# Middleware and Web Services

# **Lecture 2: Application Protocols**

#### doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • http://vitvar.com



Czech Technical University in Prague
Faculty of Information Technologies • Software and Web Engineering • http://vitvar.com/courses/mdw





Modified: Thu Nov 23 2017, 11:19:27 Humla v0.3

## **Overview**

- Introduction to Application Protocols
  - Synchronous and Asynchronous Communication
  - Selected Networking Concepts
- Introduction to HTTP

# **Application Protocols**

Remember this

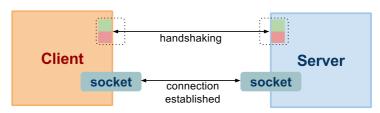
	All	People	Seem	То	Need	Data	Processing
OSI Model	Application	Presentation	Session	Transport	Network	Data Link	Physical
TCP/IP (services)	Application HTTP, XML-RPC, SOAP, RMI			Transport TCP	Network IP	Data Link	Physical

- App protocols mostly on top of the TCP Layer
  - use TCP socket for communication
- Major protocols
  - HTTP most of the app protocols layered on HTTP
    - → wide spread, but: implementors often break HTTP semantics
  - RMI Remote Method Invocation
    - $\rightarrow$  Java-specific, rather interface
    - → may use HTTP underneath (among other things)
  - XML-RPC Remote Procedure Call and SOAP
    - → Again, HTTP underneath
  - WebSocket new protocol part of HTML5

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 3 -

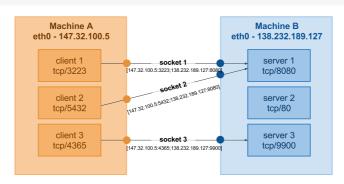
## Socket



- Handshaking (connection establishment)
  - The server listens at [dst ip,dsp port]
  - Three-way handshake:
    - → the client at [src\_ip,src\_port] sends a connection request
    - $\rightarrow$  the server responds
    - → the client acknowledges the response, can send data along
  - Result is a socket (virtual communication channel) with unique identification: socket=[src\_ip,src\_port;dst\_ip,dst\_port]
- Data transfer (resource usage)
  - Client/server writes/reads data to/from the socket
  - TCP features: reliable delivery, correct order of packets, flow control
- Connection close

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

# **Addressing in Application Protocol**

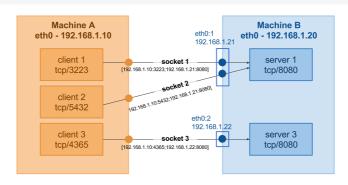


- IP addressing: IP is an address of a machine interface
  - A machine can have multiple interfaces (eth0, eth1, bond0, ...)
- TCP addressing: TCP port is an address of an app running on a machine and listening on a machine interface
  - Multiple applications with different TCP ports may listen on a machine interface
- Application addressing
  - Additional mechanisms to address entities within an application
  - They are out of scope of IP/TCP, they are app specific
    - $\rightarrow$  for example, Web apps served by a single Web server

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 5 -

### Virtual IP



- Virtual IP
  - Additional IP addresses assigned to a network interface
    - $\rightarrow$  For example, eth0 eth0:1, eth0:2, eth0:3, ...
    - $\rightarrow$  A process can bind to the virtual IP
    - → Multiple processes can listen on the same tcp port but on different virtual IPs
- Benefits
  - Floating IP a process can move transparently to another physical machine
  - Network configuration can be preserved, no need to reconfigure
  - Failover concept uses floating IPs

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

# **Virtual IP Configuration**

- Steps to configure virtual IP in Linux (example for eth0)
  - 1. Find out the interface's network mask

- 2. Create virtual IP using ifconfig
  - it should use the same network mask
  - it should be free, usually allocated to be used as a virtual IP

- 3. Update neighbours' ARP (Address Resolution Protocol) caches
  - to associate the virtual IP with MAC address of eth0
  - when the virtual IP was in use on other node or interface
  - 9 | \$ sudo arping -q -U -c 3 -I eht0 172.16.169.184



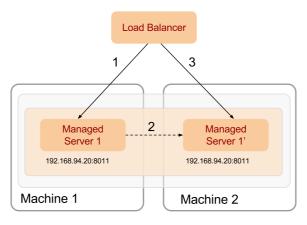
- Configure a virtual IP on your computer and test it using ping

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

-7-

### Server Failover

- Failover
  - Failover = ability to relocate the server to another machine without an impact on the performance



- Managed server listens on virtual\_IP:port
- A load balancer forwards a request to virtual IP:port
- When the server moves, virtual\_IP:port remains the same

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

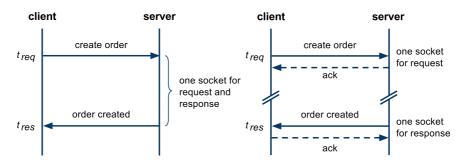
### Overview

- Introduction to Application Protocols
  - Synchronous and Asynchronous Communication
  - Selected Networking Concepts
- Introduction to HTTP

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 9 -

# **Synchronous and Asynchronous Communication**



## Synchronous

- one socket,  $|t_{req} t_{res}|$  is small
- easy to implement and deploy, only standard firewall config
- only the server defines endpoint

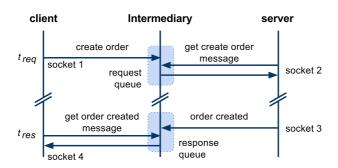
## Asynchronous

- request, response each has socket, client and server define endpoints
- $-|t_{reg}-t_{res}|$  can be large (hours, even days)
- harder to do across network elements (private/public networks issue)

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

– 10 –

# **Asynchronous via Intermediary**



### Intermediary

- A component that decouples a client-server communication
- It increases reliability and performance
  - $\rightarrow$  The server may not be available when a client sends a request
  - → There can be multiple servers that can handle the request

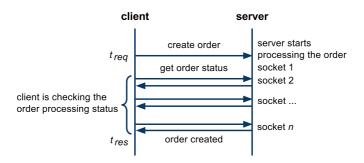
### • Further Concepts

- Message Queues (MQ) queue-based communication
- Publish/Subscribe (P/S) event-driven communication

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 11 -

# **Asynchronous via Polling**



### • Polling – only clients open sockets

- A client performs multiple request-response interactions
  - $\rightarrow$  The first interaction initiates a process on the server
  - → Subsequent interactions check for the processing status
  - $\rightarrow$  The last interaction retrieves the processing result

### Properties of environments

- A server cannot open a socket with the client (network restrictions)
- Typically on the Web (a client runs in a browser)

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

– 12 -

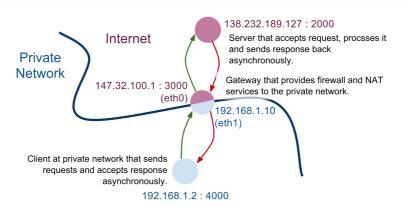
### **Overview**

- Introduction to Application Protocols
  - Synchronous and Asynchronous Communication
  - Selected Networking Concepts
- Introduction to HTTP

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 13 -

# **Public/Private Network Configuration**



- Adds complexity to configuration of application
  - Config example at server with eth0 = 147.32.100.1 (iptables)

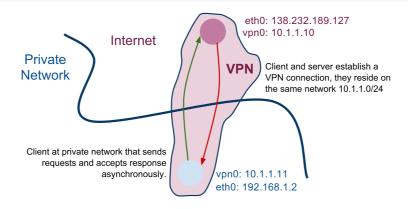
```
# enable ip forwarding from one interface to another within linux core
echo 1 > /proc/sys/net/ipv4/ip_forward

# redirect all communication coming to tcp/3000 to 192.168.1.2:4000
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 3000 -j DNAT \
--to-dest 192.168.1.2 --to-port 4000
```

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

– 14 -

### Virtual Private Network



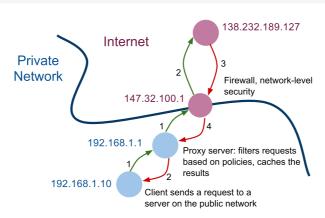
#### • VPN = Virtual Private Network

- an overlay network between a client and a server
- the network spans accross underlying network elements
- Example:
  - → VPN client starts a VPN connection with the VPN server via network interfaces
  - → VPN server assigns an IP address to the VPN client from the server's subnet
  - → Packets in VPN communication are encrypted and sent out in an outer VPN

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 15 -

# **Forward Proxy**



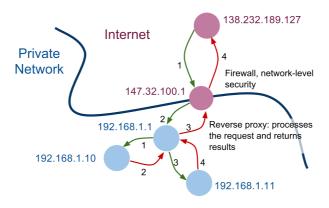
#### Forward Proxy

- Centralized access control based on content
- The client knows about the site it wants to access
- Perfoms request on behalf of the client
  - → Caches content to increase performance, limits network traffic
  - → Filters requests or controls access based on destinations or origins
- Widely used in private networks in companies
- Most of the proxy servers today are Web proxy servers

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

– 16 -

# **Reverse Proxy**



- Reverse Proxy
  - Aggregates multiple request-response interactions with back-end systems
  - Processes the request on behalf of the client
  - The client does not know about the back-end systems
  - May provide additional capabilities
    - $\rightarrow$  Data transformations
    - $\rightarrow$  Security authentication, authorization
    - ightarrow Orchestration of communication with back-end systems

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 17 -

## **Overview**

- Introduction to Application Protocols
- Introduction to HTTP
  - State Management

## **Hypertext Transfer Protocol – HTTP**

- Application protocol, basis of Web architecture
  - Part of HTTP, URI, and HTML family
  - Request-response protocol
- One socket for single request-response
  - original specification
  - have changed due to performance issues
    - → many concurrent requests
    - → overhead when establishing same connections
    - → HTTP 1.1 offers persistent connection and pipelining
- HTTP is stateless
  - Multiple HTTP requests cannot be normally related at the server
    - → "problems" with state management
    - → REST goes back to the original HTTP idea

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

**- 19** -

## **HTTP Request and Response**

Request Syntax

```
method uri http-version <crlf>
(header : value <crlf>)*
<crlf>
[ data ]
```

Response Syntax

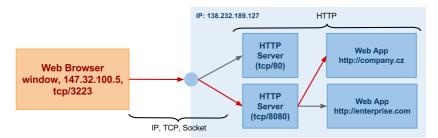
```
http-version response-code [ message ] <crlf>
(header : value <crlf>)*
<crlf>
[ data ]
```

• Semantics of terms

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

– 20 -

# **Serving HTTP Request**



- IP and TCP addressing
  - 1. User enters URL http://company.cz:8080/orders to the browser
  - 2. Browser gets an IP address for company.cz, IP:138.232.189.127
  - 3. Browser and Web Server creates a socket [147.32.100.5:3223;138.232.189.127:8080]
- Application addressing
  - 4. Browser sends HTTP request, that is, writes following data to the socket
    - 1 | GET /orders HTTP/1.1 2 | Host: company.cz
  - 5. Web server passes the request to the web application company.cz which serves GET orders and that writes a response back to the socket.

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 21 -

### **HTTP Listener**

- HTTP listener implementation in Java using Jetty <a>8</a>
  - Server listens on port 8080
  - *Jetty parses HTTP request data into* HttpServletRequest *object*.
  - When a client connects, the method handleRequest is called
  - The method tests the value of the host header and responds back if the header matches company.cz value.

```
/** handles the request when client connects **/
     public void handleRequest(HttpServletRequest request,
               HttpServletResponse response) throws IOException, ServletException {
4
          // test if the host is company.cz
5
          if (request.getHeader("Host").equals("company.cz")) {
              response.setStatus(200);
response.setHeader("Content-Type", "text/plain");
response.getWrite(".write("This is the response");
8
9
10
               response.flushBuffer();
11
12
               response.sendError(400); // bad request
    }
13
```

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

# **HTTP Listener (Cont.)**

• Test it using Telnet

```
telnet 127.0.0.1 8080
# ...lines omitted due to brevity
GET /orders HTTP/1.1
Host: company.cz

HTTP/1.1 201 OK
Content-Type: plain/text

This is the response...
```

- HTTP listener in bash
  - Use it to test incomming HTTP connections quickly
  - Uses nc utility (netcat)

```
# ctrl-c to stop http listener
control_c() {
    echo -en "\n* Exiting\n"
    exit $?
}
trap control_c SIGINT

for ((;;))
do
    echo -e "\n\n* Listening on port $1..."
    echo -e "\nHTTP/1.0 204 No Content\n\n" | nc -l $port
done
```

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 23 -

### Virtual Web Server

- Virtual server
  - Configuration of a named virtual web server
  - Web server uses host request header to distinguish among multiple virtual web servers on a single physical host.
- Apache virtual Web server configuration
  - Two virtual servers hosted on a single physical host

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

# **Better Support for HTTP Testing**

Use curl to test HTTP protocol

Example

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 25 -

## **Overview**

- Introduction to Application Protocols
- Introduction to HTTP
  - State Management

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

– 26 -

# **State Management**

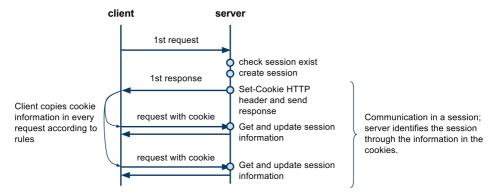
- HTTP is a stateless protocol original design
  - No information to relate multiple interactions at server-side
    - $\rightarrow$  Except Authorization header is copied in every request
    - → IP addresses do not work, one public IP can be shared by multiple clients
- Solutions to check for a valid state at server-side
  - Cookies obvious and the most common workaround
    - → RFC 2109 HTTP State Management Mechanism &
    - $\rightarrow$  Allow clients and servers to talk in a context called **sessions**
  - Hypertext original HTTP design principle
    - → App states represented by resources (hypermedia), links define transitions between states
    - → Adopted by the REST principle **statelessness**

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

\_ 27 -

## **Interaction with Cookies**

- Request-response interaction with cookies
  - Session is a logical channel maintained by the server



- Stateful Server
  - Server remembers the session information in a server memory
  - Server memory is a non-persistent storage, when server restarts the memory content is lost!

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

– 28 –

### Set-Cookie and Cookie Headers

• Set-Cookie response header

```
set-cookie = "Set-Cookie:" cookie ("," cookie)*

cookie = NAME "=" VALUE (";" cookie-av)*

cookie-av = "Comment" "=" value

"Domain" "=" value

"Max-Age" "=" value

"Path" "=" value
```

- − domain − a domain for which the cookie is applied
- Max-Age number of seconds the cookie is valid
- − Path − URL path for which the cookie is applied
- Cookie request header. A client sends the cookie in a request if:
  - domain matches the origin server's fully-qualified host name
  - path matches a prefix of the request-URI
  - Max-Age has not expired

```
cookie = "Cookie:" cookie-value (";" cookie-value)*
cookie-value = NAME "=" VALUE [";" path] [";" domain]
path = "$Path" "=" value
domain = "$Domain" "=" value
```

 domain, and path are values from corresponding attributes of the Set-Cookie header

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 29 -

# **Session Management Java Class**

Manages client sessions in a server memory

```
public class Sessions<E> {
           // storage for the session data;
private Hashtable<String, E> sessions = new Hashtable<String, E>();
            /stst Returns session id based on the information in the http request stst
           public String getSessionID(HttpServletRequest request) throws Exception {
   String sid = null;
8
                // extract the session id from the cookie
if (request.getHeader("cookie") != null) {
   Pattern p = Pattern.compile(".*session-id=([a-zA-Z0-9]+).*");
   Matcher m = p.matcher(request.getHeader("cookie"));
   if (m.matches()) sid = m.group(1);
10
11
15
16
17
18
                }
                // create the session id md5 hash; use random number to generate a client-id
                19
20
21
22
23
24
                      sid = Utils.toHexString(md.digest());
                return sid;
26
           }
           public E getData(String sid) ... // returns session data from sessions object
28
           public void setData(String sid, E d) ... // sets session data to sessions object
30
```

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

# **Stateful Server Implementation**

Simple per-client counter

```
public void handleRequest(HttpServletRequest request,
               HttpServletResponse response) throws Exception {
           // get the session id
4
          String sid = sessions.getSessionID(request);
5
6
           // create the new data if none exists
           if (sessions.getData(sid) != null)
               sessions.setData(sid,
9
                    Integer.valueOf(sessions.getData(sid).intValue() + 1));
10
          else
11
               sessions.setData(sid, Integer.valueOf(1));
13
          // send the response
14
          response.setStatus(200);
          response.setJtatus(200);
response.setHeader("Set-Cookie", "session-id="+ sid + "; MaxAge=3600");
response.setHeader("Content-Type", "text/plain");
response.getWriter().write("Number of hits from you: " +
15
16
17
               sessions.getData(sid).toString());
18
19
          response.flushBuffer();
20
     }
```

## \* Task

- What happens when the server restarts?
- How do you change the code to count requests from all clients?

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

- 31 -

# **Testing**

- Testing
  - curl will require you to specify cookies in every request
  - Browser handles cookies automatically

```
# run curl for the first time
    curl -v 127.0.0.1:8080
> GET / HTTP/1.1
    > Host: 127.0.0.1:8080
6
    < HTTP/1.1 200 OK
    < Set-Cookie: session-id=3a9c3cdc5ff36434aa1ba860727ca401;max-age=3600
8
9
    Number of hits from you: 1
    # copy the cookie session-id from previous response
11
    curl -v -b session-id=3a9c3cdc5ff36434aa1ba860727ca401 127.0.0.1:8080
12
    > GET / HTTP/1.1
    > Host: 127.0.0.1:9900
15
    > Cookie: session-id=3a9c3cdc5ff36434aa1ba860727ca401
16
17
    < HTTP/1.1 200 OK
18
    < Set-Cookie: session-id=3a9c3cdc5ff36434aa1ba860727ca401;max-age=3600
    Number of hits from you: 2
```

Lecture 2: Application Protocols, CTU Winter Semester 2017/2018, @TomasVitvar

– 32 -