the Internet



TU Delft

67

67

BBR

- Novel congestion control scheme by Google
- Motivation: problems with loss-based CCAs (e.g., Cubic)
  - When buffers are small: low throughput (high loss, CCA cuts cwnd too often)
  - When buffers are large: high queueing delay (bufferbloat)
- BBR: Bottleneck Bandwidth and Round-trip propagation time
- Idea: adjust behavior based on a model of the network path

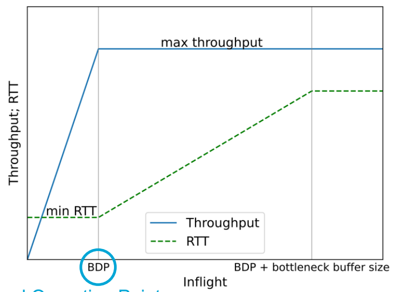
TU Delft

68

68

BBR

- Optimal operating point: 1 Bandwidth-Delay Product (BDP) of data is inflight (i.e., inside the network)



Optimal Operating Point

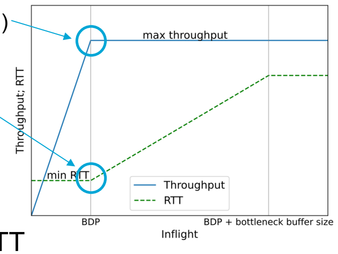
TU Delft

69

69

BBR

- Model the network path: measure
  - Bottleneck bandwidth (BtlBw)
  - Round-trip time (RTT)



- Estimated BDP = BtlBw * RTT
- Keep *inflight* around 1 estimated BDP

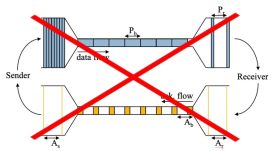
TU Delft

70

70

BBR

- Control both cwnd and pacing rate
  - cwnd: cap how much data is allowed inflight
  - pacing rate: control how fast packets are sent
- Both needed: self-clocking through ACKs no longer possible nowadays
  - DOCSIS or WiFi aggregate ACKs and send them out in a burst ("delayed ACKs")



TU Delft

71

71

BBR

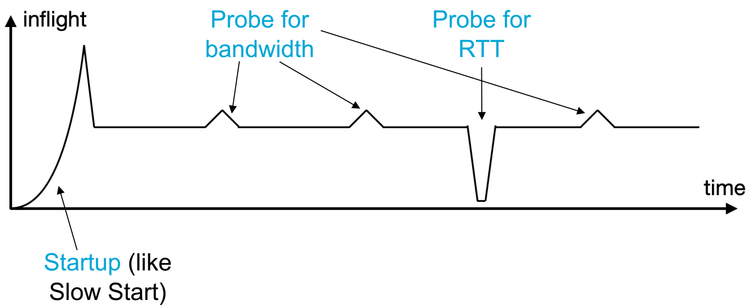
- Regularly raise sending rate to probe for bandwidth
- Regularly lower sending rate to obtain samples of the RTT

TU Delft

72

72

BBR



TU Delft

73

73

BBR

- Deployed in Google's networks
- Now second-most commonly used CCA in the Internet, adoption growing

- Problems: fairness issues, high queueing delay, no reaction to packet loss
  – Now in development: BBRv2, BBRv3

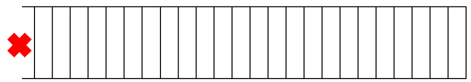
TU Delft

74

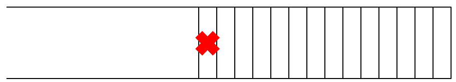
74

Active Queue Management (AQM)

- Classic buffers ("tail drop"): drop packets when buffer is full
  – High queueing delay
  – Typically drops multiple packets of the same flow



- AQM: drop packets before buffer is full and randomize which packets are dropped
  – Lower queueing delay
  – Fairer

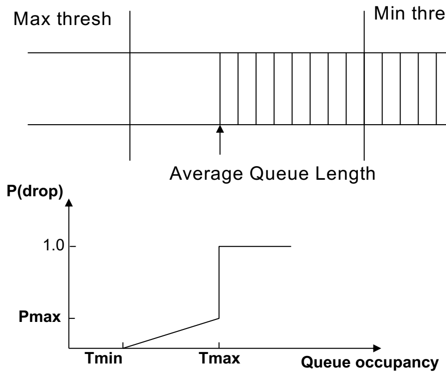


TU Delft

75

75

Random Early Detection (RED)



Drop arriving packet with probability *p*
Increase *p* with increasing queue occupancy

76

76

Controlled Delay (CoDel)

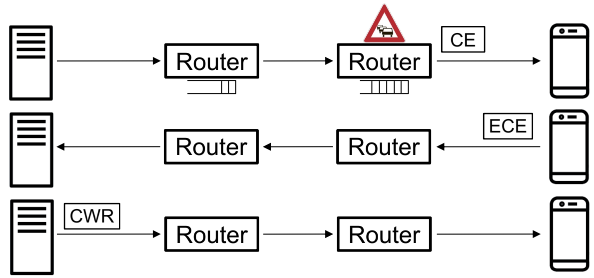
- Measure sojourn time of packets in the buffer
- Interval length initially 100 ms
- After each interval: if minimum sojourn time during interval was > 5 ms:
  - Drop a single packet
  - Shorten interval length
    - After n intervals where min. sojourn time was > 5 ms, interval length is 100 - sqrt(n) ms
- Otherwise:
  - Reset interval length to 100 ms

77

77

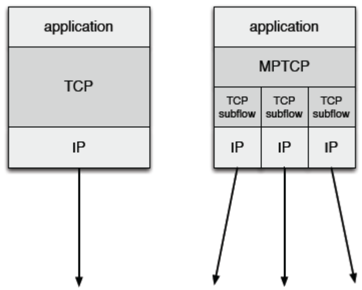
Explicit Congestion Notification (ECN)

- Routers mark packets with explicit congestion notification instead of dropping them
  - Requires AQM
- Sender reacts to ECN-Echo like it would react to packet loss



CE: Congestion Experienced
ECE: ECN-Echo
CWR: Congestion Window Reduced

MPTCP



Traditional versus Multi-Path TCP

QUIC

TU Delft

80

80

Quick UDP Internet Connections (QUIC)

- New transport protocol developed by Google
- Goals
  - Low latency
  - Security
  - Multiplexing without head-of-line blocking
  - Connection migration
  - …
- Utilizes UDP for transport
  - Firewalls or NATs sometimes block traffic that is not TCP or UDP
- But adds TCP-like features such as retransmissions
- Implements congestion control on application layer

TU Delft

81

81

QUIC

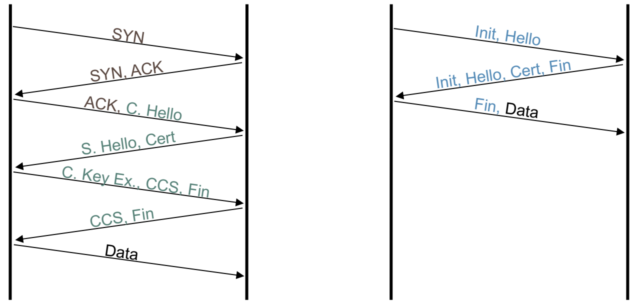
	Traditional TCP Stack	QUIC Stack
Application	HTTP/2	HTTP API
Security	TLS	QUIC
Transport	TCP	QUIC / UDP
Network	IP	IP

TU Delft

82

82

QUIC Handshake



TCP handshake +
TLS handshake +
data transmission

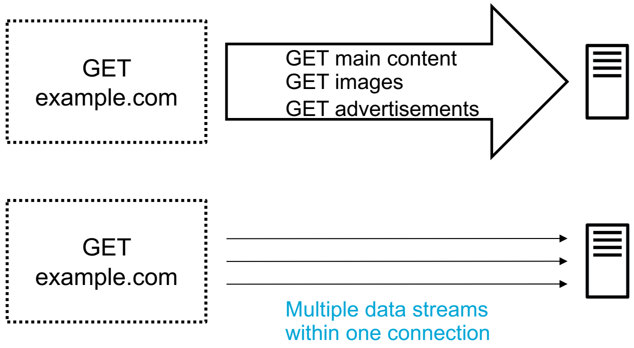
QUIC handshake +
data transmission

TU Delft

83

83

Multiplexing

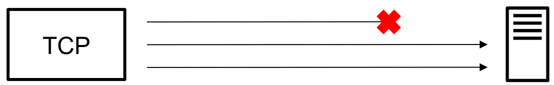


TU Delft

84

84

Head-of-Line Blocking



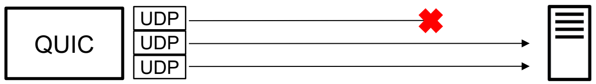
- Error in one stream (e.g. packet loss)
- TCP: blocks all streams until the error is resolved
  - Unnecessary delay!

TU Delft

85

85

QUIC Multiplexing



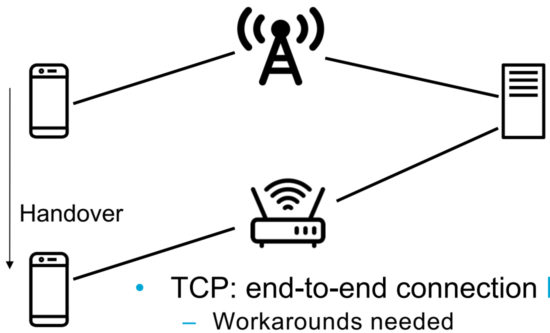
- QUIC: Error in one stream (e.g. packet loss) does not affect the other streams

TU Delft

86

86

QUIC Connection Migration



- TCP: end-to-end connection breaks
  - Workarounds needed
- QUIC: packets contain connection ID
  - Can reestablish connection fast

TU Delft

87

87