net2o: Application Layer

Factor the Content

Bernd Paysan

EuroForth 2011, Vienna

Outline

Motivation

Requirements

Solutions
- Some Basic Insights
- Factor Data
- Distribute the Code

net2o Recap: Lower–Level Parts

- Shared memory datagrams with short headers, packet size a power of two, see [1].
- Abstraction: files and attributes (name $\longrightarrow$ value pairs)
- Network model: peer to peer distributed file system with version control, “single data cloud,” see also [2]
- Real–time ability allows data streaming
- Legacy implementation based on UDP
- P2P principle: access data by cryptographic hash (Merkle trees for larger data)

net2o Recap: Lower–Level Parts

- Shared memory datagrams with short headers, packet size a power of two, see [1].
- Abstraction: files and attributes (name $\longrightarrow$ value pairs)
- Network model: peer to peer distributed file system with version control, “single data cloud,” see also [2]
- Real–time ability allows data streaming
- Legacy implementation based on UDP
- P2P principle: access data by cryptographic hash (Merkle trees for larger data)

net2o Recap: Lower–Level Parts

- Shared memory datagrams with short headers, packet size a power of two, see [1].
- Abstraction: files and attributes (name $\longrightarrow$ value pairs)
- Network model: peer to peer distributed file system with version control, “single data cloud,” see also [2]
- Real–time ability allows data streaming
- Legacy implementation based on UDP
- P2P principle: access data by cryptographic hash (Merkle trees for larger data)

net2o Recap: Lower–Level Parts

- Shared memory datagrams with short headers, packet size a power of two, see [1].
- Abstraction: files and attributes (name $\longrightarrow$ value pairs)
- Network model: peer to peer distributed file system with version control, “single data cloud,” see also [2]
- Real–time ability allows data streaming
- Legacy implementation based on UDP
- P2P principle: access data by cryptographic hash (Merkle trees for larger data)

net2o Recap: Lower–Level Parts

- Shared memory datagrams with short headers, packet size a power of two, see [1].
- Abstraction: files and attributes (name $\longrightarrow$ value pairs)
- Network model: peer to peer distributed file system with version control, “single data cloud,” see also [2]
- Real–time ability allows data streaming
- Legacy implementation based on UDP
- P2P principle: access data by cryptographic hash (Merkle trees for larger data)

net2o Recap: Lower–Level Parts

- Shared memory datagrams with short headers, packet size a power of two, see [1].
- Abstraction: files and attributes (name $\longrightarrow$ value pairs)
- Network model: peer to peer distributed file system with version control, “single data cloud,” see also [2]
- Real–time ability allows data streaming
- Legacy implementation based on UDP
- P2P principle: access data by cryptographic hash (Merkle trees for larger data)

Status: Legacy Flow Control

- TCP flow control too aggressive, fills buffers until packet drops occur — huge delays (in the order of 4s for a typical DSL account)
- Somebody did my homework: LEDBAT flow control adds constant delay — not so aggressive
- Can’t use LEDBAT in naive form, needs adaption — I like to have a bandwidth control, not a window or even packet size control
- BitTorrent already implements LEDBAT, seems to be a good idea [3]
- Assumption is single bottleneck with few shared connections

Status: Legacy Flow Control

- TCP flow control too aggressive, fills buffers until packet drops occur — huge delays (in the order of 4s for a typical DSL account)
- Somebody did my homework: LEDBAT flow control adds constant delay — not so aggressive
- Can't use LEDBAT in naive form, needs adaption — I like to have a bandwidth control, not a window or even packet size control
- BitTorrent already implements LEDBAT, seems to be a good idea [3]
- Assumption is single bottleneck with few shared connections

Status: Legacy Flow Control

- TCP flow control too aggressive, fills buffers until packet drops occur — huge delays (in the order of 4s for a typical DSL account)
- Somebody did my homework: LEDBAT flow control adds constant delay — not so aggressive
- Can’t use LEDBAT in naive form, needs adaption — I like to have a bandwidth control, not a window or even packet size control
- BitTorrent already implements LEDBAT, seems to be a good idea [3]
- Assumption is single bottleneck with few shared connections

Status: Legacy Flow Control

- TCP flow control too aggressive, fills buffers until packet drops occur — huge delays (in the order of 4s for a typical DSL account)
- Somebody did my homework: LEDBAT flow control adds constant delay — not so aggressive
- Can’t use LEDBAT in naive form, needs adaption — I like to have a bandwidth control, not a window or even packet size control
- BitTorrent already implements LEDBAT, seems to be a good idea [3]
- Assumption is single bottleneck with few shared connections

Status: Legacy Flow Control

- TCP flow control too aggressive, fills buffers until packet drops occur — huge delays (in the order of 4s for a typical DSL account)
- Somebody did my homework: LEDBAT flow control adds constant delay — not so aggressive
- Can't use LEDBAT in naive form, needs adaption — I like to have a bandwidth control, not a window or even packet size control
- BitTorrent already implements LEDBAT, seems to be a good idea [3]
- Assumption is single bottleneck with few shared connections

What for?

People like to share information (*share* means making *copies*)

- Messages, photos, videos, music
- Longer, structured documents
- Two–way real–time (chat, telephony, video conferencing)
- Collaborative gaming

What for?

People like to share information (*share* means making *copies*)

- Messages, photos, videos, music
- Longer, structured documents
- Two–way real–time (chat, telephony, video conferencing)
- Collaborative gaming

What for?

People like to share information (*share* means making *copies*)

- Messages, photos, videos, music
- Longer, structured documents
- Two–way real–time (chat, telephony, video conferencing)
- Collaborative gaming

What for?

People like to share information (*share* means making *copies*)

- Messages, photos, videos, music
- Longer, structured documents
- Two–way real–time (chat, telephony, video conferencing)
- Collaborative gaming

What for?

People like to share information (*share* means making *copies*)

- Messages, photos, videos, music
- Longer, structured documents
- Two–way real–time (chat, telephony, video conferencing)
- Collaborative gaming

How to find?

Information needs to be organized, or we are lost

- By person/group
- By topic
- By content
- By relevance (“page rank”)
- By preference (“like it”–button, visited regularly. “hate it”–button to rate annoying pages down)
- By date or geographic location

How to find?

Information needs to be organized, or we are lost

- By person/group
- By topic
- By content
- By relevance (“page rank”)
- By preference (“like it”–button, visited regularly. “hate it”–button to rate annoying pages down)
- By date or geographic location

How to find?

Information needs to be organized, or we are lost

- By person/group
- By topic
- By content
- By relevance (“page rank”)
- By preference (“like it”–button, visited regularly. “hate it”–button to rate annoying pages down)
- By date or geographic location

How to find?

Information needs to be organized, or we are lost

- By person/group
- By topic
- By content
- By relevance (“page rank”)
- By preference (“like it”–button, visited regularly. “hate it”–button to rate annoying pages down)
- By date or geographic location

How to find?

Information needs to be organized, or we are lost

- By person/group
- By topic
- By content
- By relevance ("page rank")
- By preference ("like it"–button, visited regularly. "hate it"–button to rate annoying pages down)
- By date or geographic location

How to find?

Information needs to be organized, or we are lost

- By person/group
- By topic
- By content
- By relevance (“page rank”)
- By preference (“like it”–button, visited regularly. “hate it”–button to rate annoying pages down)
- By date or geographic location

How to find?

Information needs to be organized, or we are lost

- By person/group
- By topic
- By content
- By relevance (“page rank”)
- By preference (“like it”–button, visited regularly. “hate it”–button to rate annoying pages down)
- By date or geographic location

Web Pages as Application

How to Present

Elements of a web page, classified by user experience

1. User interface — navigation, modes, and actions
2. Textual content
3. Graphics and photos
4. Videos
5. Games…

Web Pages as Application

How to Present

Elements of a web page, classified by user experience

1. User interface — navigation, modes, and actions
2. Textual content
3. Graphics and photos
4. Videos
5. Games. . .

Web Pages as Application

How to Present

Elements of a web page, classified by user experience

1. User interface — navigation, modes, and actions
2. Textual content
3. Graphics and photos
4. Videos
5. Games…

Web Pages as Application

How to Present

Elements of a web page, classified by user experience

1. User interface — navigation, modes, and actions
2. Textual content
3. Graphics and photos
4. Videos
5. Games…

Web Pages as Application

How to Present

Elements of a web page, classified by user experience

1. User interface — navigation, modes, and actions
2. Textual content
3. Graphics and photos
4. Videos
5. Games…

Web Pages as Application

How to Present

Elements of a web page, classified by user experience

1. User interface — navigation, modes, and actions
2. Textual content
3. Graphics and photos
4. Videos
5. Games. . .

Factoring 1.0

- HTML glues together all textual and user interface elements into one file
- Separated CSS for layout
- Separated JavaScript for application logic
- Separated graphics and videos
- Plugins for games and videos

Factoring 1.0

- HTML glues together all textual and user interface elements into one file
- Separated CSS for layout
- Separated JavaScript for application logic
- Separated graphics and videos
- Plugins for games and videos

Factoring 1.0

- HTML glues together all textual and user interface elements into one file
- Separated CSS for layout
- Separated JavaScript for application logic
- Separated graphics and videos
- Plugins for games and videos

Factoring 1.0

- HTML glues together all textual and user interface elements into one file
- Separated CSS for layout
- Separated JavaScript for application logic
- Separated graphics and videos
- Plugins for games and videos

Factoring 1.0

- HTML glues together all textual and user interface elements into one file
- Separated CSS for layout
- Separated JavaScript for application logic
- Separated graphics and videos
- Plugins for games and videos

Browser as Application Environment

Remember: The browser is meant as application environment for net–centric applications!

- Requires a general purpose language to write the applications in
- Requires a good development and debugging environment
- Must be fast/low power budget (performance is key on mobile devices, PCs are fast enough)
- HTML5+CSS+JavaScript doesn’t cut the mustard

Browser as Application Environment

Remember: The browser is meant as application environment for net–centric applications!

- Requires a general purpose language to write the applications in
- Requires a good development and debugging environment
- Must be fast/low power budget (performance is key on mobile devices, PCs are fast enough)
- HTML5+CSS+JavaScript doesn’t cut the mustard

Browser as Application Environment

Remember: The browser is meant as application environment for net–centric applications!

- Requires a general purpose language to write the applications in
- Requires a good development and debugging environment
- Must be fast/low power budget (performance is key on mobile devices, PCs are fast enough)
- HTML5+CSS+JavaScript doesn't cut the mustard

Browser as Application Environment

Remember: The browser is meant as application environment for net–centric applications!

- Requires a general purpose language to write the applications in
- Requires a good development and debugging environment
- Must be fast/low power budget (performance is key on mobile devices, PCs are fast enough)
- HTML5+CSS+JavaScript doesn’t cut the mustard

Browser as Application Environment

Remember: The browser is meant as application environment for net–centric applications!

- Requires a general purpose language to write the applications in
- Requires a good development and debugging environment
- Must be fast/low power budget (performance is key on mobile devices, PCs are fast enough)
- HTML5+CSS+JavaScript doesn’t cut the mustard

Factor the Text!

- Dynamic web pages use AJAX to partially replace elements of the page
- Why put everything into a single file first, and then start tearing it apart?
- Put each element (article, comment, message, navigation bar, images, videos, sound) into a file of its own, and *refer* to embedded objects, regardless of their type
- Use scripts to generate dynamic references (“the last three comments to that article”)
- Provide low–level services, and let the application logic and libraries do the rest

Factor the Text!

- Dynamic web pages use AJAX to partially replace elements of the page
- Why put everything into a single file first, and then start tearing it apart?
- Put each element (article, comment, message, navigation bar, images, videos, sound) into a file of its own, and *refer* to embedded objects, regardless of their type
- Use scripts to generate dynamic references (“the last three comments to that article”)
- Provide low–level services, and let the application logic and libraries do the rest

Factor the Text!

- Dynamic web pages use AJAX to partially replace elements of the page
- Why put everything into a single file first, and then start tearing it apart?
- Put each element (article, comment, message, navigation bar, images, videos, sound) into a file of its own, and *refer* to embedded objects, regardless of their type
- Use scripts to generate dynamic references (“the last three comments to that article”)
- Provide low–level services, and let the application logic and libraries do the rest

Factor the Text!

- Dynamic web pages use AJAX to partially replace elements of the page
- Why put everything into a single file first, and then start tearing it apart?
- Put each element (article, comment, message, navigation bar, images, videos, sound) into a file of its own, and *refer* to embedded objects, regardless of their type
- Use scripts to generate dynamic references (“the last three comments to that article”)
- Provide low–level services, and let the application logic and libraries do the rest

Factor the Text!

- Dynamic web pages use AJAX to partially replace elements of the page
- Why put everything into a single file first, and then start tearing it apart?
- Put each element (article, comment, message, navigation bar, images, videos, sound) into a file of its own, and *refer* to embedded objects, regardless of their type
- Use scripts to generate dynamic references (“the last three comments to that article”)
- Provide low–level services, and let the application logic and libraries do the rest

Downsides

- Client has to pull data together, more network traffic
- Data assembly may be pretty static
- Collated queries work in a client–server environment, but not in a P2P environment

Downsides

- Client has to pull data together, more network traffic
- Data assembly may be pretty static
- Collated queries work in a client–server environment, but not in a P2P environment

Downsides

- Client has to pull data together, more network traffic
- Data assembly may be pretty static
- Collated queries work in a client–server environment, but not in a P2P environment

ACID on Data Clouds

- You want your data to be in shape, thus you need the ACID[1] properties of the data repository
- You can't do read–modify–write on data clouds. You can push new data, that's it. Forget about "delete," there's no easy way to call your data back. You can't implement locks. All you get is durability: "the net will not forget."
- However, you can say "this is the next revision of document *x*."
- Two concurrent edits will produce a fork. It's up to the authors to decide how to merge forks back.
- When you publish something, you can't guarantee that it's available in order. If the reader gets an incomplete transaction, he must either retry or fall back to the previous version.

[1] Atomicity, Consistency, Isolation, and Durability.

ACID on Data Clouds

- You want your data to be in shape, thus you need the ACID[1] properties of the data repository
- You can't do read–modify–write on data clouds. You can push new data, that's it. Forget about "delete," there's no easy way to call your data back. You can't implement locks. All you get is durability: "the net will not forget."
- However, you can say "this is the next revision of document *x*."
- Two concurrent edits will produce a fork. It's up to the authors to decide how to merge forks back.
- When you publish something, you can't guarantee that it's available in order. If the reader gets an incomplete transaction, he must either retry or fall back to the previous version.

[1] Atomicity, Consistency, Isolation, and Durability.

ACID on Data Clouds

- You want your data to be in shape, thus you need the ACID[1] properties of the data repository
- You can't do read–modify–write on data clouds. You can push new data, that's it. Forget about "delete," there's no easy way to call your data back. You can't implement locks. All you get is durability: "the net will not forget."
- However, you can say "this is the next revision of document *x*."
- Two concurrent edits will produce a fork. It's up to the authors to decide how to merge forks back.
- When you publish something, you can't guarantee that it's available in order. If the reader gets an incomplete transaction, he must either retry or fall back to the previous version.

[1] Atomicity, Consistency, Isolation, and Durability.

ACID on Data Clouds

- You want your data to be in shape, thus you need the ACID[1] properties of the data repository
- You can't do read–modify–write on data clouds. You can push new data, that's it. Forget about "delete," there's no easy way to call your data back. You can't implement locks. All you get is durability: "the net will not forget."
- However, you can say "this is the next revision of document *x*."
- Two concurrent edits will produce a fork. It's up to the authors to decide how to merge forks back.
- When you publish something, you can't guarantee that it's available in order. If the reader gets an incomplete transaction, he must either retry or fall back to the previous version.

[1] Atomicity, Consistency, Isolation, and Durability.

ACID on Data Clouds

- You want your data to be in shape, thus you need the ACID[1] properties of the data repository
- You can't do read–modify–write on data clouds. You can push new data, that's it. Forget about "delete," there's no easy way to call your data back. You can't implement locks. All you get is durability: "the net will not forget."
- However, you can say "this is the next revision of document *x*."
- Two concurrent edits will produce a fork. It's up to the authors to decide how to merge forks back.
- When you publish something, you can't guarantee that it's available in order. If the reader gets an incomplete transaction, he must either retry or fall back to the previous version.

[1] Atomicity, Consistency, Isolation, and Durability.

Hash–Indexed Content

Hashes as “handle” to actual content are the key to data management

- $h(data) \longrightarrow hash$ produces a unique hash for each data file
- $d(hash) \longrightarrow data$ allows to retrieve the data when the hash is known
- *Hash trees* provide a mean to distribute large files
- *Relationships* between data revisions are stored as *graph*, using the hashes as symbol for the actual data

Hash–Indexed Content

Hashes as “handle” to actual content are the key to data management

- $h(data) \longrightarrow hash$ produces a unique hash for each data file
- $d(hash) \longrightarrow data$ allows to retrieve the data when the hash is known
- *Hash trees* provide a mean to distribute large files
- *Relationships* between data revisions are stored as *graph*, using the hashes as symbol for the actual data

Hash–Indexed Content

Hashes as “handle” to actual content are the key to data management

- $h(\mathit{data}) \longrightarrow \mathit{hash}$ produces a unique hash for each data file
- $d(\mathit{hash}) \longrightarrow \mathit{data}$ allows to retrieve the data when the hash is known
- *Hash trees* provide a mean to distribute large files
- *Relationships* between data revisions are stored as *graph*, using the hashes as symbol for the actual data

Hash–Indexed Content

Hashes as “handle” to actual content are the key to data management

- $h(data) \longrightarrow hash$ produces a unique hash for each data file
- $d(hash) \longrightarrow data$ allows to retrieve the data when the hash is known
- *Hash trees* provide a mean to distribute large files
- *Relationships* between data revisions are stored as *graph*, using the hashes as symbol for the actual data

Hash–Indexed Content

Hashes as “handle” to actual content are the key to data management

- $h(data) \longrightarrow hash$ produces a unique hash for each data file
- $d(hash) \longrightarrow data$ allows to retrieve the data when the hash is known
- *Hash trees* provide a mean to distribute large files
- *Relationships* between data revisions are stored as *graph,* using the hashes as symbol for the actual data

Privacy

- You can control *with whom you share* (cryptography)
- Recalling information requires cooperation *("the net will not forget")*
- You *can't control* with whom your receivers *re–share* (impossibly of DRM)

There is no real anonymity, but your *traces can be lost* in the clouds

Privacy

- You can control *with whom you share* (cryptography)
- Recalling information requires cooperation *("the net will not forget")*
- You *can't control* with whom your receivers *re–share* (impossibly of DRM)
- There is no real anonymity, but your *traces can be lost* in the clouds

Privacy

- You can control *with whom you share* (cryptography)
- Recalling information requires cooperation *("the net will not forget")*
- You *can't control* with whom your receivers *re–share* (impossibly of DRM)
- There is no real anonymity, but your *traces can be lost* in the clouds

Privacy

- You can control *with whom you share* (cryptography)
- Recalling information requires cooperation *("the net will not forget")*
- You *can't control* with whom your receivers *re–share* (impossibly of DRM)
- There is no real anonymity, but your *traces can be lost* in the clouds

Secure Execution Environment

- Any sufficiently powerful language let you write malware
- The libraries of any sufficiently powerful environment (even with a very restricted language) contain enough exploits to write malware

Sandbox

- Sandbox the process, restrict network access (read is ok, write needs user permit)
- Using your keys (decyption, encryption, signing) must be outside the sandbox
- “Same origin”–policy doesn’t work for a data cloud — the destination is again “the cloud”

- Signed scripts and social control can help to some extend
- The boundary to malware is non–trivial to define. Is Farmville malware?

Secure Execution Environment

- Any sufficiently powerful language let you write malware
- The libraries of any sufficiently powerful environment (even with a very restricted language) contain enough exploits to write malware

Sandbox

- Sandbox the process, restrict network access (read is ok, write needs user permit)
- Using your keys (decyption, encryption, signing) must be outside the sandbox
- “Same origin”–policy doesn’t work for a data cloud — the destination is again “the cloud”

- Signed scripts and social control can help to some extend
- The boundary to malware is non–trivial to define. Is Farmville malware?

Secure Execution Environment

- Any sufficiently powerful language let you write malware
- The libraries of any sufficiently powerful environment (even with a very restricted language) contain enough exploits to write malware

Sandbox

- Sandbox the process, restrict network access (read is ok, write needs user permit)
- Using your keys (decyption, encryption, signing) must be outside the sandbox
- “Same origin”–policy doesn’t work for a data cloud — the destination is again “the cloud”

- Signed scripts and social control can help to some extend
- The boundary to malware is non–trivial to define. Is Farmville malware?

Secure Execution Environment

- Any sufficiently powerful language let you write malware
- The libraries of any sufficiently powerful environment (even with a very restricted language) contain enough exploits to write malware

Sandbox

- Sandbox the process, restrict network access (read is ok, write needs user permit)
- Using your keys (decyption, encryption, signing) must be outside the sandbox
- “Same origin”–policy doesn’t work for a data cloud — the destination is again “the cloud”

- Signed scripts and social control can help to some extend
- The boundary to malware is non–trivial to define. Is Farmville malware?

Secure Execution Environment

- Any sufficiently powerful language let you write malware
- The libraries of any sufficiently powerful environment (even with a very restricted language) contain enough exploits to write malware

Sandbox

- Sandbox the process, restrict network access (read is ok, write needs user permit)
- Using your keys (decyption, encryption, signing) must be outside the sandbox
- “Same origin”–policy doesn’t work for a data cloud — the destination is again “the cloud”

- Signed scripts and social control can help to some extend
- The boundary to malware is non–trivial to define. Is Farmville malware?

Secure Execution Environment

- Any sufficiently powerful language let you write malware
- The libraries of any sufficiently powerful environment (even with a very restricted language) contain enough exploits to write malware

Sandbox

- Sandbox the process, restrict network access (read is ok, write needs user permit)
- Using your keys (decyption, encryption, signing) must be outside the sandbox
- “Same origin”–policy doesn’t work for a data cloud — the destination is again “the cloud”

- Signed scripts and social control can help to some extend
- The boundary to malware is non–trivial to define. Is Farmville malware?

Secure Execution Environment

- Any sufficiently powerful language let you write malware
- The libraries of any sufficiently powerful environment (even with a very restricted language) contain enough exploits to write malware

Sandbox

- Sandbox the process, restrict network access (read is ok, write needs user permit)
- Using your keys (decyption, encryption, signing) must be outside the sandbox
- “Same origin”–policy doesn’t work for a data cloud — the destination is again “the cloud”

- Signed scripts and social control can help to some extend
- The boundary to malware is non–trivial to define. Is Farmville malware?

Push vs. Polling

- Polling is stupid
- Push–style solutions require open connections or ports
- Stored procedures in the cloud — or better call them “callbacks,” because they can only call the originator

Push vs. Polling

- Polling is stupid
- Push–style solutions require open connections or ports
- Stored procedures in the cloud — or better call them “callbacks,” because they can only call the originator

Push vs. Polling

- Polling is stupid
- Push–style solutions require open connections or ports
- Stored procedures in the cloud — or better call them “callbacks,” because they can only call the originator

Source Code vs. tokenized Binaries

- We (Forth) can compile source code quickly
- Source code distribution allows to inspect software, and reduce the malware threat (reduce, not eliminate!)
- Secretive companies like binaries, more difficult to reverse engineer
- Everybody can build his own VM compiler, if he likes to

Source Code vs. tokenized Binaries

- We (Forth) can compile source code quickly
- Source code distribution allows to inspect software, and reduce the malware threat (reduce, not eliminate!)
- Secretive companies like binaries, more difficult to reverse engineer
- Everybody can build his own VM compiler, if he likes to

Source Code vs. tokenized Binaries

- We (Forth) can compile source code quickly
- Source code distribution allows to inspect software, and reduce the malware threat (reduce, not eliminate!)
- Secretive companies like binaries, more difficult to reverse engineer
- Everybody can build his own VM compiler, if he likes to

Source Code vs. tokenized Binaries

- We (Forth) can compile source code quickly
- Source code distribution allows to inspect software, and reduce the malware threat (reduce, not eliminate!)
- Secretive companies like binaries, more difficult to reverse engineer
- Everybody can build his own VM compiler, if he likes to

Factor Data

scalability of graphics — bandwidth and detail reduction for small and slow devices

Images	Use progressive formats where scaled–down versions of the same image are transferred first (progressive JPEG, wavelet compression)
Video	Encode streams with a lowres, low–FPS base video, and additional streams which add spatial and temporal resolution
Geometries	Use level–of–detail algorithms to provide approximations of complex geometries (2D and 3D)

Factor Data

scalability of graphics — bandwidth and detail reduction for small and slow devices

Images	Use progressive formats where scaled–down versions of the same image are transferred first (progressive JPEG, wavelet compression)
Video	Encode streams with a lowres, low–FPS base video, and additional streams which add spatial and temporal resolution
Geometries	Use level–of–detail algorithms to provide approximations of complex geometries (2D and 3D)

Factor Data

scalability of graphics — bandwidth and detail reduction for small and slow devices

Images	Use progressive formats where scaled–down versions of the same image are transferred first (progressive JPEG, wavelet compression)
Video	Encode streams with a lowres, low–FPS base video, and additional streams which add spatial and temporal resolution
Geometries	Use level–of–detail algorithms to provide approximations of complex geometries (2D and 3D)

Factor the Code

- Provide a way to distribute common libraries
- Use the version control system to request the right dependencies, if you need those
- Allow precompiled basic functions to speed up rendering startup

Factor the Code

- Provide a way to distribute common libraries
- Use the version control system to request the right dependencies, if you need those
- Allow precompiled basic functions to speed up rendering startup

Factor the Code

- Provide a way to distribute common libraries
- Use the version control system to request the right dependencies, if you need those
- Allow precompiled basic functions to speed up rendering startup

Basic Libraries

- Canvas and OpenGL for 2D and 3D rendering
- HarfBuzz for text layout and shaping engine
- A typesetting engine (codename “$\mathrm{B_UX}$”)
- JPEG/PNG decoding
- Video engine
- Audio engine
- GUI library (MINOS–like, but using the rendering infrastructure)

Basic Libraries

- Canvas and OpenGL for 2D and 3D rendering
- HarfBuzz for text layout and shaping engine
- A typesetting engine (codename “$\mathrm{B_{U}X}$”)
- JPEG/PNG decoding
- Video engine
- Audio engine
- GUI library (MINOS–like, but using the rendering infrastructure)

Basic Libraries

- Canvas and OpenGL for 2D and 3D rendering
- HarfBuzz for text layout and shaping engine
- A typesetting engine (codename “B$_{\text{U}}$X”)
- JPEG/PNG decoding
- Video engine
- Audio engine
- GUI library (MINOS–like, but using the rendering infrastructure)

Basic Libraries

- Canvas and OpenGL for 2D and 3D rendering
- HarfBuzz for text layout and shaping engine
- A typesetting engine (codename “B$_{U}$X”)
- JPEG/PNG decoding
- Video engine
- Audio engine
- GUI library (MINOS–like, but using the rendering infrastructure)

Basic Libraries

- Canvas and OpenGL for 2D and 3D rendering
- HarfBuzz for text layout and shaping engine
- A typesetting engine (codename “B$_U$X”)
- JPEG/PNG decoding
- Video engine
- Audio engine
- GUI library (MINOS–like, but using the rendering infrastructure)

Basic Libraries

- Canvas and OpenGL for 2D and 3D rendering
- HarfBuzz for text layout and shaping engine
- A typesetting engine (codename “$\text{B}_{\text{U}}\text{X}$”)
- JPEG/PNG decoding
- Video engine
- Audio engine
- GUI library (MINOS–like, but using the rendering infrastructure)

Basic Libraries

- Canvas and OpenGL for 2D and 3D rendering
- HarfBuzz for text layout and shaping engine
- A typesetting engine (codename “$\mathrm{B}_{\mathrm{U}}\mathrm{X}$”)
- JPEG/PNG decoding
- Video engine
- Audio engine
- GUI library (MINOS–like, but using the rendering infrastructure)

Most Stupid Mistakes of the Net 1.0

- There really must be an easy to use (i.e. WYSIWYG) *in–browser editor* for the content!
- *Client–server* instead of peer to peer — the original idea behind the Internet was peer to peer, but it was soon forgotten
- *Postel principle* — do not be liberal in what you receive, do explicit consistency checks even if that is costly (Rose principle)
- *Unencrypted* by default (was too costly; but then, cryptographic protocols such as SSL are really ugly and full of mistakes)

Most Stupid Mistakes of the Net 1.0

- There really must be an easy to use (i.e. WYSIWYG) *in–browser editor* for the content!
- *Client–server* instead of peer to peer — the original idea behind the Internet was peer to peer, but it was soon forgotten
- *Postel principle* — do not be liberal in what you receive, do explicit consistency checks even if that is costly (Rose principle)
- *Unencrypted* by default (was too costly; but then, cryptographic protocols such as SSL are really ugly and full of mistakes)

Most Stupid Mistakes of the Net 1.0

- There really must be an easy to use (i.e. WYSIWYG) *in–browser editor* for the content!
- *Client–server* instead of peer to peer — the original idea behind the Internet was peer to peer, but it was soon forgotten
- *Postel principle* — do not be liberal in what you receive, do explicit consistency checks even if that is costly (Rose principle)
- *Unencrypted* by default (was too costly; but then, cryptographic protocols such as SSL are really ugly and full of mistakes)

Most Stupid Mistakes of the Net 1.0

- There really must be an easy to use (i.e. WYSIWYG) *in–browser editor* for the content!
- *Client–server* instead of peer to peer — the original idea behind the Internet was peer to peer, but it was soon forgotten
- *Postel principle* — do not be liberal in what you receive, do explicit consistency checks even if that is costly (Rose principle)
- *Unencrypted* by default (was too costly; but then, cryptographic protocols such as SSL are really ugly and full of mistakes)

For Further Reading

- BERND PAYSAN
  Internet 2.0
  `http://net2o.de/`
- POUWELSE, GRISHCHENKO, BAKKER
  swift, the multiparty transport protocol
  `http://libswift.org/`
- ROSSI, TESTA, VALENTI
  Yes, we LEDBAT
  `http://www.pam2010.ethz.ch/papers/full-length/4.pdf`

For Further Reading

- BERND PAYSAN
  Internet 2.0
  http://net2o.de/
- POUWELSE, GRISHCHENKO, BAKKER
  swift, the multiparty transport protocol
  http://libswift.org/
- ROSSI, TESTA, VALENTI
  Yes, we LEDBAT
  http:
  //www.pam2010.ethz.ch/papers/full-length/4.pdf

For Further Reading

- BERND PAYSAN
  Internet 2.0
  http://net2o.de/
- POUWELSE, GRISHCHENKO, BAKKER
  swift, the multiparty transport protocol
  http://libswift.org/
- ROSSI, TESTA, VALENTI
  Yes, we LEDBAT
  http://www.pam2010.ethz.ch/papers/full-length/4.pdf