NSWI090: **Computer Networks**

http://www.ksi.mff.cuni.cz/~svoboda/courses/202-NSWI090/

Lecture 6

Transport Layer and Protocols

Martin Svoboda

svoboda@ksi.mff.cuni.cz

12. 4. 2021

Charles University, Faculty of Mathematics and Physics

Lecture Outline

Transport Layer

- **End-to-end communication**
- **Adaptation services**
  - Byte streams
  - Establishing connections
  - Reliability
    - Error detection and recovery
    - Acknowledgement schemes
  - Flow control
  - Congestion control
  - Quality of Service

End-to-End Communication

Ensuring **end-to-end** **communication**

- I.e., communication of particular application **entities** <u>within</u> the **sender / recipient nodes**
  - Lower layers (L1 – L3) always treat nodes at atomic units
    - I.e., they are unable to distinguish the individual communicating entities inside these nodes
  - L4 and higher layers are **only implemented in end nodes**
    - I.e., the highest layer implemented in routers is L3
    - And so L4 does not occur in typical network elements at all

End-to-End Communication

Ensuring **end-to-end** **communication** (cont'd)

- **Tasks** to be tackled
  - **Access points**
    - I.e., points between L4 and the higher layer (L7 in TCP/IP)
  - **Addresses** and addressing
    - **Port numbers** in TCP/IP
    - E.g., 25 (SMTP), 80 (HTTP), …
  - **Interface**: **sockets** in TCP/IP
    - Data structure allowing applications to send / receive data
    - Created on demand
    - Dynamically bound with particular ports
  - **De/multiplexing**

End-to-End Communication

De/multiplexing

- Several **concurrent communications** need to be handled
  - However, we have **only one transmission path** at L3
- **Multiplexing**
  - From the **sender** point of view…
    - Merging of several separate L4 transmissions together
- **Demultiplexing**
  - From the **recipient** point of view…
    - Sorting and processing of incoming L3 datagrams

End-to-End Communication

Identification of transport (application) **connections**

- One application entity can concurrently communicate with several remote entities at a time
  - And so we must be able to mutually distinguish between them
    - From the application point of view
- Tuple **(IP_1, $port_1$, protocol, IP_2, $port_2$)**
  - From the **sender** point of view (**outgoing** **transmission**)…
    - Intended recipient entity is identified by (IP_2, $port_2$, protocol)
  - From the **recipient** point of view (**incoming** **transmission**)…
    - Actual sender entity is identified by (IP_1, $port_1$, protocol)
- Example
  - (89.176.122.77, 55123, TCP, 195.113.20.128, 80)

Adaptation Services

Motivation

- **Lower layers** (L1 – L3)
  - Focus on transmissions themselves
- **Higher layers** (L5 – L7)
  - Focus on applications needs
- L4 forms an interface between the lower and higher layers
  - Offers various ways of **adapting** the **expectations of higher layers** to the actual **possibilities of lower layers**
    - More specifically…
    - **IP** at L3: blocks, connectionless, unreliable, Best Effort

Adaptation Services

Adaptation objectives

- **Byte streams** over blocks
- **Connection-oriented** transmissions over connectionless
- **Reliable** transmissions over unreliable
- **Quality of Service** over the Best Effort principle

Additional objectives

- **Flow control**
  - Preventing **slower recipients** to be overwhelmed by **faster senders**
- **Congestion control**
  - Preventing the **whole network** to be overwhelmed by the overall traffic generated by senders

Transport Protocols

User Datagram Protocol (UDP)

- Very simple and straightforward, minimal changes to IP
  - Blocks, connectionless, unreliable, Best Effort
  - No control flow, nor congestion control

Transmission Control Protocol (TCP)

- Very complex protocol
  - **Byte stream**, **connection-oriented**, **reliable**, Best Effort
  - **Flow control**, **congestion control**

Newer alternatives (not widely used)

- Stream Control Transmission Protocol (SCTP)
  - Connection-oriented, reliable
- Datagram Congestion Control Protocol (DCCP)
  - Connection-oriented, unreliable

Byte Streams

Providing **illusion of a byte stream** over block transmissions

- Application entity generates a **stream of bytes to be sent**
  - These bytes are provided through the socket interface
  - They are not sent immediately, only stored within a buffer
- **When the buffer is filled** (or when explicitly requested)
  - Its contents is taken and a **TCP segment is created and send**
    - Suitable size is derived in order to avoid fragmentation at L3
- Individual **segments must be numbered**
  - So that the recipient can reconstruct the sequence back again
    - Because L3 does not ensure that the segments will be delivered in the same order as they were sent
  - **Positions in a byte stream** are used for this purpose
    - Moreover, because of security reasons, they do not start at 0

Establishing Connections

Creation of a **connection-oriented** transmission

- Establishment procedure
  - **3-way handshake mechanism**
    - (1) `SYN`: initiator node A sends a connection request to node B
    - (2) `SYN-ACK`: node B sends a confirmation back to node A
    - (3) `ACK`: node A sends a final confirmation to node B
  - Only now the whole connection is considered as established
  - **Byte stream starting positions** are also negotiated
    - Proposed as random numbers
    - Mutually confirmed by both the sides
- Basic requirements
  - Whole process must be as **efficient** as possible
    - Since new connections are established on a frequent basis
  - Not many **system resources** should be needed, too

Establishing Connections

Undesirable situations are needed to be avoided

- Connection requests or confirmations may get lost
  - **Congestion**
    - Another attempt is sent too soon, may overload the other side
  - **Starvation**
    - On the contrary, waiting is unnecessarily too long
- Security aspects
  - (**Distributed**) Denial of Service **attacks** (**DoS** / **DDoS**)
    - Attempts of **overloading the target system** and so preventing some or all otherwise legitimate requests from being fulfilled
    - SYN Flooding – sending of **excessive number of `SYN` requests** without actually wanting new connections to be established
  - **Connection hijacking**
  - …

Reliability Paradigm

Reliability

- Ensuring **successful delivery of unchanged data**

Reliable transmissions

- **Errors are detected and treated** appropriately
  - Sender and recipient must mutually cooperate
- Suitable in most cases, but not always
  - Since reliability brings non-trivial overhead

Unreliable transmissions

- **Errors are not detected, nor treated** in any way
  - We may even not be aware of them
  - Transmission simply goes on
- Suitable for multimedia applications

Ensuring Reliability

Reliability **issues**

- **Losses of blocks** (or data in general)
  - Entire blocks are lost
    - I.e., blocks are not delivered to the intended recipient
- **Damage to blocks** (or data in general)
  - One or more individual isolated bits or whole clusters of bits are randomly or systematically damaged
    - I.e., replaced with the opposite ones (e.g., 0 instead of 1)

Required mechanisms

- **Detection** of lost / damaged blocks and **recovery**

Reliability Issues

Losses of blocks

- **Causes** primarily at L3
  - Calculated **routing path** is incorrect
    - Obsolete, wrong, unknown, …
  - Exceeded **time to live**
    - Packet is discarded when its hop counter is depleted
  - **Network congestion**
    - Insufficient transmission or computing capacity (Best Effort)
  - Security threats, unreliable hardware, software bugs, …
- However, also at lower layers
  - Frame is lost within a local network at L2
  - Frame is not recognized from the stream of bits at L1
  - …

Reliability Issues

Losses of blocks (cont'd)

- **Detection** mechanisms
  - Each block is in/directly assigned with an **ordinal number**
    - **Block counting** – consecutive sequence number
    - **Position marking** – position in a stream of useful data
  - When numbers of received blocks do not follow each other
    - One or more blocks are missed and so considered as lost
    - Unfortunately, blocks may not be delivered in the same order
- **Recovery** options
  - **Retransmission**
    - **Recipient requests the sender to repeat the transmission**
    - Acknowledgement mechanisms are needed

Reliability Issues

Damage to blocks

- **Causes** primarily at L1
  - Attenuation, distortion, interference, …
    - I.e., physical transmission paths are never optimal
- **Detection** mechanism
  - **Sender** calculates a certain **check value** of the block to be sent
    - Can be based on header and / or body
    - The calculated value is attached to the block and sent as well
  - **Recipient** calculates the **check value** over the received block
    - The same parts of the data are involved
  - Both the values are **mutually compared**
    - When they are identical, everything is ok
    - Otherwise, there is one or even more errors

Reliability Issues

Damage to blocks (cont'd)

- Possible strategies
  - **Error Detection Codes**
    - Allow for detection only
    - E.g.: Parity Bit, Checksum, Cyclic Redundancy Check (CRC), …
  - **Error Correction Codes**
    - Allow for detection and self-correction, too
    - E.g.: Hamming Code, Multidimensional Parity-Check Code, …
- Error control is <u>always relative</u> (will never work for 100%)
  - **Only a certain maximal number of errors can be detected**
    - And even a smaller number can possibly be corrected
  - Abilities of the individual codes vary greatly

Reliability Issues

Damage to blocks (cont'd)

- **Recovery** options
  - **Self-correction** (if possible)
    - Not efficient enough (requires high redundancy)
    - Used only rarely
    - I.e., when feedback is missing and retransmission is impossible
  - **Retransmission**
- Observations
  - **When retransmission is exploited**…
    - The actual number of errors, their character, as well as places of occurrence become all irrelevant
    - Simply because the entire blocks will be retransmitted anyway
  - $\Rightarrow$ **error control at the level of whole blocks** is sufficient

Parity Bit

Parity Bit Check

- **Groups of transmitted bits** are enriched with **parity bits** ensuring that the **overall number of bits 1 in a group is…**
  - … **even** / **odd** in case of the **Even Parity** / **Odd Parity**
- Possible approaches
  - **Transverse Parity**
    - Group = each individual byte (word)
  - **Longitudinal Parity**
    - Group = equally positioned bits across all bytes (words)
- Very limited capabilities
  - **Only odd numbers of errors can successfully be detected**
    - I.e., even number of errors mutually suppresses their impact
  - Combinations of both the approaches perform slightly better

Checksum

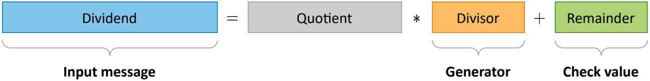
Checksum

- **Sum of individual bytes** (words) in a sequence is calculated
  - Each is treated as an unsigned integer
- The **resulting total** is used as the check value
  - **Overflow area** is discarded
    - Or alternatively added up as well
  - Recipient calculates the same total, both are tested for equality
- **Two's complement** can alternatively be used instead
  - Recipient calculates the normal sum
  - It is then summed with the received one
  - When zeros only are obtained, everything is ok
  - Otherwise an error must have occurred
- Better than parity bits, but still not efficient enough

Cyclic Redundancy Check

Cyclic Redundancy Check (**CRC**)

- **Input message** is treated as a sequence of individual bits
  - These **bits form coefficients of a polynomial in GF(2)**
    - I.e., the Galois field (**finite field**) with two elements (0 and 1)
    - Characteristic of this field is 2 (i.e., $1 + 1 = 0$)
    - All operations are evaluated using **modulo 2**
  - E.g.: $01101001 \rightarrow x^6 + x^5 + x^3 + x^0$
- **Input polynomial is divided by generator polynomial**
  - Specifically designed by a particular CRC method
  - E.g.: $x^5 + x^4 + x^2 + 1$ (order $n = 5$)
- Remainder of this polynomial division forms the **check value**



Cyclic Redundancy Check

Hardware implementation

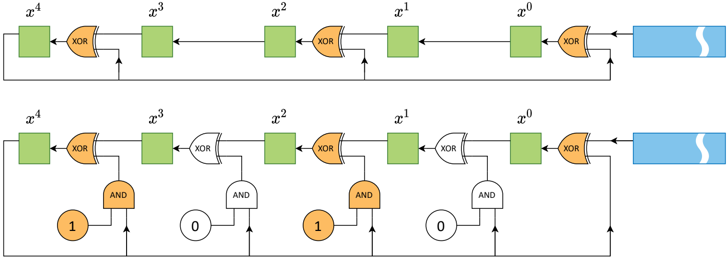
- `XOR` / `AND` gateways and shift registers are needed
- **Fixed scheme** (hardwired generator polynomial)
  - One **shift register** in a sequence is placed for each order
    - Except for the most significant one (x^n)
  - `XOR` **gateway** is put before each non-zero term
  - Output of the last register is connected with all these gateways
- **Generic scheme**
  - `XOR` **gateways** are placed before all orders
  - Additional `AND` **gateways** are used to suppress / activate them
    - Except for the lowest one (x^0)
    - Since it is assumed that it will always be non-zero

Cyclic Redundancy Check

Hardware implementation (cont'd)

- Input message is first appended with n zeros at the end
- **Input bits are pushed into the CRC circuit**, one by one
- Once finished, **registers contain the check value** (remainder)

Example for $x^5 + x^4 + x^2 + 1$ ($n = 5$)



Cyclic Redundancy Check

Verification by the recipient

- Received CRC is appended to the end of the received data
- New CRC is calculated as usual
    - When zeros only are obtained, everything is ok

Real-world **examples** (dozens of alternatives exist)

- **CRC-8**
    - $x^8 + x^7 + x^6 + x^4 + x^2 + 1$
- **CRC-32**
    - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- …
- CRC number determines the generator polynomial degree
    - And so the fixed-size of the check value

Cyclic Redundancy Check

Observations

- Built on strong theoretical results from algebra
  - Yet particularly easy to implement in hardware
  - Becomes useful at L2
- **Detection capabilities are excellent** (e.g., CRC-32)
  - All error clusters with an odd number of bits
  - All error clusters up to n bits
  - All error clusters with $> n$ bits with 99.99999998% probability
- **Generator polynomial** must be chosen very carefully
  - Even a small input change should have a significant impact
- However, not suitable for maliciously introduced errors

Acknowledgement Schemes

Error control via **retransmission**

- Recovery mechanism for **lost and damaged blocks**
  - Based on repeated transmission of the impacted blocks
  - In the expectation that problems will not occur again
    - Which may not be the case
- Necessary condition
  - Both the sender and recipient must mutually cooperate
  - I.e., particular **acknowledgement strategy** must be adopted
    - **Automatic Repeat Request** (**ARQ**)

Acknowledgement Schemes

Automatic Repeat Request (**ARQ**) (Automatic Repeat Query)

- Group of particular retransmission strategies
  - Based on **positive / negative acknowledgements** and **timeouts**
  - As well as **sequence numbers** ensuring correct block ordering
    - Or a similar mechanism
- **Individual** acknowledgement
  - **Stop-and-Wait ARQ**
- **Continuous** acknowledgement
  - **Go-Back-N ARQ**
  - **Selective Repeat ARQ**

Stop-and-Wait ARQ

Individual **acknowledgement**: Stop-and-Wait **ARQ**

- **Sender**…
  - (1) **sends** **one** **block** and **starts waiting**
  - (3) when an acknowledgement is received (if any)
    - If it is **negative**, the same block is sent once again
    - If it is **positive**, the next block can be sent
  - (4) when timeout **elapses** without any acknowledgement
    - Not knowing what was actually lost (whether the original block or the acknowledgement), the same block is sent once again
- **Recipient**…
  - (2) **receives this block** (if at all) and verifies its check value
    - If no error is detected, positive **acknowledgement** (`ACK`) is sent
    - Otherwise, negative **acknowledgement** (`NACK`) is sent
  - (5) repeatedly received duplicate must also be acknowledged
    - So that the sender will not resend the same block indefinitely

Stop-and-Wait ARQ

Observations

- **Timeout period**
  - Should not be too short nor too long
    - Techniques for defining reasonable timeouts can be elaborate
    - Yet they only **affect efficiency**, not functionality as such
- Straightforward and **easy to implement**
  - Causes the communication to become **half-duplex**
- **Unusable in larger networks**
  - Simply because of higher **latency** / Round Trip Time (**RTT**)
    - 10 Mb/s Ethernet: propagation delay $\approx 25\mu s$, efficiency $\approx 90\%$
    - Wi-Fi: propagation delay $\approx 50ms$, efficiency $\approx 2\%$
  - In other words, **only suitable for local networks**
    - Especially wired ones

Continuous Acknowledgement

Continuous **acknowledgement**

- **Blocks are sent continuously**, one by one
  - Acknowledgements are received and processed later on
    - I.e., we are not waiting for them
  - Timeout runs for each of the blocks separately
- The only question is how unsuccessful deliveries are handled
  - I.e., explicit **negative acknowledgements** / **elapsed timeouts**
    - Since several other blocks could already have been sent meanwhile, i.e., after the impacted one

Two possible strategies

- Go-Back-N **ARQ**
- Selective Repeat **ARQ**

Continuous Acknowledgement

Go-Back-N **ARQ**

- Whole transmission returns to the point of failure, i.e., …
  - **The impacted block is sent again**
  - **As well as all the subsequent ones**
- **Easier implementation** of the recipient
  - Since when a damaged block is received or not received at all, all subsequent blocks are intentionally discarded even when otherwise received successfully
    - Simply because we know they will be delivered once again
    - And so they do not need to be stored in a local buffer now
- As a consequence, **transmission capacity is wasted**
  - Since even successfully delivered blocks must also be sent again

Continuous Acknowledgement

Selective Repeat **ARQ**

- **Only the impacted block itself is selectively sent again**
  - And so transmission of other blocks stays unaffected and continues as if nothing actually happened
- Transmission capacity is not wasted
- However, **implementation of the recipient gets complicated**
  - Simply because **successfully received subsequent blocks** cannot yet be processed and so **must locally be buffered**

Continuous acknowledgement (both the methods)

- **How many blocks can be sent at a time?**
  - It could seem the sender is not limited in any way
    - In reality, the maximal possible rate would not be a good idea…

Sliding Window Method

Motivation for **sliding windows**

- **Sender must buffer all sent and not yet acknowledged blocks**
  - Otherwise retransmission would not be possible if needed
    - Simply because we would no longer have the actual data
- **Sender may be faster than the recipient**
  - I.e., recipient may not be able to process all incoming blocks
    - And so even successfully received blocks could be discarded
- **Network may not have sufficient capacity**
  - I.e., it may not be able to deliver all blocks that were sent
- **Space of block sequence numbers is not unlimited**
  - When depleted, sequence generator will need to be restarted
    - And so lower sequence values will start to appear
    - Which may confuse the whole acknowledgement mechanism

Sliding Window Method

Sliding windows

- **Transmit sliding window** – managed by the **sender**
  - **Contains all sent and not yet acknowledged blocks**
  - Its size limits the number of blocks that can be sent
  - Sliding behavior
    - New block can only be sent when the window is not full
    - When a positive acknowledgement is received, a given block is removed from the window
- **Receive sliding window** – managed by the **recipient**
  - **Contains all received and not yet processed blocks**
  - Its size limits the number of blocks that can be received
  - Sliding behavior
    - Successfully received block can only be accepted when the window is not full
    - When a block is processed, it is removed from the window

Sliding Window Method

Acknowledgement schemes revisited

- All the so far discussed methods can be seen just as special cases of the generic sliding window approach
  - I.e., they only **differ in sizes of windows they presume**
- In particular, …
  - **Individual**: **Stop-and-Wait ARQ**
    - Transmit window = 1, receive window = 1
  - **Continuous**: **Go-Back-N ARQ**
    - Transmit window = N, receive window = 1
  - **Continuous**: **Selective Repeat ARQ**
    - Transmit window = N, receive window = N
    - In fact, both the windows may have different sizes

Sliding Window Method

What are the optimal window sizes?

- Given as a trade-off between both the sender and recipient
  - Sender may try to adapt to the current situation
  - Recipient may declare its current capabilities
- Moreover, **sizes may change** during the communication

Additional observations

- **Not every block needs to be acknowledged immediately**
  - At least under the condition that sooner or later it will eventually be acknowledged
    - E.g., TCP normally acknowledges only every second segment

Flow Control

Flow control

- Making sure that **slower recipients** cannot potentially be overwhelmed by **faster senders**

Solution **principle**

- Sender takes into account **recipient capacity possibilities**
  - Which means that the sender must advertise these possibilities

Example

- **TCP** at L4
  - Usage of the **sliding window** method
    - I.e., recipient co-determines the maximum size of the sliding window by declaring the amount of data it is willing to receive

Congestion Control

Congestion control

- Attempting to prevent the **whole network** to be overwhelmed by the overall traffic generated by all senders
  - I.e., dealing with the **insufficient network capacity**
    - In terms of capacity of individual transmission paths
    - And computing capacity of individual network elements

Possible solutions

- **Feedback techniques**
  - We are attempting to respond to various congestion symptoms
- **Forward techniques**
  - We are proactively attempting to influence what is actually sent to the network

Congestion Control

Feedback techniques

- **ICMP** at `L3`
  - *Source Quench* message – not widely used, though
- **TCP** at `L4`
  - Usage of the **sliding window** method
    - When the acknowledgement is not received within the timeout, it is interpreted as potential network congestion
  - Slow start
    - Sender switches to the individual acknowledgement scheme (window size 1) and gradually increases the window size

Forward techniques (traffic conditioning)

- Traffic shaping: excessive traffic is **delayed**
- Traffic policing: excessive traffic is **discarded**

Guarantee Paradigm

Guaranteed transmission

- **Sufficient resources** are available for the whole transmission
  - In terms of **computing and transmission capacity**
- Works with **exclusive** capacity
  - Cannot be used by anyone else

Non-guaranteed transmission

- It may happen that sufficient resources will not be available
- Works with **shared** capacity
  - Cheaper, more efficient and flexible
- **Best Effort** principle
  - Maximum effort, but uncertain outcome
    - Packet loss may become inevitable

Quality of Service

Quality of Service

- In general, anything else when compared to Best Effort
- Desirable especially for **multimedia services**
  - Both interactive / non-interactive, audio / video
  - Reliability is not essential, **low jitter and latency** is essential

Possible strategies preserving the Best Effort principle

- **Capacity oversizing**
  - Intentional increasing of the available capacity
    - Deploying faster transmission paths, more powerful routers, …
  - Decreases the probability of network congestion
  - Cheap, simple, **the most common solution in practice**
- **Client buffering** – intentional delay balancing uneven latency

Pure **Relative** / **Absolute** **Quality of Service** solutions

Quality of Service

Relative QoS

- Based on the **prioritization** principle
  - **Better conditions are provided for certain kinds of data**
- When sufficient resources are no longer available…
  - Excessive packets are started to be treated differently
    - I.e., delayed / discarded based on these priorities

Differentiated Services (DiffServ)

- Several **classes of priorities** are introduced
  - Each IP packet contains this priority information
    - Forgotten *Type of Service* header field is used for this purpose
- Support of all the routers on the way is essential
  - Even a single non-cooperating router would breach the effect

Quality of Service

Absolute QoS

- Based on the **reservation** principle
  - **Required resources must be defined and reserved in advance**
    - When not attainable, request must be rejected

Integrated Services (IntServ)

- **Part of the available L3 capacity is detached**
  - So that it can only be used **solely for QoS transmissions**
  - The remaining part still follows the Best Effort principle
- **Resource Reservation Protocol (RSVP)**
  - Allows to traverse all the routers on the way
    - So that conditions can be **negotiated** and resources **reserved**
    - Based on the requirements provided by application entities
- Once again, all routers on the way must be willing to cooperate

Lecture Conclusion

End-to-end communication

- Ports, sockets, de/multiplexing, transport connections

Adaptation services

- **Byte streams**
- **Establishing connections**
- **Reliability**: losses of blocks, damage to blocks
  - Parity bit, Checksum, CRC
  - Stop-and-Wait ARQ, Go-Back-N ARQ, Selective Repeat ARQ
  - **Sliding window** method
- Flow control, congestion control
- **Quality of Service**
  - Relative DiffServ, absolute IntServ