Notizen, Brainstorm, Ideen

Referenz https://solden.zhaw.ch/icclab/InternalWeeklyMeetings Minutes Meeting 27.03.2014

Discuss how the KT_STREAM is implemented

Make the KT_STREAM an intelligent Router

The idea is to use a router socket from zeromq to forward/distribute the incoming messages to different servers not only for load-balancing but also by different application types. Let's assume the central broker learns the protocols or preferences of the servers (it has an internal learning table) then it can easily forward the message to the according server providing the best service. Zeromq provides a lot of modules for building more complex compounds of such as the Majordomo Pattern, the Titanic Pattern or the Binary Star Pattern.

Discuss Use-Cases for the possible implementations like dynamic spawning of the servers

Since KIARA is heavily aimed at cloud infrastructure, using a dynamic approach by spawning new resources when needed is sensitive. For e.g. service providers would register to a central broker, which in turn activates the concrete service instance when a message arrives for the targeted service and forwards the message to the recipient service instance. Later it would be possible to scale the amount of instances either by measuring the message rate (blind scaling), response time (deadline-scheduling) or by workload reported by the service instances (dynamic scaling by workload). But this also brings new challenges: How does the broker indicate the surrounding service providing infrastructure to spawn new instances? This would require a back-channel to the (OpenStack-) Cloud-Controller and a way to rapidly provision new machines.

On the other hand reducing the amount under-capacity running instances provides a good possibility to effectively improve the GreenIT value. Here the usual approach to shutdown instances if the accumulated load is bearable by *n-x* instances does only consider the load percentage but not the response time aka deadline-scheduling. Since zeromq provides a message queue this enables to monitor the queue at the broker and to determine when under a certain threshold an amount *x* is able to shutdown while preserving the required response time. This scenario is interesting for near-realtime applications like VoIP or when considering the user-experience of heavily frequented websites. By providing mechanisms that guarantee the process of query cloud providers can easily buffer or absorb load peaks resulting from different reasons while preserving a certain level of user experience.

Discuss different Use-Cases or scenarios for network topologies and client/server/communication-paradigm situations

Define this first and decide what to negotiate

Discuss also possible Use-Cases for the implementation in an existing architecture

Send capabilities to the clients from the SDN controller

This is a further SDN application

Scenario definition

What endpoints do we have - sensor, server, client

Server
A KIARA enabled server able to negotiate the used protocol.

Client
A possibly KIARA enabled client with the ability to possibly negotiate the protocol used.

Sensor
A "dumb" sensor that only speaks a certain protocol and is not able to negotiate the used protocol. It just sends a given set of data with a given protocol to the server.

Aggregation of data for the link, buffering/storing and push

SDN Controller
<PHILIPP>

Multidomain, Singledomain
Singledomain specifies an environment where only one network is in use which KIARA can fully control by itself. Multidomain refers to an environment where several networks are in play where KIARA may or may not have control, this may also reflect the situation where several KIARA instances have to communicate and decide upon a suitable configuration or setup. This is especially interesting when a network in between is used without any SDN capability, it is possible to use some tunneling techniques such as GRE for bridging the two networks if available.

Identify optimization potential Layer1 to Layer6
Let's run through the network layer from bottom to top.

Layer 1 - PHY-Layer
We will skip this layer as it more or less specifies the physical encoding and without inventing new encoding techniques and therefore hardware — which in turn requires to deploy new

hardware and this disables us from re-using existing data-center equipment etc. — there's not much possible.

Layer 2 - Data Link Layer

Media Access Control

Modifying this protocol will result in breaking functionality of widely deployed devices though there exist appliances (namely from Cisco) that use different protocols than CSMA/CD.

Logical Link Control

The three most used protocols IEEE 802.3 (ethernet), 802.11 (wifi) and ATM provide little possibility to optimize without having to interfere with physical devices. Most devices (like a 802.3 switch) provide an specified behaviour to the standard and are not very programmable.

Layer 3 - Network Layer

IPv4

IPv4 headers are at least 20 bytes and maximum 60 bytes, maximum packet size is 64KB, therefore the overhead of an IPv4 header is between 100% and 0.9‰. According to http://www.caida.org/data/passive/trace_stats/ we can assume an average (mean) of 595 bytes per packet, therefore 3.3% to 8.4% overhead.
Proposals using the header checksum field have to be rejected as router on the route eventually drop these malformed packets.
Nevertheless it is possible to discard this whole protocol layer and consider a switched network hoisted by a SDN but this requires both endpoints to handle this by using a special network stack just relying on raw sockets provided by NIX-systems. This leads to an other problem that the userspace application will require full root access to access the raw socket, without further restrictions this is an immense risk.

IPv6

IPv6 headers are at least 40 bytes and at maximum of 72 bytes (assuming no AH, ESP or MobileIP RFC 6275). Due to the lack of data resulting from little IPv6 usage the resulting overhead percentage is probably around the same of IPv4 assuming the maximum payload is 65535 bytes (and no jumbo-packets are used). But since IPv6 does not allow packet fragmentation the payload is effectively limited to 1500 bytes since most links are using Ethernet.

ICMP

Since this protocol is not typically used by end-user applications it is relatively safe to skip this entire protocol. The rare use for any end-user is ping and maybe traceroute. Of course this protocol needs to be supported as it is used for error and diagnostic purposes.

IGMP

There is little sense in optimizing IGMP since this is a pure control/management protocol. But it would probably be an applicable effort to optimize the multicast traffic though. The question seems rather to be at what scale. To run an internal CDN with NFV for a few streams to a very small amount of consumer does not seem to make much sense since it requires a data flow

separation. At the edge router the multicast informations must be added anyway so this turns the edge router into a more complex appliance, in consequence the edge router throughput will degrade (talking about 10GE, 40GE or even 100GE order of magnitude) for providing multicast support.

IPsec

Skip this one since it is a extremely complex protocol.

Layer 4 - Transport Layer

TCP

UDP

MTCP

This protocol is still very young and not widely implemented nor supported.

UDP Lite

UDP Lite has an interesting concept of allowing only certain parts being checksummed, this could help to reduce the computation time used for checksum creation/verification (even though this is usually offloaded to the NIC). It unfortunately does not provide any size advantages to UDP as it has the same header size.

Micro Transport Protocol

µTP is a protocol aimed at P2P mainly used in Bittorrent applications therefore the relevance is little.

DCCP

DCCP was first implemented in the Linux Kernel 2.6.14 and has little support by other OS. Also it seems to be problematic with routers not supporting this protocol.

SCTP

Basically it transports the data as messages and is also able to multiplex several message-chunks in one packet while providing tcp-like ack-behaviour along with multi-path support (multihoming when failure occurs). Further it supports a 4-way-handshake similar to TCP-SYN-Cookies and a heartbeat feature to check which idle connections are still alive.

QUIC

Layer 5

Since this layer is rarely used and for our case not relevant it is skipped.

Layer 6

Since this layer is rarely used and for our case not relevant it is skipped.

Layer 7

FTP

FTP is a very old protocol which uses two channels: One control and one data channel. Unless used in passive mode FTP won't work behind firewalls and NAT enabled router. Furthermore this protocol lacks of encryption which is a no-go when it comes to authentication.
It has no meta-data so therefore no overhead when transferring data or chunked data. But on the other hand it is not able to reuse existing connections, has no built in gzip support, to transmit data it takes more round trips than HTTP and uses two sockets. Furthermore when the transmission of data takes longer than the TCP socket timeout for the control channel, the connection will be dropped and therefore lost.
For pure data transmission this protocol does not make a lot of sense since it also provides file system level functionality. It would be possible to transfer data by using *netcat* which relies either on TCP or UDP. Appropriately extended (a small header like filename, filesize, hash-sum etc.) this could be more effective as except for the transport protocol it does not create any overhead and would only use one connection. Plus, by using a NAT punching techniques it could technically bypass those restrictions (which is rather a security risk than anything else).

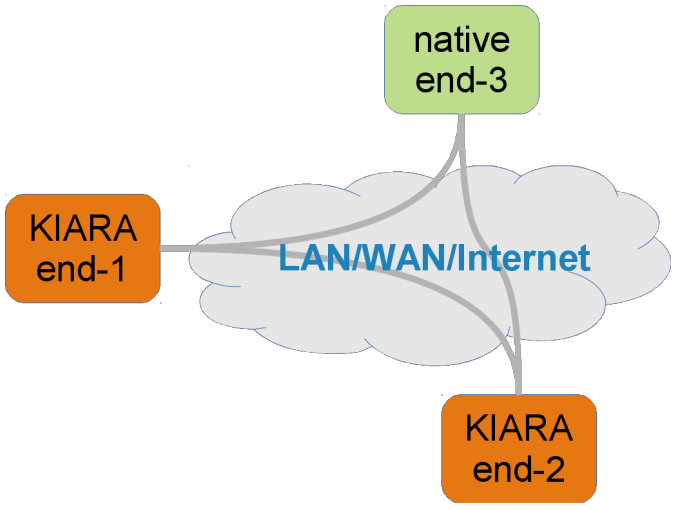
HTTP

Due to its wide prevalence in the world wide web there is no doubt this protocol must be supported. This also implicates that no modification may be done without losing substantial client support.
http://www.heise.de/netze/meldung/HTTP-2-0-Wie-sich-Webseitenabrufe-beschleunigen-lassen-2165377.html
What if we could bundle the requests to the same domain (wouldn't interfere with the same-origin-policy) and then the server can answer this single request for multiple resources in one HTTP response? We basically "multiplex" the request and response and therefore reduce the RTT by n, where n is the number of resources linked in the HTML data.

Negotiation Use-Cases and optimization parameters

In the following section we consider 3 different main use-cases with different knowledge about the network.

Scenario I: The middleware as well as the programmer have no knowledge about the network



The first scenario is the simplest one but the hardest to optimize. In this scenario, the KIARA middle does not know anything about the network and how KIARA is connected to the other endpoints. The only thing that can be detected is, if the other endpoint is as well a KIARA endpoint. If so, the two

endpoints can negotiate their local capabilities. The following parameters are valid:

MTUPathDiscovery	no dictionary values needed, local setting
Rational:	The Payload Size of the IP packet must be optimized to avoid fragmentation of the packets through the path on the network
Description:	The middleware checks, if the MTU path discovery of the operating system is set to true (/proc/sys/net/ipv4/ip_no_pmtu_disc). If the value is true, the transport stack itself sends an ICMP request and checks the error state of the response.
Responsibility:	This value is auto-set by the middleware default is OFF
OSI Layer 3	Needs no support by the network

ApplicationPayloadType	<pre>"application": {
	  "payload-type": {
	  "file-transfer": {
	  "prec": "MUST"
	  }
	  }
	},</pre>

Rational:	For real-time critical applications, the TCP segment size should be small. In contrast to large file transmission, the TCP segment size should be big. This will also affect the lower layers and decrease e.g. the ethernet frames.
Description:	Based on the result of the negotiation, the payload type is communicated by a MUST precedence to the server or the respective endpoint. The sender will then adjust the TCP payload to a optimized value.
Responsibility:	The user of the middleware set's this value, default is NOT_SET
OSI Layer 4	Needs no support by the network

ApplicationPayloadSize	<pre>"application": {
	  "payload-small-frequent": {
	  "tcp-nodelay": {
	  "prec": "MUST"
	  }
	  }
	},</pre>

Rational:	If the application is known for sending fast small requests, the transmission over TCP should be unbuffered an has with this less delay. With this option, TCP is better suited for soft real-time applications.
Description:	Based on the result of the negotiation, the TCP_NODELAY flag is set to 1 by the transport-stack.
Responsibility:	The user of the middleware set's this value, default is NOT_SET
OSI Layer 4	Needs no support by the network

TransportProtocol	`"transport": {`
	`  "transport-protocol": {`
	`  "tcp": {`
	`  "prec": "MUST"`
	`  }`
	`  }`
	`},`

Rational:	To get a real-time behaviour for error-aware applications or a reliable transmission is not needed.
Description:	The middleware uses UDP or exploits the advantages of SCTP (if available) to get a real-time behaviour of the transport. If UDP is used at all, is decided by the negotiation
Responsibility:	1. The user of the middleware set's this value 2. The transport stack automatically sets this value The default value is TCP
OSI Layer 4	● No support by the network ● Intelligent decision by the transport stack and the negotiation

DeadLine	`"qos": {`
	`  "real-time": {`
	`  "deadline": {`
	`  "value": "2"`
	`  "unit": "s"`
	`  "operator": ">"`
	`  }`
	`  }`
	`},`

Rational:	Check validity of the received data on the receiver.
Description:	The middleware adds a further attribute to a higher-level protocol like HTTP or ZMTP with a timestamp when the packet was sent. This timestamp is compared to the local time of the receiver.
Responsibility:	The user of the middleware set's this value, the default value is NOT_SET
OSI Layer 7	No support by the network needed

ActivateCompression	`"compression": {`
	`  "compression-format": {`
	`  "tgz": {`
	`  "prec": "MUST"`
	`  }`
	`  }`
	`},`

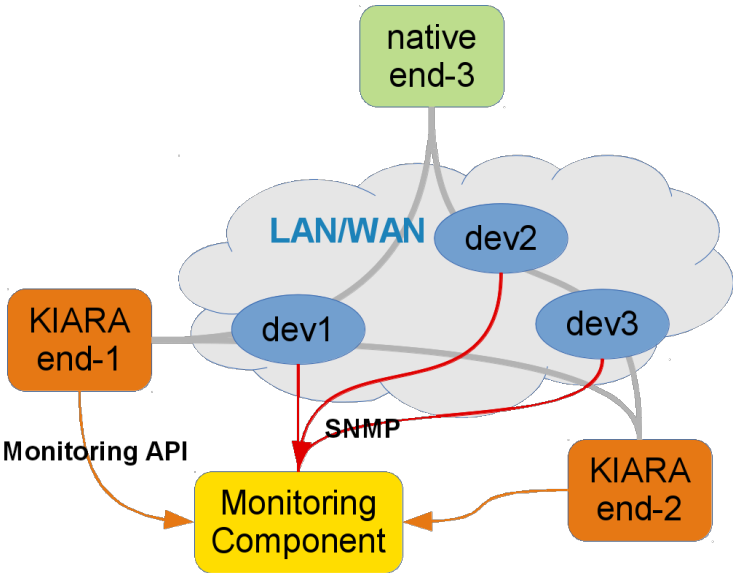
Rational:	Reduce the transmitted payload size to increase the throughput for a given bandwidth.
Description:	The middleware will compress the sended data to the given format where the format is agreed on during the negotiation process.
Responsibility:	The user of the middleware set's this value, the default value is NOT_SET
OSI Layer 6	No support by the network needed

UserProtocol	`"transport": {`
	`  "user-protocol": {`
	`  "zmtp": {`
	`  "prec": "MUST"`
	`  }`
	`  }`
	`},`

Rational:	Reduce the metadata overhead for RPC by not using HTTP.
Description:	The middleware will set, based on the result of the negotiation and the application type, a layer 7 protocol that is better suited for RPC or message based transport.

Responsibility:	The user of the middleware set's this value, the default value is HTTP
OSI Layer 7	No support by the network needed

Scenario II: The middleware or the programme do have knowledge about the network



In this scenario, the middleware is used and opperated in a self controlled network or can gather different information by a monitoring component. The monitoring component is offering by it's own API Information to the middleware on demand of the programmer of the application is aware of the network, where the middleware is used. The Informations can also be exposed by a SDN controller

Layer2	`"transport": { "link-protocol": { "ethernet-only": { "prec": "MUST" } } },`
Rational:	Reduce the overhead of higher level protocols.
Description:	The transport stack will drop completely the internet- and transport-layer. The endpoints are identified by e.g. the MAC address and the data is directly framed by the ethernet frame. This only works in a pure Layer2 routed network.
Responsibility:	The user of the middleware or the monitoring component will set this value. The default value is NOT_SET
OSI Layer 2	Support by the network needed like a monitoring component and a pure Layer2 routed network.

NetworkBoundary	<pre>"network": {
	  "ldap-group": {
	  "ou=STAFF": {
	  "prec": "MUST"
	  }
	  }
	},</pre>

Rational:	Assure that network traffic does not leave a certain physical network.
Description:	Additional Information can be fetched from a LDAP server to assure, that the connection to a certain host does not leave a specified LDAP organisation unit.
Responsibility:	The user of the middleware or the monitoring component will set this value. The default value is NOT_SET
OSI Layer 1	Support by the network needed like a monitoring component.

QoSComparison	<pre>"qos": {
	  "compare": {
	  "latency": {
	  "prec": "SHOULD",
	  "value": 2,
	  "unit": "s",
	  "operator": ">"
	  }
	  }
	},</pre>

Rational:	Assure certain QoS settings that are fixed for a given service. The application can not work under any other QoS settings.
Description:	The middleware collects information from the monitoring component and compares the median over a certain period of time with the specified value.
Responsibility:	The user of the middleware and the monitoring component will set this value. The default value is NOT_SET
OSI Layer 1	Support by the network needed like a monitoring component.

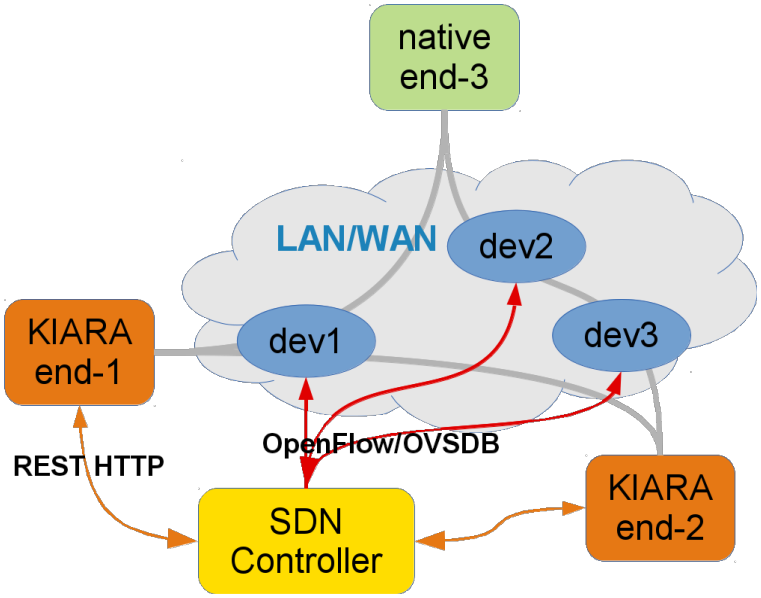
OptimizeRetransmission	no dictionary values needed, local setting

Rational:	Reduce the retransmission of packets in a network with a high packet-drop or packet-error rate.
Description:	The middleware will collect informations about packet drop rate and packet error rate. If the rates on the own path are too high, the middleware will send smaller TCP packets
Responsibility:	A monitoring system or a SDN controller with OpenFlow is needed. The default value is NOT_SET
OSI Layer 4	Support by the network needed like a monitoring component.

SCTPRedundancy	no dictionary values needed, local setting
Rational:	The make the connection to a certain endpoint more reliable.
Description:	If the middleware can use SCTP in a network, the multihomeing feature of SCTP is used to create redundant paths between the endpoints for the data channel.
Responsibility:	A monitoring system is needed. The default value is NOT_SET
OSI Layer 4	Support by the network needed like a monitoring component.

TODO: Cross-Layer optimization (kill retransmissions over radio link)

Scenario III: The middleware has access to a SDN controller



In this scenario has the middleware access to a SDN controller that exposes a API to configure the network to it's needs.

ConnectBestCell	no dictionary values needed, local setting
Rational:	Get the best available bandwidth of a multi-cell radio network.
Description:	The middleware client connects to any available cell in a radio network and is then handed over to the cell with the lowest amount of users and thus, to the cell with the best QoE. The SDN controller makes an intelligent decision based on the amount of available bandwidth and current connected users to a certain cell.
Responsibility:	A SDN controller is needed for this. If the API is detected, it will use it automatically
OSI Layer 1	Support by the network needed (SDN controller).

SetBandwidth	`"qos": { "sdn": { "bandwidth": { "prec": "SHOULD", "value": 200, "unit": "Gbps", "operator": "=" } } },`
Rational:	Bandwidth critical applications do expect a guaranteed bandwidth point-to-point.
Description:	The SDN controller will setup meter-bands along the desired path and associate the forwarding along a certain matching-criteria.
Responsibility:	A SDN controller is needed for this. The default value is NOT_SET
OSI Layer 1	Support by the network needed (SDN controller).

SetMultiPath	`"qos": { "sdn": { "multipath": { "prec": "SHOULD" } }`

	},
Rational:	Critical endpoints need a failover path for high reliability.
Description:	The SDN controller will setup, if possible, failover paths betwee two endpoints where the first match is activated
Responsibility:	A SDN controller is needed for this. The default value is NOT_SET
OSI Layer 1	Support by the network needed (SDN controller).

SetQoSLimit	<pre>"qos": {
	  "sdn": {
	  "latency": {
	  "prec": "SHOULD",
	  "value": 3,
	  "unit": "s",
	  "operator": "<"
	  }
	  }
	},</pre>

Rational:	Provide quick information about latency over certain links, and later also a possible heat-map where the network is struggling to keep up or where is room for improvements.
Description:	Every KIARA node is also doing measurements on the network and will send the result to the SDN Controller who collects all this
Responsibility:	A SDN controller is needed for this. The default value is NOT_SET
OSI Layer 1	Support by the network needed (SDN controller)

TCPOptimization	no dictionary values needed, local setting
Rational:	Optimize the TCP algorithm based on cross layer information provided by the SDN controller via TCP field matches and TCP option fields.
Description:	Introduce a new OpenFlow match on certain TCP header fields of the 3-way handshake. The modified KIARA TCP stack can

	based on the newly set values choose a better algorithm or star with modified values.
Responsibility:	A SDN controller is needed for this. The default is not used
OSI Layer 1	Support by the network needed (SDN controller)

Further QoS parameters not related to networking

Appendix

What can be optimized if we don't know anything about the network

Layer 2 (Ethernet)

- Rational: Reduce the workload for client and server
- Howto: Skip everything above layer 2 if we are in the same broadcast domain
    - Client knows the destination ethernet address of the server and communicates with raw-sockets with the server.
    - This is different from a switched network, because TCP/IP is not interpreted by server and client and the network does not interpret this like ARP broadcasts.

Layer 3 (IP)

- Rational: Optimize the payload-size to avoid fragmentation
- Howto: If MTU discovery is disabled, (based on the value of /proc/sys/net/ipv4/ip_no_pmtu_disc) the MTU path discovery is made by the transport stack itself.

Layer 4 (most often TCP)

- Rational: For real-time critical applications, the TCP segment size should be smaller. In contrast, large file transmission, the TCP segment size should be bigger. This will also affect the lower layers and decrease e.g. the ethernet frames.
- Howto: Change the size of TCP segment size out of the transport stack depending on the application needs (Applications needs are defined through the negotiation)
- Rational: For real-time critical applications, packets should be send fast without buffering
- Howto: Change the value of TCP_NODELAY to 1 to disable the TCP-nagle-algorithm based on the transport negotiation or the need of the application.
- Rational: Use the advantages of SCTP over TCP (multihoming, multistreaming) to get a better transport behaviour

- Howto: Negotiate the capabilities if the endpoints do support SCTP with the transport negotiation protocol
- Rational: To get a real-time behaviour for error-aware applications or non reliable transmission
- Howto: Use non reliable protocols like UDP or as such configured SCTP
- Rational: real-time behaviour by using a deadline on the receiver side
- Howto: Use a timestamp in the payload and check if the timestamp is older than the negotiated deadline

Layer 6

- Rational: Reduce the transmitted payload size to increase the throughput for a given bandwidth
- Howto: Use compression for the payload as LZW, GZIP, DEFLATE

Layer 7

- Rational: Reduce the metadata overhead for RPC by not using HTTP
- Howto: Use ZMTP instead of HTTP

What can be optimized if we do know about the network but cannot influence it

Layer 2

- No need to do specific things here

Layer 3

- Rational: Assure that network traffic does not leave a certain physical network
- Howto: Check the topology along with the OU defined in the LDAP dictionary
- Rational: Assure certain QoS settings that are fixed for a given service, the application can not work under any other QoS settings
- Howto: Check the median for the 4 major QoS settings along a given network path and do not setup the connection, if any median is poorer than the allowed one

Layer 4

- Rational: Reduce the retransmission of packets in a network with a high packet-drop or packet-error rate
- Howto: Send packets with smaller TCP payload

General

- Metrics (latency, jitter, RTT, drops etc.) about the network known, how can we react accordingly?
- Schengen-Routing, if packet goes via USA, then NSA. Anyway NSA. (Layer 3)
- National-Routing, if packet is leaving country borders, then we don't send data
- Paranoia-Routing, only encrypted links are allowed, depends if we are in LAN or go over WAN (Layer 3)
- If we are on a radio link we can increase the timeout of connections to smoother the fast-fading effects (Layer 1-4)

- If we are on a radio link connected to an over-saturated tower, we can handover to a less saturated tower even though the link quality is worse (Layer 1)
- If hop count is over a given limit we don't connect (Layer 3)

RTT/Latency

- Packet size influences buffering latency (only for hard-real-time systems relevant) (Layer 7, relevant for application -> see VoIP)
- Optimum packet/frame size from the beginning (Layer 2-3)
- Reduce the MTU to prevent fragmentation at gateways/routers if links have a smaller MTU, this reduces the latency introduced by fragmenting the frames/packets.
- IPv4 ECN support (RFC 3168) (Layer 3)
- If QoS parameter is out of requested range we don't initialize a connection (needs monitoring tool to read SNMP values → Ceilometer)
- GeoIP, choose closest server, optimize latency

Bandwidth/Throughput

- Use protocols that don't do CRCs or do forward-correction if a lot of bit-flipping occurs
- Adjust timeouts if on a LFN resp. window sizes

What can be optimized if we have full control over the network (SDN)

Layer 1

- Rational: Get the best available bandwidth of a multi-cell radio network
- Howto: The client connects to the first available access point, send a list to the negotiator of visible access points along with the link quality, receive a prioritized list of optimal access points to connect to regarding to the requested QoS.
- Rational: To optimize the communication between two endpoints the path with the smallest congestion should be used
- Howto: The SDN controller sets up paths not via shortest hops. It sets the path via less congested network devices
- Rational: Bandwidth critical applications do expect a guaranteed bandwidth point-to-point
- Howto: The SDN controller can assure with OFP meter-bands a certain amount of bandwidth
- Rational: Latency critical applications should have a guarantee, that the RTT is not rising a certain amount of time
- Howto: KIARA endpoints do report to the SDN controller, the latency to a certain peer. The SDN controller stores the information and offer them to other endpoints who take the same path over the devices.
- Rational: Provide quick information about latency over certain links, and later also a possible heat-map where the network is struggling to keep up or where is room for improvements.
- Howto: Every KIARA node is also doing measurements on the network and will send the result to the SDN Controller who collects all this

Layer 2

- Create 'route' to target host which allows to skip higher protocol layers and use plain MAC (as if we were in a plain switched network)

Layer 3

- Rewrite IP address
- Move server closer to boundary gateway if it's a traffic intensive service (unclog the internal network from unnecessary traffic, improve QoS)

Layer 4

Layer 5

Layer 6

Layer 7