net2o: Reinventing the Internet

Bernd Paysan (forthy)

YBTI in depth session, 30C3, Hamburg *(later added stuff in italics)*

Outline

Somebody Broke the Internet...

- My thoughts about reinventing the Internet started in 2005. Yes, in 2005.
- Things broken in 2005: IE6 won the browser war, Windows XP “naked” on the Internet was infected within 30 seconds with Sasser...
- Back then I had a new responsibility: do the IT of my (former) employer on top of the IC design duties.
- 1000 competing protocols and standards for 100 things, none of them really good...
- Then we got Facebook and Cloud computing...
- Fast forward: in June 2013 Edward Snowden revealed that it's worse than the worst conspiracy theory

Somebody Broke the Internet...

- My thoughts about reinventing the Internet started in 2005. Yes, in 2005.
- Things broken in 2005: IE6 won the browser war, Windows XP "naked" on the Internet was infected within 30 seconds with Sasser...
- Back then I had a new responsibility: do the IT of my (former) employer on top of the IC design duties.
- 1000 competing protocols and standards for 100 things, none of them really good...
- Then we got Facebook and Cloud computing...
- Fast forward: in June 2013 Edward Snowden revealed that it's worse than the worst conspiracy theory

Somebody Broke the Internet...

- My thoughts about reinventing the Internet started in 2005. Yes, in 2005.
- Things broken in 2005: IE6 won the browser war, Windows XP “naked” on the Internet was infected within 30 seconds with Sasser...
- Back then I had a new responsibility: do the IT of my (former) employer on top of the IC design duties.
- 1000 competing protocols and standards for 100 things, none of them really good...
- Then we got Facebook and Cloud computing...
- Fast forward: in June 2013 Edward Snowden revealed that it's worse than the worst conspiracy theory

Somebody Broke the Internet...

- My thoughts about reinventing the Internet started in 2005. Yes, in 2005.
- Things broken in 2005: IE6 won the browser war, Windows XP "naked" on the Internet was infected within 30 seconds with Sasser...
- Back then I had a new responsibility: do the IT of my (former) employer on top of the IC design duties.
- 1000 competing protocols and standards for 100 things, none of them really good...
- Then we got Facebook and Cloud computing...
- Fast forward: in June 2013 EDWARD SNOWDEN revealed that it's worse than the worst conspiracy theory

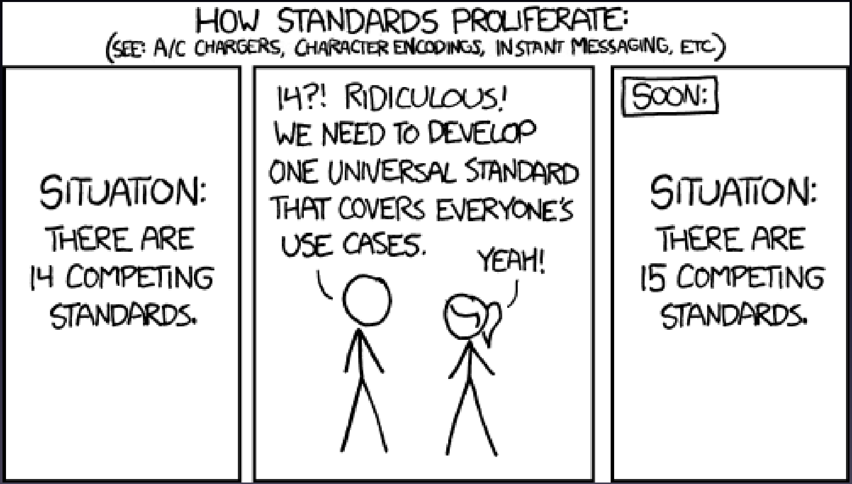
Somebody Broke the Internet...

- My thoughts about reinventing the Internet started in 2005. Yes, in 2005.
- Things broken in 2005: IE6 won the browser war, Windows XP "naked" on the Internet was infected within 30 seconds with Sasser...
- Back then I had a new responsibility: do the IT of my (former) employer on top of the IC design duties.
- 1000 competing protocols and standards for 100 things, none of them really good...
- Then we got Facebook and Cloud computing...
- Fast forward: in June 2013 Edward Snowden revealed that it's worse than the worst conspiracy theory

Somebody Broke the Internet…

- My thoughts about reinventing the Internet started in 2005. Yes, in 2005.
- Things broken in 2005: IE6 won the browser war, Windows XP “naked” on the Internet was infected within 30 seconds with Sasser…
- Back then I had a new responsibility: do the IT of my (former) employer on top of the IC design duties.
- 1000 competing protocols and standards for 100 things, none of them really good…
- Then we got Facebook and Cloud computing…
- Fast forward: in June 2013 Edward Snowden revealed that it’s worse than the worst conspiracy theory

The Problem of 1000 Standards



Solution: Start from Scratch

Pretty radical step

What to keep from the current Internet, and what to throw away:

The Good: Packet–oriented protocol, open and free standards, connect everybody with everybody else

The Bad: Unencrypted by default, not enough addresses in IPv4, very slow adaption of IPv6, Postel principle leads to pretty bad implementations

The Ugly: Complex protocol stacks requires lots of resources to be fast, layering violations e.g. in encryption, many protocols doing similar stuff

Solution: Start from Scratch

Pretty radical step

What to keep from the current Internet, and what to throw away:

The Good Packet–oriented protocol, open and free standards, connect everybody with everybody else

The Bad Unencrypted by default, not enough addresses in IPv4, very slow adaption of IPv6, Postel principle leads to pretty bad implementations

The Ugly Complex protocol stacks requires lots of resources to be fast, layering violations e.g. in encryption, many protocols doing similar stuff

Solution: Start from Scratch

Pretty radical step

What to keep from the current Internet, and what to throw away:

- **The Good** Packet–oriented protocol, open and free standards, connect everybody with everybody else
- **The Bad** Unencrypted by default, not enough addresses in IPv4, very slow adaption of IPv6, Postel principle leads to pretty bad implementations
- **The Ugly** Complex protocol stacks requires lots of resources to be fast, layering violations e.g. in encryption, many protocols doing similar stuff

Solution: Start from Scratch

Pretty radical step

What to keep from the current Internet, and what to throw away:

The Good Packet–oriented protocol, open and free standards, connect everybody with everybody else

The Bad Unencrypted by default, not enough addresses in IPv4, very slow adaption of IPv6, Postel principle leads to pretty bad implementations

The Ugly Complex protocol stacks requires lots of resources to be fast, layering violations e.g. in encryption, many protocols doing similar stuff

Requirements

Scalability Must work well with low and high bandwidths, loose and tightly coupled systems, few and many hosts connected together over short to far distances.

Easy to implement Must work with a minimum of effort, must allow small and cheap devices to connect. One idea is to replace “busses” like USB or even Display Port with LAN links.

Security Users want authentication and authorization, but also anonymity and privacy. Firewalls and similar gatekeepers (load balancers, etc.) are common.

Media capable This requires real–time capabilities, maybe pre–allocated bandwidth and other QoS features, end-to-end.

Transparency Must be able to work together with other networks (especially Internet 1.0, using UDP).

Requirements

Scalability Must work well with low and high bandwidths, loose and tightly coupled systems, few and many hosts connected together over short to far distances.

Easy to implement Must work with a minimum of effort, must allow small and cheap devices to connect. One idea is to replace "busses" like USB or even Display Port with LAN links.

Security Users want authentication and authorization, but also anonymity and privacy. Firewalls and similar gatekeepers (load balancers, etc.) are common.

Media capable This requires real–time capabilities, maybe pre–allocated bandwidth and other QoS features, end-to-end.

Transparency Must be able to work together with other networks (especially Internet 1.0, using UDP).

Requirements

Scalability Must work well with low and high bandwidths, loose and tightly coupled systems, few and many hosts connected together over short to far distances.

Easy to implement Must work with a minimum of effort, must allow small and cheap devices to connect. One idea is to replace “busses” like USB or even Display Port with LAN links.

Security Users want authentication and authorization, but also anonymity and privacy. Firewalls and similar gatekeepers (load balancers, etc.) are common.

Media capable This requires real–time capabilities, maybe pre–allocated bandwidth and other QoS features, end-to-end.

Transparency Must be able to work together with other networks (especially Internet 1.0, using UDP).

Requirements

Scalability Must work well with low and high bandwidths, loose and tightly coupled systems, few and many hosts connected together over short to far distances.

Easy to implement Must work with a minimum of effort, must allow small and cheap devices to connect. One idea is to replace “busses” like USB or even Display Port with LAN links.

Security Users want authentication and authorization, but also anonymity and privacy. Firewalls and similar gatekeepers (load balancers, etc.) are common.

Media capable This requires real–time capabilities, maybe pre–allocated bandwidth and other QoS features, end-to-end.

Transparency Must be able to work together with other networks (especially Internet 1.0, using UDP).

Requirements

Scalability
: Must work well with low and high bandwidths, loose and tightly coupled systems, few and many hosts connected together over short to far distances.

Easy to implement
: Must work with a minimum of effort, must allow small and cheap devices to connect. One idea is to replace “busses” like USB or even Display Port with LAN links.

Security
: Users want authentication and authorization, but also anonymity and privacy. Firewalls and similar gatekeepers (load balancers, etc.) are common.

Media capable
: This requires real–time capabilities, maybe pre–allocated bandwidth and other QoS features, end-to-end.

Transparency
: Must be able to work together with other networks (especially Internet 1.0, using UDP).

My Background

- I've done hardware development from analog to digital, and software development from embedded control, compilers to GUIs
- I therefore think I can comprehend an network stack from top to bottom
- I hate bloated, complex solutions, and I like simple, elegant ones
- The rule #1 of empowering the strong is "If you want it done right, you have to do it yourself"
- Avoid unnecessary abstractions: Abstractions serve a purpose, they are not a value of their own
- I like to program in Forth, as this is simple, elegant, malleable, and close to

My Background

- I've done hardware development from analog to digital, and software development from embedded control, compilers to GUIs
- I therefore think I can comprehend an network stack from top to bottom
- I hate bloated, complex solutions, and I like simple, elegant ones
- The rule #1 of empowering the strong is "If you want it done right, you have to do it yourself"
- Avoid unnecessary abstractions: Abstractions serve a purpose, they are not a value of their own
- I like to program in Forth, as this is simple, elegant, malleable, and close to hardware

My Background

- I've done hardware development from analog to digital, and software development from embedded control, compilers to GUIs
- I therefore think I can comprehend an network stack from top to bottom
- I hate bloated, complex solutions, and I like simple, elegant ones
- The rule #1 of empowering the strong is "If you want it done right, you have to do it yourself"
- Avoid unnecessary abstractions: Abstractions serve a purpose, they are not a value of their own
- I like to program in Forth, as this is simple, elegant, malleable, and close to

My Background

- I've done hardware development from analog to digital, and software development from embedded control, compilers to GUIs
- I therefore think I can comprehend an network stack from top to bottom
- I hate bloated, complex solutions, and I like simple, elegant ones
- The rule #1 of empowering the strong is "If you want it done right, you have to do it yourself"
- Avoid unnecessary abstractions: Abstractions serve a purpose, they are not a value of their own
- I like to program in Forth, as this is simple, elegant, malleable, and close to hardware

My Background

- I've done hardware development from analog to digital, and software development from embedded control, compilers to GUIs
- I therefore think I can comprehend an network stack from top to bottom
- I hate bloated, complex solutions, and I like simple, elegant ones
- The rule #1 of empowering the strong is "If you want it done right, you have to do it yourself"
- Avoid unnecessary abstractions: Abstractions serve a purpose, they are not a value of their own
- I like to program in Forth, as this is simple, elegant, malleable, and close to hardware

My Background

- I've done hardware development from analog to digital, and software development from embedded control, compilers to GUIs
- I therefore think I can comprehend an network stack from top to bottom
- I hate bloated, complex solutions, and I like simple, elegant ones
- The rule #1 of empowering the strong is "If you want it done right, you have to do it yourself"
- Avoid unnecessary abstractions: Abstractions serve a purpose, they are not a value of their own
- I like to program in Forth, as this is simple, elegant, malleable, and close to hardware

This is a lot of Research

My usual approach is:

1. Look at what's out there
2. Evaluate or at least judge it
3. Conclude that it is broken
4. Go back to rule #1: "If you want it done right, you have to do it yourself"

Sometimes, there's something out there that does work, though.

This is a lot of Research

My usual approach is:

1. Look at what's out there
2. Evaluate or at least judge it
3. Conclude that it is broken
4. Go back to rule #1: "If you want it done right, you have to do it yourself"

Sometimes, there's something out there that does work, though.

This is a lot of Research

My usual approach is:

1. Look at what's out there
2. Evaluate or at least judge it
3. Conclude that it is broken
4. Go back to rule #1: "If you want it done right, you have to do it yourself"

Sometimes, there's something out there that does work, though.

This is a lot of Research

My usual approach is:

1. Look at what's out there
2. Evaluate or at least judge it
3. Conclude that it is broken
4. Go back to rule #1: "If you want it done right, you have to do it yourself"

Sometimes, there's something out there that does work, though.

This is a lot of Research

My usual approach is:

1. Look at what's out there
2. Evaluate or at least judge it
3. Conclude that it is broken
4. Go back to rule #1: "If you want it done right, you have to do it yourself"

Sometimes, there's something out there that does work, though.

This is a lot of Research

My usual approach is:

1. Look at what's out there
2. Evaluate or at least judge it
3. Conclude that it is broken
4. Go back to rule #1: "If you want it done right, you have to do it yourself"

Sometimes, there's something out there that does work, though.

Abstractions

- Network: Lines connected by switches
- Nodes: shared memory buffers — remote write, local read (of course, the network stack can only access the memory that it is assigned to!)
- Separation of commands and (bulk) data packets
- Every*thing* is a file — with metadata (“tags”) in a hash table, every*one* is a key (with metadata)
- Event–driven design: command packets are executed remotely and drive the protocol
- P2P: all nodes are equal, no client–server distinction, content–oriented file

Abstractions

- Network: Lines connected by switches
- Nodes: shared memory buffers — remote write, local read (of course, the network stack can only access the memory that it is assigned to!)
- Separation of commands and (bulk) data packets
- Every*thing* is a file — with metadata (“tags”) in a hash table, every*one* is a key (with metadata)
- Event–driven design: command packets are executed remotely and drive the protocol
- P2P: all nodes are equal, no client–server distinction, content–oriented file

Abstractions

- Network: Lines connected by switches
- Nodes: shared memory buffers — remote write, local read (of course, the network stack can only access the memory that it is assigned to!)
- Separation of commands and (bulk) data packets
- Every*thing* is a file — with metadata (“tags”) in a hash table, every*one* is a key (with metadata)
- Event–driven design: command packets are executed remotely and drive the protocol
- P2P: all nodes are equal, no client–server distinction, content–oriented file

Abstractions

- Network: Lines connected by switches
- Nodes: shared memory buffers — remote write, local read (of course, the network stack can only access the memory that it is assigned to!)
- Separation of commands and (bulk) data packets
- Every*thing* is a file — with metadata (“tags”) in a hash table, every*one* is a key (with metadata)
- Event–driven design: command packets are executed remotely and drive the protocol
- P2P: all nodes are equal, no client–server distinction, content–oriented file

Abstractions

- Network: Lines connected by switches
- Nodes: shared memory buffers — remote write, local read (of course, the network stack can only access the memory that it is assigned to!)
- Separation of commands and (bulk) data packets
- Every*thing* is a file — with metadata (“tags”) in a hash table, every*one* is a key (with metadata)
- Event–driven design: command packets are executed remotely and drive the protocol
- P2P: all nodes are equal, no client–server distinction, content–oriented file “objects”

Abstractions

- Network: Lines connected by switches
- Nodes: shared memory buffers — remote write, local read (of course, the network stack can only access the memory that it is assigned to!)
- Separation of commands and (bulk) data packets
- Every*thing* is a file — with metadata ("tags") in a hash table, every*one* is a key (with metadata)
- Event–driven design: command packets are executed remotely and drive the protocol
- P2P: all nodes are equal, no client–server distinction, content–oriented file "objects"

net2o in a nutshell

net2o consists of the following 6 layers:

2. Path switched packets with 2^n size writing into shared memory buffers
3. Ephemeral key exchange and signatures with Ed25519, symmetric authenticated encryption+hash+prng with Keccak
4. Timing driven delay minimizing flow control
5. Stack–oriented tokenized command language
6. Distributed data (files) and distributed metadata (prefix hash trie)
7. Apps in a sandboxed environment for displaying content

net2o in a nutshell

net2o consists of the following 6 layers:

2. Path switched packets with 2^n size writing into shared memory buffers
3. Ephemeral key exchange and signatures with Ed25519, symmetric authenticated encryption+hash+prng with Keccak
4. Timing driven delay minimizing flow control
5. Stack–oriented tokenized command language
6. Distributed data (files) and distributed metadata (prefix hash trie)
7. Apps in a sandboxed environment for displaying content

net2o in a nutshell

net2o consists of the following 6 layers:

2. Path switched packets with 2^n size writing into shared memory buffers
3. Ephemeral key exchange and signatures with Ed25519, symmetric authenticated encryption+hash+prng with Keccak
4. Timing driven delay minimizing flow control
5. Stack–oriented tokenized command language
6. Distributed data (files) and distributed metadata (prefix hash trie)
7. Apps in a sandboxed environment for displaying content

net2o in a nutshell

net2o consists of the following 6 layers:

2. Path switched packets with 2^n size writing into shared memory buffers
3. Ephemeral key exchange and signatures with Ed25519, symmetric authenticated encryption+hash+prng with Keccak
4. Timing driven delay minimizing flow control
5. Stack–oriented tokenized command language
6. Distributed data (files) and distributed metadata (prefix hash trie)
7. Apps in a sandboxed environment for displaying content

net2o in a nutshell

net2o consists of the following 6 layers:

2. Path switched packets with 2^n size writing into shared memory buffers
3. Ephemeral key exchange and signatures with Ed25519, symmetric authenticated encryption+hash+prng with Keccak
4. Timing driven delay minimizing flow control
5. Stack–oriented tokenized command language
6. Distributed data (files) and distributed metadata (prefix hash trie)
7. Apps in a sandboxed environment for displaying content

net2o in a nutshell

net2o consists of the following 6 layers:

2. Path switched packets with 2^n size writing into shared memory buffers
3. Ephemeral key exchange and signatures with Ed25519, symmetric authenticated encryption+hash+prng with Keccak
4. Timing driven delay minimizing flow control
5. Stack–oriented tokenized command language
6. Distributed data (files) and distributed metadata (prefix hash trie)
7. Apps in a sandboxed environment for displaying content

net2o in a nutshell

net2o consists of the following 6 layers:

2. Path switched packets with 2^n size writing into shared memory buffers
3. Ephemeral key exchange and signatures with Ed25519, symmetric authenticated encryption+hash+prng with Keccak
4. Timing driven delay minimizing flow control
5. Stack–oriented tokenized command language
6. Distributed data (files) and distributed metadata (prefix hash trie)
7. Apps in a sandboxed environment for displaying content

Switching Packets, Routing Connections

- Switches are faster and easier to implement than routers — LANs (Ethernet) and backbones (MPLS) already use switching; use the concept of MPLS label stacks to use switching everywhere
- Routing then is a combination of DNS–like destination resolution and routing calculation (destination path lookup)

Path Switching

- Take first n bits of path field and select destination
- Shift target address by n
- Insert bit-reversed source into the rear end of the path field to mark the way back

- The receiver bit–flips the path field, and gets the return address

Switching Packets, Routing Connections

- Switches are faster and easier to implement than routers — LANs (Ethernet) and backbones (MPLS) already use switching; use the concept of MPLS label stacks to use switching everywhere
- Routing then is a combination of DNS–like destination resolution and routing calculation (destination path lookup)

Path Switching

- Take first n bits of path field and select destination
- Shift target address by n
- Insert bit-reversed source into the rear end of the path field to mark the way back

- The receiver bit–flips the path field, and gets the return address

Switching Packets, Routing Connections

- Switches are faster and easier to implement than routers — LANs (Ethernet) and backbones (MPLS) already use switching; use the concept of MPLS label stacks to use switching everywhere
- Routing then is a combination of DNS–like destination resolution and routing calculation (destination path lookup)

Path Switching

- Take first n bits of path field and select destination
- Shift target address by n
- Insert bit-reversed source into the rear end of the path field to mark the way back

- The receiver bit–flips the path field, and gets the return address

Switching Packets, Routing Connections

- Switches are faster and easier to implement than routers — LANs (Ethernet) and backbones (MPLS) already use switching; use the concept of MPLS label stacks to use switching everywhere
- Routing then is a combination of DNS–like destination resolution and routing calculation (destination path lookup)

Path Switching

- Take first n bits of path field and select destination
- Shift target address by n
- Insert bit-reversed source into the rear end of the path field to mark the way back

- The receiver bit–flips the path field, and gets the return address

Switching Packets, Routing Connections

- Switches are faster and easier to implement than routers — LANs (Ethernet) and backbones (MPLS) already use switching; use the concept of MPLS label stacks to use switching everywhere
- Routing then is a combination of DNS–like destination resolution and routing calculation (destination path lookup)

Path Switching

- Take first *n* bits of path field and select destination
- Shift target address by *n*
- Insert bit-reversed source into the rear end of the path field to mark the way back

- The receiver bit–flips the path field, and gets the return address

Switching Packets, Routing Connections

- Switches are faster and easier to implement than routers — LANs (Ethernet) and backbones (MPLS) already use switching; use the concept of MPLS label stacks to use switching everywhere
- Routing then is a combination of DNS–like destination resolution and routing calculation (destination path lookup)

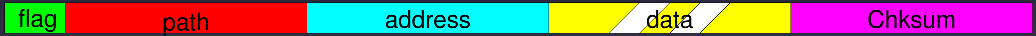
Path Switching

- Take first n bits of path field and select destination
- Shift target address by n
- Insert bit-reversed source into the rear end of the path field to mark the way back

- The receiver bit–flips the path field, and gets the return address

Packet Format

	Bytes	*Comment*
Flags	2	priority, length, flow control flags
Path	16	Internet 1.0 terminology: “address”
Address	8	address in memory, ≈port+sequence number
Data	$64 * 2^{0..15}$	up to 2MB packet size, enough for the next 40 years
Chksum	16	cryptographic checksum



Handover

- Typical problem in our mobile world: Devices hop from one network into another
- To avoid connection loss, you need a handover
- net2o handover works with the assumption that properly authenticated packets are ok, and then accepts a change in the return path
- The remaining problem are two simultaneous handovers, and the suggestion therefore is: Keep using both networks for the transition period.

Handover

- Typical problem in our mobile world: Devices hop from one network into another
- To avoid connection loss, you need a handover
- net2o handover works with the assumption that properly authenticated packets are ok, and then accepts a change in the return path
- The remaining problem are two simultaneous handovers, and the suggestion therefore is: Keep using both networks for the transition period.

Handover

- Typical problem in our mobile world: Devices hop from one network into another
- To avoid connection loss, you need a handover
- net2o handover works with the assumption that properly authenticated packets are ok, and then accepts a change in the return path
- The remaining problem are two simultaneous handovers, and the suggestion therefore is: Keep using both networks for the transition period.

Handover

- Typical problem in our mobile world: Devices hop from one network into another
- To avoid connection loss, you need a handover
- net2o handover works with the assumption that properly authenticated packets are ok, and then accepts a change in the return path
- The remaining problem are two simultaneous handovers, and the suggestion therefore is: Keep using both networks for the transition period.

Routing Example *(added for cjdns)*

Assumption: Somewhat hierarchical structure: backbones, ISPs, LANs.

- My symbolic path to a backbone: laptop.net2o.1und1.level3
- Destination's path to a backbone: foobar.webhoster.bay-net.alter-net
- Connect paths together (reverse second): laptop.net2o.1und1.level3.alter-net.bay-net.webhoster.foobar
- Neighboring entities know the path from one to the other, e.g. "1und1" knows how to connect "net2o" to "level3", so you ask them (and cache the result in the DHT)
- Splice the labels together, and you get a path: 1010 || 1101.0001.0101.1000.1011.0011 || 0110.1010 || 0111.1010 || 1010.0010.0001.1001.1010.0100 || 0110.1011.0111

Routing Example *(added for cjdns)*

Assumption: Somewhat hierarchical structure: backbones, ISPs, LANs.

- My symbolic path to a backbone: laptop.net2o.1und1.level3
- Destination's path to a backbone: foobar.webhoster.bay-net.alter-net
- Connect paths together (reverse second): laptop.net2o.1und1.level3.alter-net.bay-net.webhoster.foobar
- Neighboring entities know the path from one to the other, e.g. "1und1" knows how to connect "net2o" to "level3", so you ask them (and cache the result in the DHT)
- Splice the labels together, and you get a path: 1010 || 1101.0001.0101.1000.1011.0011 || 0110.1010 || 0111.1010 || 1010.0010.0001.1001.1010.0100 || 0110.1011.0111

Routing Example *(added for cjdns)*

Assumption: Somewhat hierarchical structure: backbones, ISPs, LANs.

- My symbolic path to a backbone: laptop.net2o.1und1.level3
- Destination's path to a backbone: foobar.webhoster.bay-net.alter-net
- Connect paths together (reverse second): laptop.net2o.1und1.level3.alter-net.bay-net.webhoster.foobar
- Neighboring entities know the path from one to the other, e.g. "1und1" knows how to connect "net2o" to "level3", so you ask them (and cache the result in the DHT)
- Splice the labels together, and you get a path: 1010 || 1101.0001.0101.1000.1011.0011 || 0110.1010 || 0111.1010 || 1010.0010.0001.1001.1010.0100 || 0110.1011.0111

Routing Example *(added for cjdns)*

Assumption: Somewhat hierarchical structure: backbones, ISPs, LANs.

- My symbolic path to a backbone: laptop.net2o.1und1.level3
- Destination's path to a backbone: foobar.webhoster.bay-net.alter-net
- Connect paths together (reverse second):
  laptop.net2o.1und1.level3.alter-net.bay-net.webhoster.foobar
- Neighboring entities know the path from one to the other, e.g. "1und1" knows how to connect "net2o" to "level3", so you ask them (and cache the result in the DHT)
- Splice the labels together, and you get a path:
  1010 || 1101.0001.0101.1000.1011.0011 || 0110.1010 || 0111.1010 || 1010.0010.0001.1001.1010.0100 || 0110.1011.0111

Routing Example *(added for cjdns)*

Assumption: Somewhat hierarchical structure: backbones, ISPs, LANs.

- My symbolic path to a backbone: laptop.net2o.1und1.level3
- Destination's path to a backbone: foobar.webhoster.bay-net.alter-net
- Connect paths together (reverse second):
  laptop.net2o.1und1.level3.alter-net.bay-net.webhoster.foobar
- Neighboring entities know the path from one to the other, e.g. "1und1" knows how to connect "net2o" to "level3", so you ask them (and cache the result in the DHT)
- Splice the labels together, and you get a path:
  1010 || 1101.0001.0101.1000.1011.0011 || 0110.1010 || 0111.1010 || 1010.0010.0001.1001.1010.0100 || 0110.1011.0111

Routing Example *(added for cjdns)*

Assumption: Somewhat hierarchical structure: backbones, ISPs, LANs.

- My symbolic path to a backbone: laptop.net2o.1und1.level3
- Destination's path to a backbone: foobar.webhoster.bay-net.alter-net
- Connect paths together (reverse second):
  laptop.net2o.1und1.level3.alter-net.bay-net.webhoster.foobar
- Neighboring entities know the path from one to the other, e.g. "1und1" knows how to connect "net2o" to "level3", so you ask them (and cache the result in the DHT)
- Splice the labels together, and you get a path:
  1010 || 1101.0001.0101.1000.1011.0011 || 0110.1010 || 0111.1010 || 1010.0010.0001.1001.1010.0100 || 0110.1011.0111

Why Source Routing

Three possible schemes

1. switched circuit (POTS, virtual: ATM, MPLS)
2. unique identifier (IP)
3. source routing

- Separation of network gear and computers: Fast, dumb, stateless equipment for routing/switching
- The hierarchical topology is a derived "law of nature": people cluster together and connect clusters
- Attack vector is only bandwidth–based, and this can be mitigated (see "fair routing" below)
- Routing slice is an implementation detail of each network segment (i.e. is a unique identifier within each subnet)

Why Source Routing

Three possible schemes

1. switched circuit (POTS, virtual: ATM, MPLS)
2. unique identifier (IP)
3. source routing

- Separation of network gear and computers: Fast, dumb, stateless equipment for routing/switching
- The hierarchical topology is a derived "law of nature": people cluster together and connect clusters
- Attack vector is only bandwidth–based, and this can be mitigated (see "fair routing" below)
- Routing slice is an implementation detail of each network segment (i.e. is a unique identifier within each subnet)

Why Source Routing

Three possible schemes

1. switched circuit (POTS, virtual: ATM, MPLS)
2. unique identifier (IP)
3. source routing

- Separation of network gear and computers: Fast, dumb, stateless equipment for routing/switching
- The hierarchical topology is a derived “law of nature”: people cluster together and connect clusters
- Attack vector is only bandwidth–based, and this can be mitigated (see “fair routing” below)
- Routing slice is an implementation detail of each network segment (i.e. is a unique identifier within each subnet)

Why Source Routing

Three possible schemes

1. switched circuit (POTS, virtual: ATM, MPLS)
2. unique identifier (IP)
3. source routing

- Separation of network gear and computers: Fast, dumb, stateless equipment for routing/switching
- The hierarchical topology is a derived "law of nature": people cluster together and connect clusters
- Attack vector is only bandwidth–based, and this can be mitigated (see "fair routing" below)
- Routing slice is an implementation detail of each network segment (i.e. is a unique identifier within each subnet)

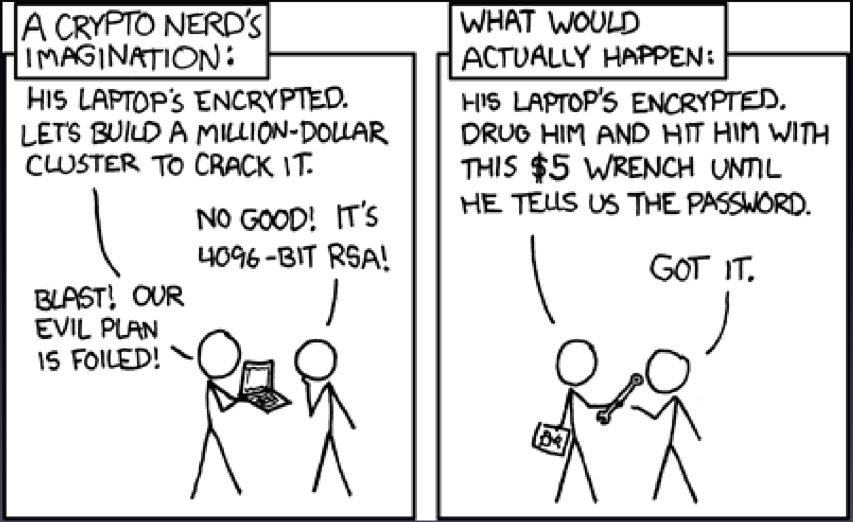
Why Source Routing

Three possible schemes

1. switched circuit (POTS, virtual: ATM, MPLS)
2. unique identifier (IP)
3. source routing

- Separation of network gear and computers: Fast, dumb, stateless equipment for routing/switching
- The hierarchical topology is a derived “law of nature”: people cluster together and connect clusters
- Attack vector is only bandwidth–based, and this can be mitigated (see “fair routing” below)
- Routing slice is an implementation detail of each network segment (i.e. is a unique identifier within each subnet)

Security: Indirect Attacks are Cheaper



Key Exchange

Evaluation of encryption algorithms

RSA Pubkeys for reasonable strength are 4kbit or more; factoring is no longer "that hard"; further breakthroughs can be expected (RSA challenge withdrew the prices). *See "the year on crypto" presentation from djb et al for more worrying stuff.* 4kbit is 512 bytes, for the session invocation protocol this is above the ∼1kB limit I've on current Internet.

Diffie–Hellman Key strength to length relation is about the same as with RSA, so the same problem applies. Breakthroughs require non–linear expansion of key size; archived encryption can be decrypted later

ECC Elliptic Curve Cryptography has still only a generic attack (i.e. can be considered "unscratched", as the attack uses a fundamental property of the problem), and therefore 256 bit keys (32 bytes) have a strength of 128 bits

Therefore the choice now is Ed25519, a variant of Curve25519 from Dan Bernstein that supports signatures, too. This is a curve where the parameters are of high quality.

Key Exchange

Evaluation of encryption algorithms

RSA Pubkeys for reasonable strength are 4kbit or more; factoring is no longer "that hard"; further breakthroughs can be expected (RSA challenge withdrew the prices). *See "the year on crypto" presentation from djb et al for more worrying stuff.* 4kbit is 512 bytes, for the session invocation protocol this is above the ∼1kB limit I've on current Internet.

Diffie–Hellman Key strength to length relation is about the same as with RSA, so the same problem applies. Breakthroughs require non–linear expansion of key size; archived encryption can be decrypted later

ECC Elliptic Curve Cryptography has still only a generic attack (i.e. can be considered "unscratched", as the attack uses a fundamental property of the problem), and therefore 256 bit keys (32 bytes) have a strength of 128 bits

Therefore the choice now is Ed25519, a variant of Curve25519 from Dan Bernstein that supports signatures, too. This is a curve where the parameters are of high quality.

Key Exchange

Evaluation of encryption algorithms

RSA Pubkeys for reasonable strength are 4kbit or more; factoring is no longer "that hard"; further breakthroughs can be expected (RSA challenge withdrew the prices). *See "the year on crypto" presentation from djb et al for more worrying stuff.* 4kbit is 512 bytes, for the session invocation protocol this is above the ∼1kB limit I've on current Internet.

Diffie–Hellman Key strength to length relation is about the same as with RSA, so the same problem applies. Breakthroughs require non–linear expansion of key size; archived encryption can be decrypted later

ECC Elliptic Curve Cryptography has still only a generic attack (i.e. can be considered "unscratched", as the attack uses a fundamental property of the problem), and therefore 256 bit keys (32 bytes) have a strength of 128 bits

Therefore the choice now is Ed25519, a variant of Curve25519 from Dan Bernstein that supports signatures, too. This is a curve where the parameters are of high quality.

Key Exchange

Evaluation of encryption algorithms

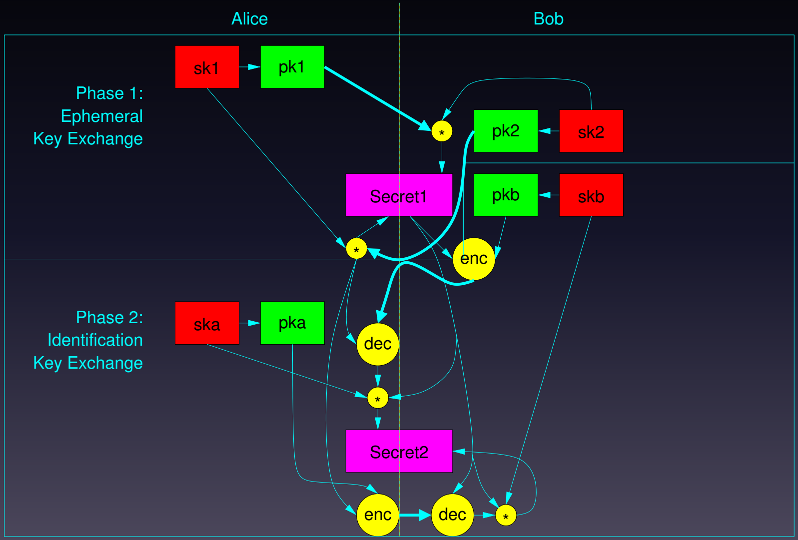
RSA Pubkeys for reasonable strength are 4kbit or more; factoring is no longer "that hard"; further breakthroughs can be expected (RSA challenge withdrew the prices). *See "the year on crypto" presentation from djb et al for more worrying stuff.* 4kbit is 512 bytes, for the session invocation protocol this is above the ∼1kB limit I've on current Internet.

Diffie–Hellman Key strength to length relation is about the same as with RSA, so the same problem applies. Breakthroughs require non–linear expansion of key size; archived encryption can be decrypted later

ECC Elliptic Curve Cryptography has still only a generic attack (i.e. can be considered "unscratched", as the attack uses a fundamental property of the problem), and therefore 256 bit keys (32 bytes) have a strength of 128 bits

Therefore the choice now is Ed25519, a variant of Curve25519 from Dan Bernstein that supports signatures, too. This is a curve where the parameters are of high quality.

Ephemeral Key Exchange+Validation



Challenge: Side–Channel Attacks

- ECC Diffie–Hellman key exchange formula is $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Operations with secret constants and variables under control of the attacker may leak information, especially if they are lengthy operations.
- Constant time and no data dependent operation mitigates computational side–channel attacks; Ed25519's pre–computed base 16 exponentiation helps further, current–measuring side–channel attacks still maybe possible
- Phase 1 (ephemeral key exchange) is not a big problem, as we choose a random secret for each connection
- Phase 2 is modified to use the shared secret s_1 to dilute the operation: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- The secret ring multiplication is a short operation (multiplication mod l instead of curve point by scalar) with much less leakage impact
- DH is faster and transmits less data than signature+verification

Challenge: Side–Channel Attacks

- ECC Diffie–Hellman key exchange formula is $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Operations with secret constants and variables under control of the attacker may leak information, especially if they are lengthy operations.
- Constant time and no data dependent operation mitigates computational side–channel attacks; Ed25519's pre–computed base 16 exponentiation helps further, current–measuring side–channel attacks still maybe possible
- Phase 1 (ephemeral key exchange) is not a big problem, as we choose a random secret for each connection
- Phase 2 is modified to use the shared secret s_1 to dilute the operation: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- The secret ring multiplication is a short operation (multiplication mod l instead of curve point by scalar) with much less leakage impact
- DH is faster and transmits less data than signature+verification

Challenge: Side–Channel Attacks

- ECC Diffie–Hellman key exchange formula is $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Operations with secret constants and variables under control of the attacker may leak information, especially if they are lengthy operations.
- Constant time and no data dependent operation mitigates computational side–channel attacks; Ed25519's pre–computed base 16 exponentiation helps further, current–measuring side–channel attacks still maybe possible
- Phase 1 (ephemeral key exchange) is not a big problem, as we choose a random secret for each connection
- Phase 2 is modified to use the shared secret s_1 to dilute the operation: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- The secret ring multiplication is a short operation (multiplication mod l instead of curve point by scalar) with much less leakage impact
- DH is faster and transmits less data than signature+verification

Challenge: Side–Channel Attacks

- ECC Diffie–Hellman key exchange formula is $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Operations with secret constants and variables under control of the attacker may leak information, especially if they are lengthy operations.
- Constant time and no data dependent operation mitigates computational side–channel attacks; Ed25519's pre–computed base 16 exponentiation helps further, current–measuring side–channel attacks still maybe possible
- Phase 1 (ephemeral key exchange) is not a big problem, as we choose a random secret for each connection
- Phase 2 is modified to use the shared secret s_1 to dilute the operation: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- The secret ring multiplication is a short operation (multiplication mod l instead of curve point by scalar) with much less leakage impact
- DH is faster and transmits less data than signature+verification

Challenge: Side–Channel Attacks

- ECC Diffie–Hellman key exchange formula is $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Operations with secret constants and variables under control of the attacker may leak information, especially if they are lengthy operations.
- Constant time and no data dependent operation mitigates computational side–channel attacks; Ed25519's pre–computed base 16 exponentiation helps further, current–measuring side–channel attacks still maybe possible
- Phase 1 (ephemeral key exchange) is not a big problem, as we choose a random secret for each connection
- Phase 2 is modified to use the shared secret s_1 to dilute the operation: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- The secret ring multiplication is a short operation (multiplication mod l instead of curve point by scalar) with much less leakage impact
- DH is faster and transmits less data than signature+verification

Challenge: Side–Channel Attacks

- ECC Diffie–Hellman key exchange formula is $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Operations with secret constants and variables under control of the attacker may leak information, especially if they are lengthy operations.
- Constant time and no data dependent operation mitigates computational side–channel attacks; Ed25519's pre–computed base 16 exponentiation helps further, current–measuring side–channel attacks still maybe possible
- Phase 1 (ephemeral key exchange) is not a big problem, as we choose a random secret for each connection
- Phase 2 is modified to use the shared secret s_1 to dilute the operation: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- The secret ring multiplication is a short operation (multiplication mod l instead of curve point by scalar) with much less leakage impact
- DH is faster and transmits less data than signature+verification

Challenge: Side–Channel Attacks

- ECC Diffie–Hellman key exchange formula is $s_1 = pk_1 * [sk_2] = pk_2 * [sk_1]$
- Operations with secret constants and variables under control of the attacker may leak information, especially if they are lengthy operations.
- Constant time and no data dependent operation mitigates computational side–channel attacks; Ed25519's pre–computed base 16 exponentiation helps further, current–measuring side–channel attacks still maybe possible
- Phase 1 (ephemeral key exchange) is not a big problem, as we choose a random secret for each connection
- Phase 2 is modified to use the shared secret s_1 to dilute the operation: $s_2 = pk_a * [sk_b * s_1] = pk_b * [sk_a * s_1]$
- The secret ring multiplication is a short operation (multiplication mod l instead of curve point by scalar) with much less leakage impact
- DH is faster and transmits less data than signature+verification

Achieved: 3–Way Handshake

- The setup for an encrypted communication is done with three packets exchanged, no latency overhead to TCP
- The identifying pubkeys are encrypted, so they don't reveal the identity of Alice and Bob to Eve
- All state for Bob is "stored" in packets on the net, so the third packet is the one that actually opens the connection at Bob.
- The third packet also contains a random initialization vector, so if you want to continue a communication with Bob, a single packet is sufficient.
- The time window for re–connecting is currently set to 10s, but can be made significantly longer
- Both sides use "tickets" with self–encrypted messages to store state in the network
- The key for self–encryption is rotated frequently (the mentioned 10s) to invalidate old tickets

Achieved: 3–Way Handshake

- The setup for an encrypted communication is done with three packets exchanged, no latency overhead to TCP
- The identifying pubkeys are encrypted, so they don't reveal the identity of Alice and Bob to Eve
- All state for Bob is "stored" in packets on the net, so the third packet is the one that actually opens the connection at Bob.
- The third packet also contains a random initialization vector, so if you want to continue a communication with Bob, a single packet is sufficient.
- The time window for re–connecting is currently set to 10s, but can be made significantly longer
- Both sides use "tickets" with self–encrypted messages to store state in the network
- The key for self–encryption is rotated frequently (the mentioned 10s) to invalidate old tickets

Achieved: 3–Way Handshake

- The setup for an encrypted communication is done with three packets exchanged, no latency overhead to TCP
- The identifying pubkeys are encrypted, so they don't reveal the identity of Alice and Bob to Eve
- All state for Bob is "stored" in packets on the net, so the third packet is the one that actually opens the connection at Bob.
- The third packet also contains a random initialization vector, so if you want to continue a communication with Bob, a single packet is sufficient.
- The time window for re–connecting is currently set to 10s, but can be made significantly longer
- Both sides use "tickets" with self–encrypted messages to store state in the network
- The key for self–encryption is rotated frequently (the mentioned 10s) to invalidate old tickets

Achieved: 3–Way Handshake

- The setup for an encrypted communication is done with three packets exchanged, no latency overhead to TCP
- The identifying pubkeys are encrypted, so they don't reveal the identity of Alice and Bob to Eve
- All state for Bob is "stored" in packets on the net, so the third packet is the one that actually opens the connection at Bob.
- The third packet also contains a random initialization vector, so if you want to continue a communication with Bob, a single packet is sufficient.
- The time window for re–connecting is currently set to 10s, but can be made significantly longer
- Both sides use "tickets" with self–encrypted messages to store state in the network
- The key for self–encryption is rotated frequently (the mentioned 10s) to invalidate old tickets

Achieved: 3–Way Handshake

- The setup for an encrypted communication is done with three packets exchanged, no latency overhead to TCP
- The identifying pubkeys are encrypted, so they don't reveal the identity of Alice and Bob to Eve
- All state for Bob is "stored" in packets on the net, so the third packet is the one that actually opens the connection at Bob.
- The third packet also contains a random initialization vector, so if you want to continue a communication with Bob, a single packet is sufficient.
- The time window for re–connecting is currently set to 10s, but can be made significantly longer
- Both sides use "tickets" with self–encrypted messages to store state in the network
- The key for self–encryption is rotated frequently (the mentioned 10s) to invalidate old tickets

Achieved: 3–Way Handshake

- The setup for an encrypted communication is done with three packets exchanged, no latency overhead to TCP
- The identifying pubkeys are encrypted, so they don't reveal the identity of Alice and Bob to Eve
- All state for Bob is "stored" in packets on the net, so the third packet is the one that actually opens the connection at Bob.
- The third packet also contains a random initialization vector, so if you want to continue a communication with Bob, a single packet is sufficient.
- The time window for re–connecting is currently set to 10s, but can be made significantly longer
- Both sides use "tickets" with self–encrypted messages to store state in the network
- The key for self–encryption is rotated frequently (the mentioned 10s) to invalidate old tickets

Achieved: 3–Way Handshake

- The setup for an encrypted communication is done with three packets exchanged, no latency overhead to TCP
- The identifying pubkeys are encrypted, so they don't reveal the identity of Alice and Bob to Eve
- All state for Bob is "stored" in packets on the net, so the third packet is the one that actually opens the connection at Bob.
- The third packet also contains a random initialization vector, so if you want to continue a communication with Bob, a single packet is sufficient.
- The time window for re–connecting is currently set to 10s, but can be made significantly longer
- Both sides use "tickets" with self–encrypted messages to store state in the network
- The key for self–encryption is rotated frequently (the mentioned 10s) to invalidate old tickets

Trust&PKI

- Certification Authorities are a broken PKI and trust model
- The simple “remember the key” strategy of SSH is actually better
- First connection requires more attention, e.g. ask the other side to solve a captcha to prove human
- Social networks can provide a network of trust: Trust your friends, and use them to connect you further
- Pubkeys don’t need (nor should) to be public if you only want to be connected with your friends or peers
- Use the pubkey authentication for logins and alike, instead of passwords
- *Shared secrets (e.g. for Socialist Millionaire Protocol) are usually not available: If you have a shared secret, please use symmetric crypto directly.*

Trust&PKI

- Certification Authorities are a broken PKI and trust model
- The simple "remember the key" strategy of SSH is actually better
- First connection requires more attention, e.g. ask the other side to solve a captcha to prove human
- Social networks can provide a network of trust: Trust your friends, and use them to connect you further
- Pubkeys don't need (nor should) to be public if you only want to be connected with your friends or peers
- Use the pubkey authentication for logins and alike, instead of passwords
- *Shared secrets (e.g. for Socialist Millionaire Protocol) are usually not available. If you have a shared secret, please use symmetric crypto directly.*

Trust&PKI

- Certification Authorities are a broken PKI and trust model
- The simple “remember the key” strategy of SSH is actually better
- First connection requires more attention, e.g. ask the other side to solve a captcha to prove human
- Social networks can provide a network of trust: Trust your friends, and use them to connect you further
- Pubkeys don’t need (nor should) to be public if you only want to be connected with your friends or peers
- Use the pubkey authentication for logins and alike, instead of passwords
- *Shared secrets (e.g. for Socialist Millionaire Protocol) are usually not available: If you have a shared secret, please use symmetric crypto directly.*

Trust&PKI

- Certification Authorities are a broken PKI and trust model
- The simple "remember the key" strategy of SSH is actually better
- First connection requires more attention, e.g. ask the other side to solve a captcha to prove human
- Social networks can provide a network of trust: Trust your friends, and use them to connect you further
- Pubkeys don't need (nor should) to be public if you only want to be connected with your friends or peers
- Use the pubkey authentication for logins and alike, instead of passwords
- *Shared secrets (e.g. for Socialist Millionaire Protocol) are usually not available. If you have a shared secret, please use symmetric crypto directly.*

Trust&PKI

- Certification Authorities are a broken PKI and trust model
- The simple “remember the key” strategy of SSH is actually better
- First connection requires more attention, e.g. ask the other side to solve a captcha to prove human
- Social networks can provide a network of trust: Trust your friends, and use them to connect you further
- Pubkeys don’t need (nor should) to be public if you only want to be connected with your friends or peers
- Use the pubkey authentication for logins and alike, instead of passwords
- *Shared secrets (e.g. for Socialist Millionaire Protocol) are usually not available. If you have a shared secret, please use symmetric crypto directly.*

Trust&PKI

- Certification Authorities are a broken PKI and trust model
- The simple “remember the key” strategy of SSH is actually better
- First connection requires more attention, e.g. ask the other side to solve a captcha to prove human
- Social networks can provide a network of trust: Trust your friends, and use them to connect you further
- Pubkeys don’t need (nor should) to be public if you only want to be connected with your friends or peers
- Use the pubkey authentication for logins and alike, instead of passwords
- *Shared secrets (e.g. for Socialist Millionaire Protocol) are usually not available. If you have a shared secret, please use symmetric crypto directly.*

Trust&PKI

- Certification Authorities are a broken PKI and trust model
- The simple "remember the key" strategy of SSH is actually better
- First connection requires more attention, e.g. ask the other side to solve a captcha to prove human
- Social networks can provide a network of trust: Trust your friends, and use them to connect you further
- Pubkeys don't need (nor should) to be public if you only want to be connected with your friends or peers
- Use the pubkey authentication for logins and alike, instead of passwords
- *Shared secrets (e.g. for Socialist Millionaire Protocol) are usually not available: If you have a shared secret, please use symmetric crypto directly.*

Symmetric Crypto

Evaluation of encryption algorithms

- Must do AEAD encryption — authenticate and encrypt/decrypt together
- Widely used candidate: AES in GCM
- Caveats: Galois counter is not a secure hash, but “only” a polynom checksum, which is known to be fragile [2], and security is ≤ 64 bits [3], that paper suggests using GF(p) with $p = 2^{128} + 12451$ to improve the weak key situation
- AES uses a constant key — makes side channel attacks more feasible
- Counter mode is actually a stream cipher, suggests using other steam ciphers
- Compare Dan Bernstein’s xsalsa+poly1305: uses a prime field for the polynome, so this would be a better candidate. Poly1305 still relies on the encryption for security.

Symmetric Crypto

Evaluation of encryption algorithms

- Must do AEAD encryption — authenticate and encrypt/decrypt together
- Widely used candidate: AES in GCM
- Caveats: Galois counter is not a secure hash, but "only" a polynom checksum, which is known to be fragile [2], and security is ≤ 64 bits [3], that paper suggests using GF(p) with $p = 2^{128} + 12451$ to improve the weak key situation
- AES uses a constant key — makes side channel attacks more feasible
- Counter mode is actually a stream cipher, suggests using other steam ciphers
- Compare Dan Bernstein's xsalsa+poly1305: uses a prime field for the polynome, so this would be a better candidate. Poly1305 still relies on the encryption for security.

Symmetric Crypto

Evaluation of encryption algorithms

- Must do AEAD encryption — authenticate and encrypt/decrypt together
- Widely used candidate: AES in GCM
- Caveats: Galois counter is not a secure hash, but “only” a polynom checksum, which is known to be fragile [2], and security is ≤ 64 bits [3], that paper suggests using GF(p) with $p = 2^{128} + 12451$ to improve the weak key situation
- AES uses a constant key — makes side channel attacks more feasible
- Counter mode is actually a stream cipher, suggests using other steam ciphers
- Compare Dan Bernstein’s xsalsa+poly1305: uses a prime field for the polynome, so this would be a better candidate. Poly1305 still relies on the encryption for security.

Symmetric Crypto

Evaluation of encryption algorithms

- Must do AEAD encryption — authenticate and encrypt/decrypt together
- Widely used candidate: AES in GCM
- Caveats: Galois counter is not a secure hash, but “only” a polynom checksum, which is known to be fragile [2], and security is ≤ 64 bits [3], that paper suggests using GF(p) with $p = 2^{128} + 12451$ to improve the weak key situation
- AES uses a constant key — makes side channel attacks more feasible
- Counter mode is actually a stream cipher, suggests using other steam ciphers
- Compare Dan Bernstein’s xsalsa+poly1305: uses a prime field for the polynome, so this would be a better candidate. Poly1305 still relies on the encryption for security.

Symmetric Crypto

Evaluation of encryption algorithms

- Must do AEAD encryption — authenticate and encrypt/decrypt together
- Widely used candidate: AES in GCM
- Caveats: Galois counter is not a secure hash, but "only" a polynom checksum, which is known to be fragile [2], and security is ≤ 64 bits [3], that paper suggests using GF(p) with $p = 2^{128} + 12451$ to improve the weak key situation
- AES uses a constant key — makes side channel attacks more feasible
- Counter mode is actually a stream cipher, suggests using other steam ciphers
- Compare DAN BERNSTEIN's xsalsa+poly1305: uses a prime field for the polynome, so this would be a better candidate. Poly1305 still relies on the encryption for security.

Symmetric Crypto

Evaluation of encryption algorithms

- Must do AEAD encryption — authenticate and encrypt/decrypt together
- Widely used candidate: AES in GCM
- Caveats: Galois counter is not a secure hash, but “only” a polynom checksum, which is known to be fragile [2], and security is ≤ 64 bits [3], that paper suggests using GF(p) with $p = 2^{128} + 12451$ to improve the weak key situation
- AES uses a constant key — makes side channel attacks more feasible
- Counter mode is actually a stream cipher, suggests using other steam ciphers
- Compare Dan Bernstein's xsalsa+poly1305: uses a prime field for the polynome, so this would be a better candidate. Poly1305 still relies on the encryption for security.

Keccak

- Suggestion: Use a strong hash for authentication instead
- Obvious candidate is the SHA–3 winner, Keccak, as this has a very good cryptanalysis
- Even better: Keccak in duplex mode can encrypt while computing the hash (at almost no cost)
- There's no constant key, either: Perfect side–channel protected AEAD operation
- Strength >256 bits, whereas AES–256 suffers from related–key attacks: very good security margin
- Keccak is a universal crypto primitive, with AES in GCM we need three primitives: hash+AES+GHASH
- Keccak is both NIST–approved and (still) NSA–independent. I use Keccak with $r = 1024$ and capacity $c = 576$ as suggested by the Keccak authors.

Keccak

- Suggestion: Use a strong hash for authentication instead
- Obvious candidate is the SHA–3 winner, Keccak, as this has a very good cryptanalysis
- Even better: Keccak in duplex mode can encrypt while computing the hash (at almost no cost)
- There's no constant key, either: Perfect side–channel protected AEAD operation
- Strength >256 bits, whereas AES–256 suffers from related–key attacks: very good security margin
- Keccak is a universal crypto primitive, with AES in GCM we need three primitives: hash+AES+GHASH
- Keccak is both NIST–approved and (still) NSA–independent. I use Keccak with $r = 1024$ and capacity $c = 576$ as suggested by the Keccak authors.

Keccak

- Suggestion: Use a strong hash for authentication instead
- Obvious candidate is the SHA–3 winner, Keccak, as this has a very good cryptanalysis
- Even better: Keccak in duplex mode can encrypt while computing the hash (at almost no cost)
- There's no constant key, either: Perfect side–channel protected AEAD operation
- Strength >256 bits, whereas AES–256 suffers from related–key attacks: very good security margin
- Keccak is a universal crypto primitive, with AES in GCM we need three primitives: hash+AES+GHASH
- Keccak is both NIST–approved and (still) NSA–independent. I use Keccak with $r = 1024$ and capacity $c = 576$ as suggested by the Keccak authors.

Keccak

- Suggestion: Use a strong hash for authentication instead
- Obvious candidate is the SHA–3 winner, Keccak, as this has a very good cryptanalysis
- Even better: Keccak in duplex mode can encrypt while computing the hash (at almost no cost)
- There's no constant key, either: Perfect side–channel protected AEAD operation
- Strength >256 bits, whereas AES–256 suffers from related–key attacks: very good security margin
- Keccak is a universal crypto primitive, with AES in GCM we need three primitives: hash+AES+GHASH
- Keccak is both NIST–approved and (still) NSA–independent. I use Keccak with $r = 1024$ and capacity $c = 576$ as suggested by the Keccak authors.

Keccak

- Suggestion: Use a strong hash for authentication instead
- Obvious candidate is the SHA–3 winner, Keccak, as this has a very good cryptanalysis
- Even better: Keccak in duplex mode can encrypt while computing the hash (at almost no cost)
- There's no constant key, either: Perfect side–channel protected AEAD operation
- Strength >256 bits, whereas AES–256 suffers from related–key attacks: very good security margin
- Keccak is a universal crypto primitive, with AES in GCM we need three primitives: hash+AES+GHASH
- Keccak is both NIST–approved and (still) NSA–independent. I use Keccak with $r = 1024$ and capacity $c = 576$ as suggested by the Keccak authors.

Keccak

- Suggestion: Use a strong hash for authentication instead
- Obvious candidate is the SHA–3 winner, Keccak, as this has a very good cryptanalysis
- Even better: Keccak in duplex mode can encrypt while computing the hash (at almost no cost)
- There's no constant key, either: Perfect side–channel protected AEAD operation
- Strength $>$256 bits, whereas AES–256 suffers from related–key attacks: very good security margin
- Keccak is a universal crypto primitive, with AES in GCM we need three primitives: hash+AES+GHASH
- Keccak is both NIST–approved and (still) NSA–independent: I use Keccak with $r = 1024$ and capacity $c = 576$ as suggested by the Keccak authors.

Keccak

- Suggestion: Use a strong hash for authentication instead
- Obvious candidate is the SHA–3 winner, Keccak, as this has a very good cryptanalysis
- Even better: Keccak in duplex mode can encrypt while computing the hash (at almost no cost)
- There's no constant key, either: Perfect side–channel protected AEAD operation
- Strength $>$256 bits, whereas AES–256 suffers from related–key attacks: very good security margin
- Keccak is a universal crypto primitive, with AES in GCM we need three primitives: hash+AES+GHASH
- Keccak is both NIST–approved and (still) NSA–independent. I use Keccak with $r = 1024$ and capacity $c = 576$ as suggested by the Keccak authors.

Cipher Algorithm Replacement

General idea: Have a selection of cipher suits and replace weak or broken when identified. But this has problems:

1. All encrypted communication is stored away in Utah — if the NSA finds a weakness, they can decrypt the history
2. People are lazy and only implement the easiest and fastest cipher — this is the one broken first
3. Hardware accelerators and even software is often very difficult to update due to the “never change a running system” principle
4. The operator or the end user does not have the know–how to make the right choice of a cipher algorithm — this is guru level

Therefore, the chosen cipher algorithm must last for a long time, and all systems must have an upgrade path

Cipher Algorithm Replacement

General idea: Have a selection of cipher suits and replace weak or broken when identified. But this has problems:

1. All encrypted communication is stored away in Utah — if the NSA finds a weakness, they can decrypt the history
2. People are lazy and only implement the easiest and fastest cipher — this is the one broken first
3. Hardware accelerators and even software is often very difficult to update due to the "never change a running system" principle
4. The operator or the end user does not have the know–how to make the right choice of a cipher algorithm — this is guru level

Therefore, the chosen cipher algorithm must last for a long time, and all systems must have an upgrade path

Cipher Algorithm Replacement

General idea: Have a selection of cipher suits and replace weak or broken when identified. But this has problems:

1. All encrypted communication is stored away in Utah — if the NSA finds a weakness, they can decrypt the history
2. People are lazy and only implement the easiest and fastest cipher — this is the one broken first
3. Hardware accelerators and even software is often very difficult to update due to the "never change a running system" principle
4. The operator or the end user does not have the know–how to make the right choice of a cipher algorithm — this is guru level

Therefore, the chosen cipher algorithm must last for a long time, and all systems must have an upgrade path

Cipher Algorithm Replacement

General idea: Have a selection of cipher suits and replace weak or broken when identified. But this has problems:

1. All encrypted communication is stored away in Utah — if the NSA finds a weakness, they can decrypt the history
2. People are lazy and only implement the easiest and fastest cipher — this is the one broken first
3. Hardware accelerators and even software is often very difficult to update due to the “never change a running system” principle
4. The operator or the end user does not have the know–how to make the right choice of a cipher algorithm — this is guru level

Therefore, the chosen cipher algorithm must last for a long time, and all systems must have an upgrade path

Cipher Algorithm Replacement

General idea: Have a selection of cipher suits and replace weak or broken when identified. But this has problems:

1. All encrypted communication is stored away in Utah — if the NSA finds a weakness, they can decrypt the history
2. People are lazy and only implement the easiest and fastest cipher — this is the one broken first
3. Hardware accelerators and even software is often very difficult to update due to the “never change a running system” principle
4. The operator or the end user does not have the know–how to make the right choice of a cipher algorithm — this is guru level

Therefore, the chosen cipher algorithm must last for a long time, and all systems must have an upgrade path

Cipher Algorithm Replacement

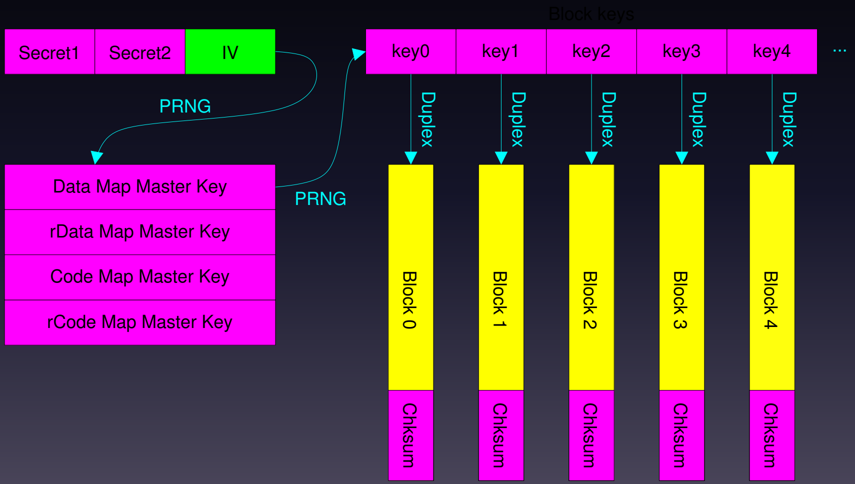
General idea: Have a selection of cipher suits and replace weak or broken when identified. But this has problems:

1. All encrypted communication is stored away in Utah — if the NSA finds a weakness, they can decrypt the history
2. People are lazy and only implement the easiest and fastest cipher — this is the one broken first
3. Hardware accelerators and even software is often very difficult to update due to the “never change a running system” principle
4. The operator or the end user does not have the know–how to make the right choice of a cipher algorithm — this is guru level

Therefore, the chosen cipher algorithm must last for a long time, and all systems must have an upgrade path

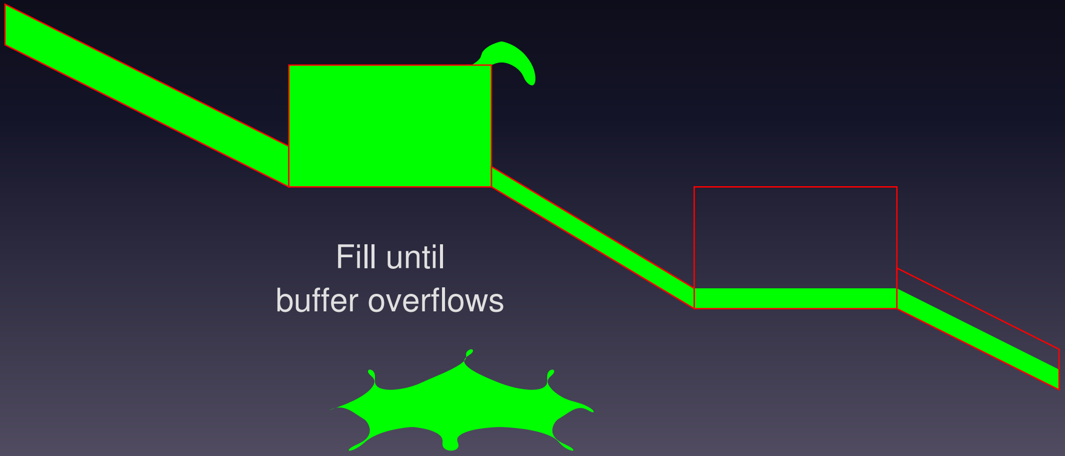
Key Usage

All keys are one–time–use only!



Flow Control (Broken)

- TCP fills the buffer, until a packet has to be dropped, instead of reducing rate before. Name of the symptom: “Buffer bloat”. But buffering is essential for good network performance.



Alternatives?

- LEDBAT tries to achieve a low, constant delay: Works, but not good on fairness
- CurveCP's flow control is still "a lot of research"
- Therefore, something new has to be done

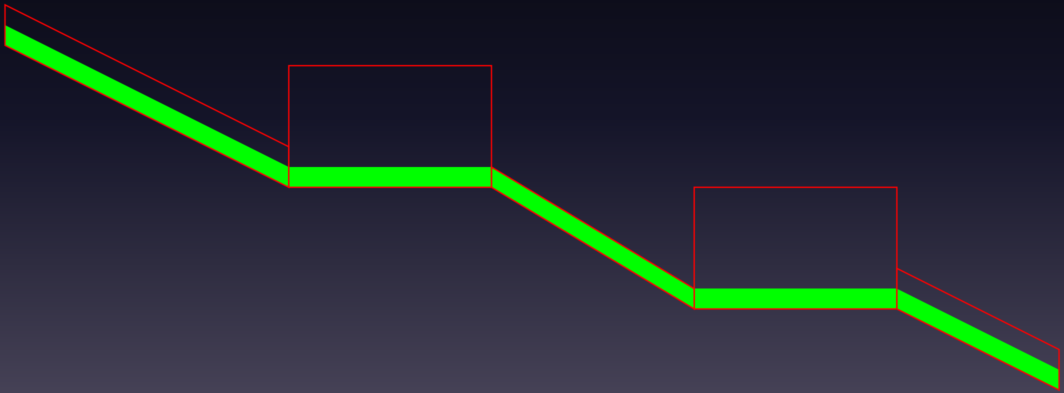


Figure : That's how proper flow control should look like

Alternatives?

- LEDBAT tries to achieve a low, constant delay: Works, but not good on fairness
- CurveCP's flow control is still "a lot of research"
- Therefore, something new has to be done

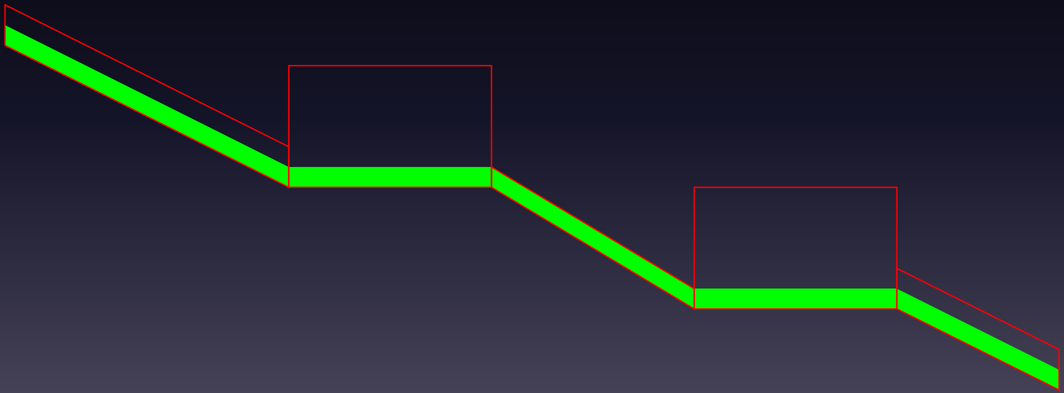


Figure : That's how proper flow control should look like

Alternatives?

- LEDBAT tries to achieve a low, constant delay: Works, but not good on fairness
- CurveCP's flow control is still "a lot of research"
- Therefore, something new has to be done

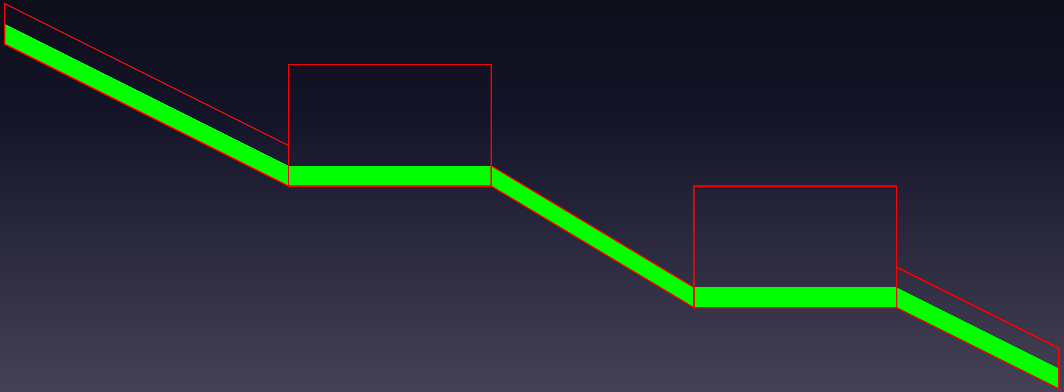


Figure : That's how proper flow control should look like

„Buffer Bloat“

- Retransmits are making the situation worse in case of congestions and therefore should be avoided
- Riddle: How big should the buffer be, under the assumption that the bandwidth is used optimally, the bottleneck is on the other side of the connection, and a second data stream is opened up?
- Answer: about half the round trip delay, which are inevitably filled before any reaction is possible
- Buffers are good, but you shouldn't fill them up to the brim
- The problem is inherent in the TCP protocol, but since Windows XP did not provide window scaling, the per-connection buffer limit was 64k for most connections on the internet for quite a long time.

„Buffer Bloat“

- Retransmits are making the situation worse in case of congestions and therefore should be avoided
- Riddle: How big should the buffer be, under the assumption that the bandwidth is used optimally, the bottleneck is on the other side of the connection, and a second data stream is opened up?
- Answer: about half the round trip delay, which are inevitably filled before any reaction is possible
- Buffers are good, but you shouldn't fill them up to the brim
- The problem is inherent in the TCP protocol, but since Windows XP did not provide window scaling, the per-connection buffer limit was 64k for most connections on the Internet for quite a long time.

„Buffer Bloat“

- Retransmits are making the situation worse in case of congestions and therefore should be avoided
- Riddle: How big should the buffer be, under the assumption that the bandwidth is used optimally, the bottleneck is on the other side of the connection, and a second data stream is opened up?
- Answer: about half the round trip delay, which are inevitably filled before any reaction is possible
- Buffers are good, but you shouldn't fill them up to the brim
- The problem is inherent in the TCP protocol, but since Windows XP did not provide window scaling, the per–connection buffer limit was 64k for most connections on the internet for quite a long time.

„Buffer Bloat“

- Retransmits are making the situation worse in case of congestions and therefore should be avoided
- Riddle: How big should the buffer be, under the assumption that the bandwidth is used optimally, the bottleneck is on the other side of the connection, and a second data stream is opened up?
- Answer: about half the round trip delay, which are inevitably filled before any reaction is possible
- Buffers are good, but you shouldn’t fill them up to the brim
- The problem is inherent in the TCP protocol, but since Windows XP did not provide window scaling, the per-connection buffer limit was 64k for most connections on the internet for quite a long time.

„Buffer Bloat“

- Retransmits are making the situation worse in case of congestions and therefore should be avoided
- Riddle: How big should the buffer be, under the assumption that the bandwidth is used optimally, the bottleneck is on the other side of the connection, and a second data stream is opened up?
- Answer: about half the round trip delay, which are inevitably filled before any reaction is possible
- Buffers are good, but you shouldn't fill them up to the brim
- The problem is inherent in the TCP protocol, but since Windows XP did not provide window scaling, the per–connection buffer limit was 64k for most connections on the Internet for quite a long time.

net2o Flow Control

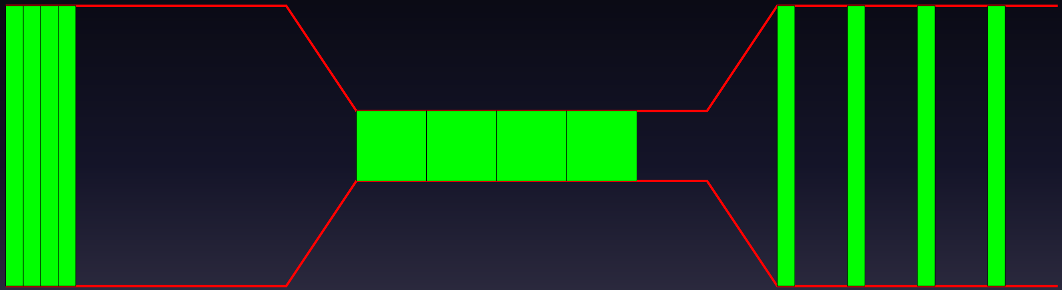


Figure : Measure the bottleneck using a burst of packets

Client Measures, Server Sets Rate

Client recores the *time* of the first and last packet in a burst, and calculates the achieved rate for received packets, extrapolating to the achievable rate including the dropped packets. This results in the requested *rate*.

$$rate := \Delta t * \frac{burstlen}{packets}$$

Server would simply use this rate

Client Measures, Server Sets Rate

Client recores the *time* of the first and last packet in a burst, and calculates the achieved rate for received packets, extrapolating to the achievable rate including the dropped packets. This results in the requested *rate*.

$$rate := \Delta t * \frac{burstlen}{packets}$$

Server would simply use this rate

Fairness

Fairness means that concurrent connections achieve about the same data rate, sharing the same line in a fair way.

- Ideally, a router/switch would schedule buffered packets round–robin, giving each connection a fair share of the bandwidth. That would change the calculated rate appropriately, and also be a big relief for current TCP buffer bloat symptoms, as each connection would have its private buffer to fill up.
- Unfortunately, routers use a single FIFO policy for all connections
- Finding a sufficiently stable algorithm to provide fairness
- We want to adopt to new situations as fast as possible, there's no point in anything slow. Especially on wireless connections, achievable rate changes are not only related to traffic.

Fairness

Fairness means that concurrent connections achieve about the same data rate, sharing the same line in a fair way.

- Ideally, a router/switch would schedule buffered packets round–robin, giving each connection a fair share of the bandwidth. That would change the calculated rate appropriately, and also be a big relief for current TCP buffer bloat symptoms, as each connection would have its private buffer to fill up.
- Unfortunately, routers use a single FIFO policy for all connections
- Finding a sufficiently stable algorithm to provide fairness
- We want to adopt to new situations as fast as possible, there's no point in anything slow. Especially on wireless connections, achievable rate changes are not only related to traffic.

Fairness

Fairness means that concurrent connections achieve about the same data rate, sharing the same line in a fair way.

- Ideally, a router/switch would schedule buffered packets round–robin, giving each connection a fair share of the bandwidth. That would change the calculated rate appropriately, and also be a big relief for current TCP buffer bloat symptoms, as each connection would have its private buffer to fill up.
- Unfortunately, routers use a single FIFO policy for all connections
- Finding a sufficiently stable algorithm to provide fairness
- We want to adopt to new situations as fast as possible, there's no point in anything slow. Especially on wireless connections, achievable rate changes are not only related to traffic.

Fairness

Fairness means that concurrent connections achieve about the same data rate, sharing the same line in a fair way.

- Ideally, a router/switch would schedule buffered packets round–robin, giving each connection a fair share of the bandwidth. That would change the calculated rate appropriately, and also be a big relief for current TCP buffer bloat symptoms, as each connection would have its private buffer to fill up.
- Unfortunately, routers use a single FIFO policy for all connections
- Finding a sufficiently stable algorithm to provide fairness
- We want to adopt to new situations as fast as possible, there's no point in anything slow. Especially on wireless connections, achievable rate changes are not only related to traffic.

Fairness

Fairness means that concurrent connections achieve about the same data rate, sharing the same line in a fair way.

- Ideally, a router/switch would schedule buffered packets round–robin, giving each connection a fair share of the bandwidth. That would change the calculated rate appropriately, and also be a big relief for current TCP buffer bloat symptoms, as each connection would have its private buffer to fill up.
- Unfortunately, routers use a single FIFO policy for all connections
- Finding a sufficiently stable algorithm to provide fairness
- We want to adopt to new situations as fast as possible, there's no point in anything slow. Especially on wireless connections, achievable rate changes are not only related to traffic.

net2o Flow Control — Fair Router

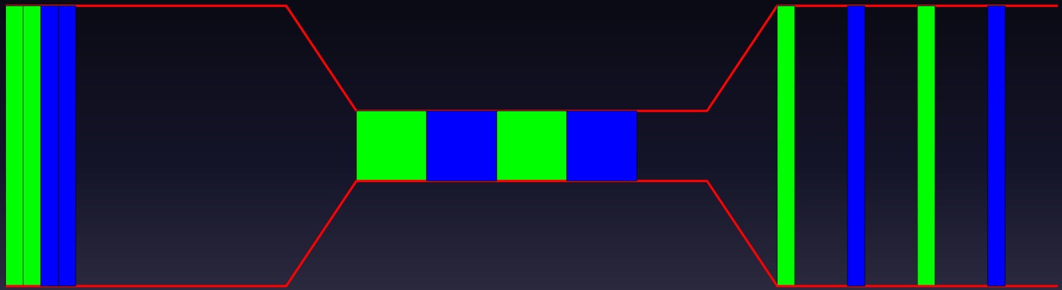


Figure : Fair queuing results in correct measurement of available bandwidth

net2o Flow Control — FIFO Router

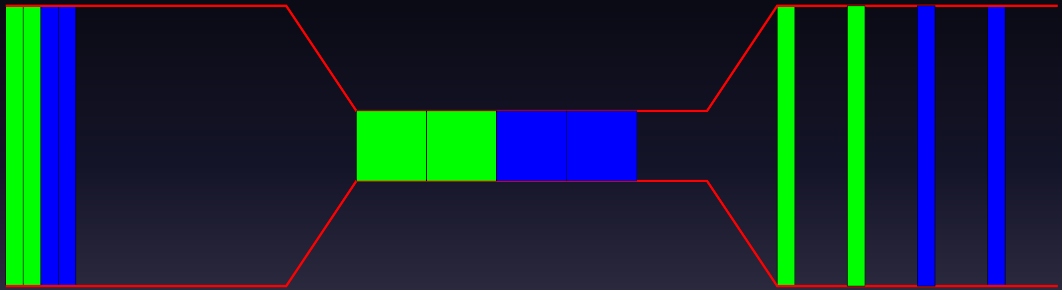


Figure : Unfair FIFO queuing results in twice the available bandwidth calculated

Fairness I

- To improve stability of unfair queued packets, we need to improve that P regulator (proportional to measured rate) to a full PID regulator
- The integral part is the accumulated slack (in the buffer), which we want to keep low, and the D part is growing/reducing this slack from one measurement to the next
- We use both parts to decrease the sending rate, and thereby achieve better fairness
- The I part is used to exponentially lengthen the rate Δt with increasing slack up to a maximum factor of 16.

$$s_{exp} = 2^{\frac{slack}{T}} \quad \text{where } T = \max(10ms, \max(slacks))$$

Fairness I

- To improve stability of unfair queued packets, we need to improve that P regulator (proportional to measured rate) to a full PID regulator
- The integral part is the accumulated slack (in the buffer), which we want to keep low, and the D part is growing/reducing this slack from one measurement to the next
- We use both parts to decrease the sending rate, and thereby achieve better fairness
- The I part is used to exponentially lengthen the rate Δt with increasing slack up to a maximum factor of 16.

$$s_{exp} = 2^{\frac{slack}{T}} \quad \text{where } T = \max(10ms, \max(slacks))$$

Fairness I

- To improve stability of unfair queued packets, we need to improve that P regulator (proportional to measured rate) to a full PID regulator
- The integral part is the accumulated slack (in the buffer), which we want to keep low, and the D part is growing/reducing this slack from one measurement to the next
- We use both parts to decrease the sending rate, and thereby achieve better fairness
- The I part is used to exponentially lengthen the rate Δt with increasing slack up to a maximum factor of 16.

$$s_{exp} = 2^{\frac{slack}{T}} \quad \text{where } T = \max(10ms, \max(slacks))$$

Fairness I

- To improve stability of unfair queued packets, we need to improve that P regulator (proportional to measured rate) to a full PID regulator
- The integral part is the accumulated slack (in the buffer), which we want to keep low, and the D part is growing/reducing this slack from one measurement to the next
- We use both parts to decrease the sending rate, and thereby achieve better fairness
- The I part is used to exponentially lengthen the rate Δt with increasing slack up to a maximum factor of 16.

$$s_{exp} = 2^{\frac{slack}{T}} \quad \text{where } T = \max(10ms, \max(slacks))$$

Fairness I

- To improve stability of unfair queued packets, we need to improve that P regulator (proportional to measured rate) to a full PID regulator
- The integral part is the accumulated slack (in the buffer), which we want to keep low, and the D part is growing/reducing this slack from one measurement to the next
- We use both parts to decrease the sending rate, and thereby achieve better fairness
- The I part is used to exponentially lengthen the rate Δt with increasing slack up to a maximum factor of 16.

$$s_{exp} = 2^{\frac{slack}{T}} \quad \text{where } T = \max(10ms, \max(slacks))$$

Fairness D

- To measure the differential term, we measure how much the slack grows (a Δt value) from the first to the last burst we do for one measurement cycle (4 bursts by default, first packet to first packet of each burst)
- This is multiplied by the total packets in flight (head of the sender queue vs. acknowledged packet), divided by the packets within the measured interval
- A low–pass filter is applied to the obtained D to prevent from speeding up too fast, with one round trip delay as time constant
- $\max(slacks)/10ms$ is used to determine how aggressive this algorithm is
- Add the obtained Δt both to the rate's Δt for one burst sequence and wait that time before starting the next burst sequence

Fairness D

- To measure the differential term, we measure how much the slack grows (a Δt value) from the first to the last burst we do for one measurement cycle (4 bursts by default, first packet to first packet of each burst)
- This is multiplied by the total packets in flight (head of the sender queue vs. acknowledged packet), divided by the packets within the measured interval
- A low–pass filter is applied to the obtained D to prevent from speeding up too fast, with one round trip delay as time constant
- $\max(slacks)/10ms$ is used to determine how aggressive this algorithm is
- Add the obtained Δt both to the rate's Δt for one burst sequence and wait that time before starting the next burst sequence

Fairness D

- To measure the differential term, we measure how much the slack grows (a Δt value) from the first to the last burst we do for one measurement cycle (4 bursts by default, first packet to first packet of each burst)
- This is multiplied by the total packets in flight (head of the sender queue vs. acknowledged packet), divided by the packets within the measured interval
- A low–pass filter is applied to the obtained D to prevent from speeding up too fast, with one round trip delay as time constant
- $\max(slacks)/10ms$ is used to determine how aggressive this algorithm is
- Add the obtained Δt both to the rate's Δt for one burst sequence and wait that time before starting the next burst sequence

Fairness D

- To measure the differential term, we measure how much the slack grows (a Δt value) from the first to the last burst we do for one measurement cycle (4 bursts by default, first packet to first packet of each burst)
- This is multiplied by the total packets in flight (head of the sender queue vs. acknowledged packet), divided by the packets within the measured interval
- A low–pass filter is applied to the obtained D to prevent from speeding up too fast, with one round trip delay as time constant
- $\max(slacks)/10ms$ is used to determine how aggressive this algorithm is
- Add the obtained Δt both to the rate's Δt for one burst sequence and wait that time before starting the next burst sequence

Fairness D

- To measure the differential term, we measure how much the slack grows (a Δt value) from the first to the last burst we do for one measurement cycle (4 bursts by default, first packet to first packet of each burst)
- This is multiplied by the total packets in flight (head of the sender queue vs. acknowledged packet), divided by the packets within the measured interval
- A low–pass filter is applied to the obtained D to prevent from speeding up too fast, with one round trip delay as time constant
- $\max(slacks)/10ms$ is used to determine how aggressive this algorithm is
- Add the obtained Δt both to the rate's Δt for one burst sequence and wait that time before starting the next burst sequence.

VDSL

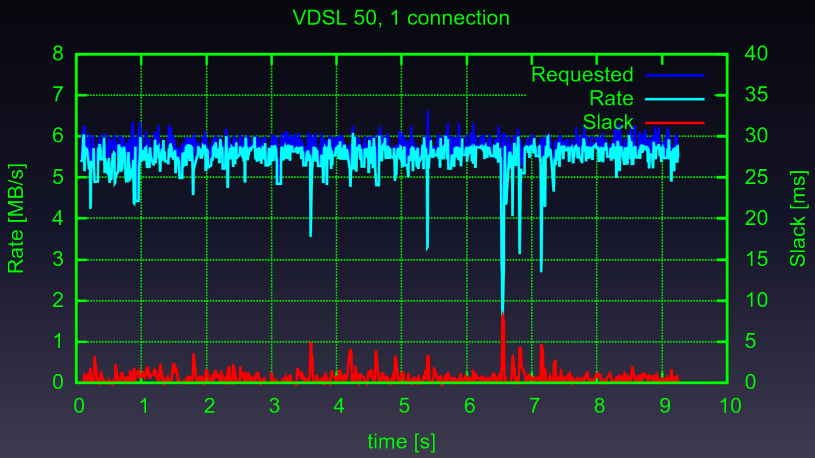


Figure : One connection on a VDSL–50 line

VDSL, Congestion

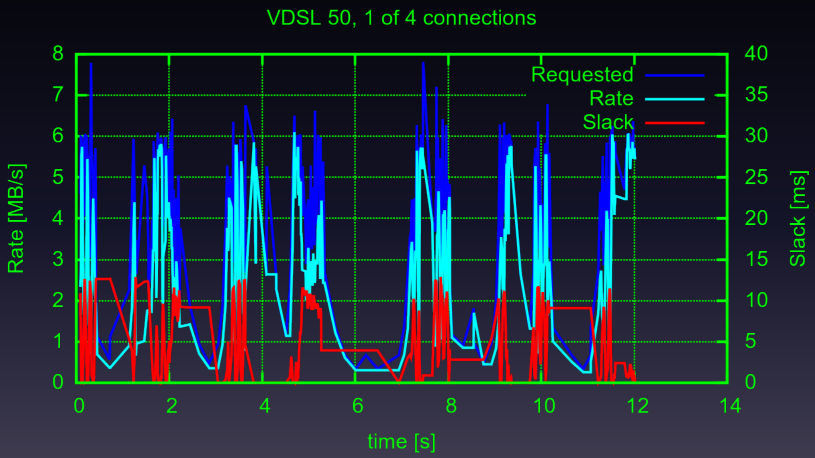


Figure : One of four connections on a VDSL–50 line

Unreliable Air Cable (WLAN)

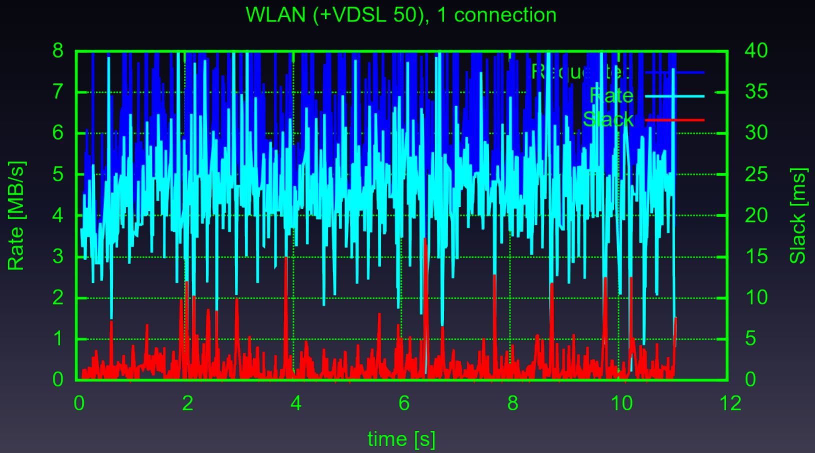


Figure : Single connection using WLAN

Unreliable Air Cable, Congestion

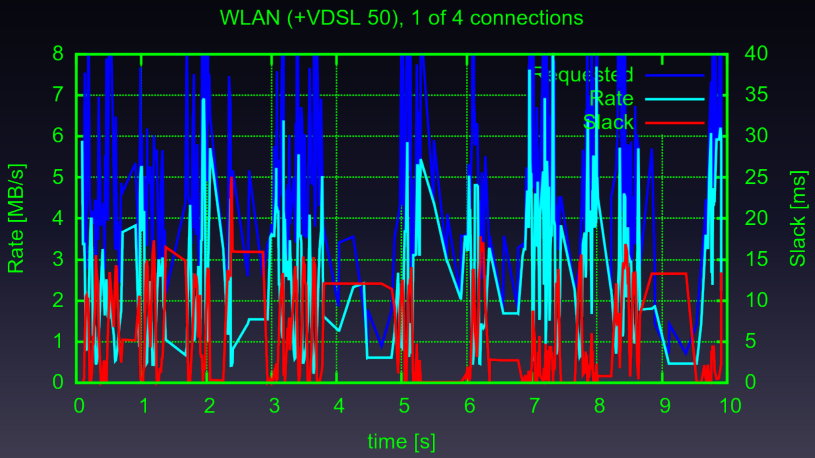


Figure : One of four connections using WLAN

LAN, 1GBE

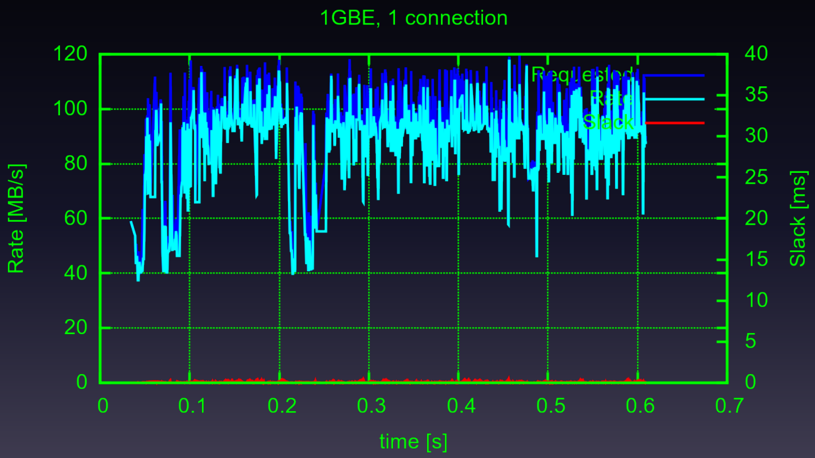


Figure : Single connection using 1GBE

LAN 1GBE, Congestion (4 servers)

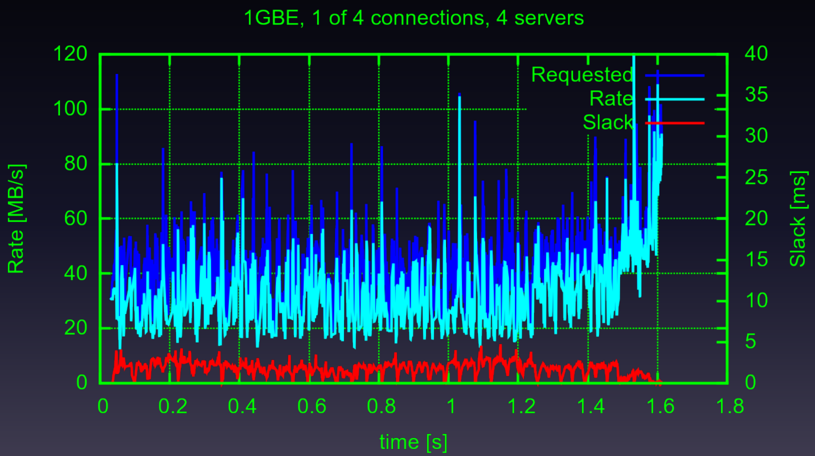


Figure : One of four connections using 1GBE

LAN 1GBE, Congestion (1 server)

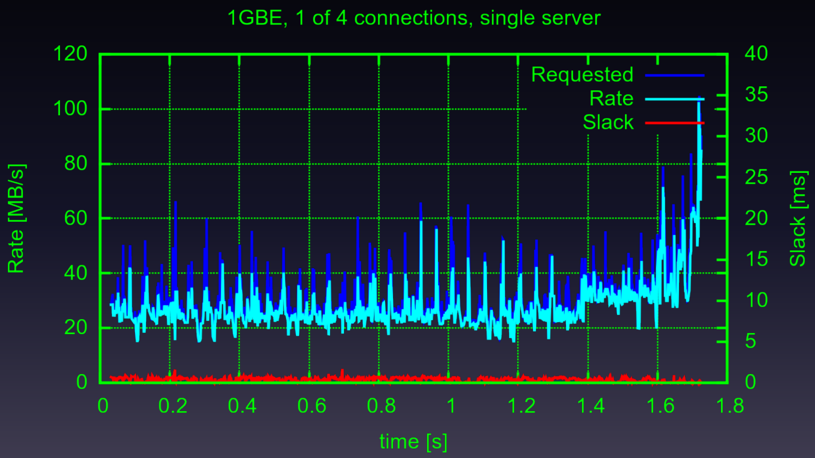


Figure : One of four connections using 1GBE, fair queuing

Flow Control Conclusion

- Flow control works, but a change in the router FIFO policy can help things a lot
- The primary flow control approach is completely different from other approaches: Measure the available bandwidth!
- Scalability to very slow connections is still lacking: bursts are 8 packets long.
- Congested traffic without fair queuing not satisfying

Flow Control Conclusion

- Flow control works, but a change in the router FIFO policy can help things a lot
- The primary flow control approach is completely different from other approaches: Measure the available bandwidth!
- Scalability to very slow connections is still lacking: bursts are 8 packets long.
- Congested traffic without fair queuing not satisfying

Flow Control Conclusion

- Flow control works, but a change in the router FIFO policy can help things a lot
- The primary flow control approach is completely different from other approaches: Measure the available bandwidth!
- Scalability to very slow connections is still lacking: bursts are 8 packets long.
- Congested traffic without fair queuing not satisfying

Flow Control Conclusion

- Flow control works, but a change in the router FIFO policy can help things a lot
- The primary flow control approach is completely different from other approaches: Measure the available bandwidth!
- Scalability to very slow connections is still lacking: bursts are 8 packets long.
- Congested traffic without fair queuing not satisfying

Why change the FIFO policy?

- Pushing the problem to the place where it occurs — the router/switch — makes the solution much easier
- Fair queuing solves the problem of TCP buffer bloat *now* (for connections competing with the bloated TCP connection)
- Mitigates DoS attacks (flooding a node with traffic)
- Not essential to deployment, but this result help people who work on improving router algorithms

Why change the FIFO policy?

- Pushing the problem to the place where it occurs — the router/switch — makes the solution much easier
- Fair queuing solves the problem of TCP buffer bloat *now* (for connections competing with the bloated TCP connection)
- Mitigates DoS attacks (flooding a node with traffic)
- Not essential to deployment, but this result help people who work on improving router algorithms

Why change the FIFO policy?

- Pushing the problem to the place where it occurs — the router/switch — makes the solution much easier
- Fair queuing solves the problem of TCP buffer bloat *now* (for connections competing with the bloated TCP connection)
- Mitigates DoS attacks (flooding a node with traffic)
- Not essential to deployment, but this result help people who work on improving router algorithms

Why change the FIFO policy?

- Pushing the problem to the place where it occurs — the router/switch — makes the solution much easier
- Fair queuing solves the problem of TCP buffer bloat *now* (for connections competing with the bloated TCP connection)
- Mitigates DoS attacks (flooding a node with traffic)
- Not essential to deployment, but this result help people who work on improving router algorithms

Data and Commands

- Data of several files/streams can be transferred interleaving, so a single connection can do multiple things in parallel
- Commands are send in command blocks, i.e. there is not just one command per block, but a sequence of commands!
- Commands are encoded like protobuf, i.e. 7 bits per byte, and if the MSB of the byte is 1, there's another byte to follow (allowing arbitrary many commands)
- The command "machine" is a stack architecture.

Data and Commands

- Data of several files/streams can be transferred interleaving, so a single connection can do multiple things in parallel
- Commands are send in command blocks, i.e. there is not just one command per block, but a sequence of commands!
- Commands are encoded like protobuf, i.e. 7 bits per byte, and if the MSB of the byte is 1, there's another byte to follow (allowing arbitrary many commands)
- The command "machine" is a stack architecture.

Data and Commands

- Data of several files/streams can be transferred interleaving, so a single connection can do multiple things in parallel
- Commands are send in command blocks, i.e. there is not just one command per block, but a sequence of commands!
- Commands are encoded like protobuf, i.e. 7 bits per byte, and if the MSB of the byte is 1, there's another byte to follow (allowing arbitrary many commands)
- The command "machine" is a stack architecture.

Data and Commands

- Data of several files/streams can be transferred interleaving, so a single connection can do multiple things in parallel
- Commands are send in command blocks, i.e. there is not just one command per block, but a sequence of commands!
- Commands are encoded like protobuf, i.e. 7 bits per byte, and if the MSB of the byte is 1, there's another byte to follow (allowing arbitrary many commands)
- The command "machine" is a stack architecture.

Example: Connection Request

```
"pk1" $, receive-tmpkey
nest[ timestamp1 lit, set-rtdelay gen-reply request-done ]nest $,
push-$ push' nest
tmpkey-request key-request
base lit, csize lit, dsize lit, map-request
```

Example: Download three files

```
net2o-code
"Download test" $, type cr ( see-me )
get-ip $400 blocksize! $400 blockalign! stat( request-stats )
"net2o.fs" 0 lit, 0 lit, open-tracked-file
"data/2011-05-13_11-26-57-small.jpg" 0 lit, 1 lit, open-tracked-file
"data/2011-05-20_17-01-12-small.jpg" 0 lit, 2 lit, open-tracked-file
gen-total slurp-all-tracked-blocks send-chunks
0 lit, tag-reply
end-code
```

Example: Answer to this request

```
net2o-code
x" 3600000000000000000000000000000000019ED2" $, set-ip
 $E373 lit, 0 lit, track-size
$134299FF6F829E62 lit, $1A4 lit, 0 lit, set-stat
$9C65C lit, 1 lit, track-size
$130AFDAE900C649E lit, $1A4 lit, 1 lit, set-stat
$9D240 lit, 2 lit, track-size
$130AFDAE92CA4E25 lit, $1A4 lit, 2 lit, set-stat
$148000 lit, set-total
 $E373 lit, 0 lit, track-seek
$79000 lit, 1 lit, track-seek
$78C00 lit, 2 lit, track-seek
0 lit, ack-reply
end-code
```

Distributed Data

- Following the “everything is a file” principle, every data object is a file
- Data objects are accessed by their hash. The associated metadata are “tags”
- Metadata is organized as a distributed prefix hash tree
- Efficient distribution of data is important!

Distributed Data

- Following the “everything is a file” principle, every data object is a file
- Data objects are accessed by their hash. The associated metadata are “tags”
- Metadata is organized as a distributed prefix hash tree
- Efficient distribution of data is important!

Distributed Data

- Following the “everything is a file” principle, every data object is a file
- Data objects are accessed by their hash. The associated metadata are “tags”
- Metadata is organized as a distributed prefix hash tree
- Efficient distribution of data is important!

Distributed Data

- Following the “everything is a file” principle, every data object is a file
- Data objects are accessed by their hash. The associated metadata are “tags”
- Metadata is organized as a distributed prefix hash tree
- Efficient distribution of data is important!

Efficient Data Distribution

Puzzle: How efficient can you distribute data (e.g. a video stream) in a P2P network? Assume all peers are equal, and have the capacity to upload one stream in realtime.

- Obvious topology: The bucket chain — this shows that each node feeds the data through — a 1:1 relation of what you get to what you send
- bucket chain: $O(n)$ latency, $O\left(\frac{1}{n}\right)$ robustness (each node can break the chain)
- Suggestion: Tree structure instead of chain, e.g. a quad–tree. The root divides the data into four parts, each going down one branch of the tree. The leafs distribute the data to the other three branches of the tree
- For the quad–tree case, each node has only 8 neighbors: 4 sources and 4 sinks

Efficient Data Distribution

Puzzle: How efficient can you distribute data (e.g. a video stream) in a P2P network? Assume all peers are equal, and have the capacity to upload one stream in realtime.

- Obvious topology: The bucket chain — this shows that each node feeds the data through — a 1:1 relation of what you get to what you send
- bucket chain: $O(n)$ latency, $O\left(\frac{1}{n}\right)$ robustness (each node can break the chain)
- Suggestion: Tree structure instead of chain, e.g. a quad–tree. The root divides the data into four parts, each going down one branch of the tree. The leafs distribute the data to the other three branches of the tree
- For the quad–tree case, each node has only 8 neighbors: 4 sources and 4 sinks

Efficient Data Distribution

Puzzle: How efficient can you distribute data (e.g. a video stream) in a P2P network? Assume all peers are equal, and have the capacity to upload one stream in realtime.

- Obvious topology: The bucket chain — this shows that each node feeds the data through — a 1:1 relation of what you get to what you send
- bucket chain: $O(n)$ latency, $O\left(\frac{1}{n}\right)$ robustness (each node can break the chain)
- Suggestion: Tree structure instead of chain, e.g. a quad–tree. The root divides the data into four parts, each going down one branch of the tree. The leafs distribute the data to the other three branches of the tree
- For the quad–tree case, each node has only 8 neighbors: 4 sources and 4 sinks

Efficient Data Distribution

Puzzle: How efficient can you distribute data (e.g. a video stream) in a P2P network? Assume all peers are equal, and have the capacity to upload one stream in realtime.

- Obvious topology: The bucket chain — this shows that each node feeds the data through — a 1:1 relation of what you get to what you send
- bucket chain: $O(n)$ latency, $O\left(\frac{1}{n}\right)$ robustness (each node can break the chain)
- Suggestion: Tree structure instead of chain, e.g. a quad–tree. The root divides the data into four parts, each going down one branch of the tree. The leafs distribute the data to the other three branches of the tree
- For the quad–tree case, each node has only 8 neighbors: 4 sources and 4 sinks

Efficient Data Distribution

Puzzle: How efficient can you distribute data (e.g. a video stream) in a P2P network? Assume all peers are equal, and have the capacity to upload one stream in realtime.

- Obvious topology: The bucket chain — this shows that each node feeds the data through — a 1:1 relation of what you get to what you send
- bucket chain: $O(n)$ latency, $O\left(\frac{1}{n}\right)$ robustness (each node can break the chain)
- Suggestion: Tree structure instead of chain, e.g. a quad–tree. The root divides the data into four parts, each going down one branch of the tree. The leafs distribute the data to the other three branches of the tree
- For the quad–tree case, each node has only 8 neighbors: 4 sources and 4 sinks

Tree Distribution Network

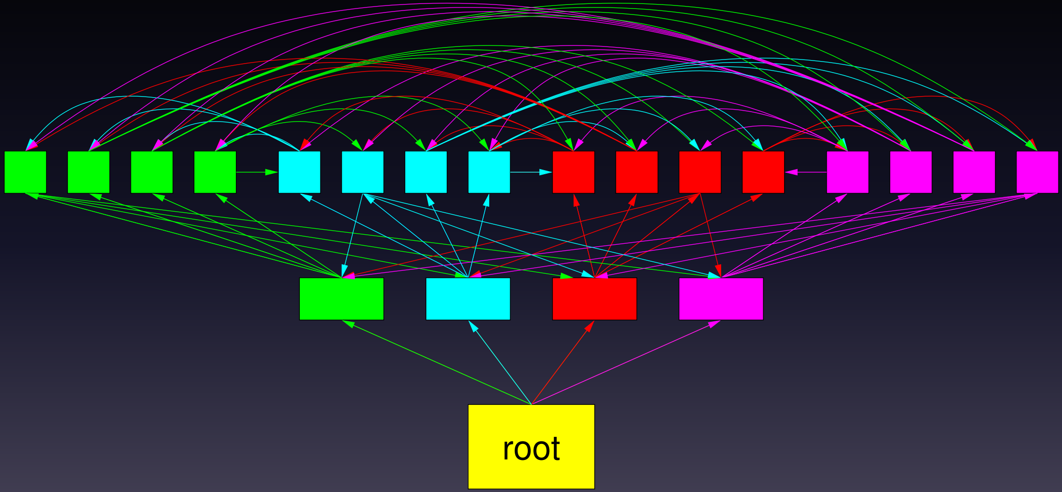


Figure : Avalanche distribution with quad–tree of depth 2

Possible Performance

- Trees with a bigger base reduce latency. Example: To transfer a Justin Bieber tweet to 50 million followers, a binary tree needs 25.5 hops on average, a quad–tree 12.8 hops, and an oct–tree 8.5 hops.
- A typical domestic (inside e.g. Germany) hop–to–hop time is just 20ms. International hops can be in the order of 250ms. Assuming there is only one international hop in the chain, the latency to distribute Justin Bieber's babbling is typically just 500ms in a quad–tree.

  Rule of thumb: *bandwidth* = *latency*, i.e. if it takes 20ms from hop to hop, each node should replicate data for 20ms — if we make the tree wider, the linear effort of replicating data will dominate transfer time, if we make the tree more narrow, the hop–to–hop time will dominate.
- The tree–like graph greatly reduces the number of nodes to know

Possible Performance

- Trees with a bigger base reduce latency. Example: To transfer a Justin Bieber tweet to 50 million followers, a binary tree needs 25.5 hops on average, a quad–tree 12.8 hops, and an oct–tree 8.5 hops.
- A typical domestic (inside e.g. Germany) hop–to–hop time is just 20ms. International hops can be in the order of 250ms. Assuming there is only one international hop in the chain, the latency to distribute Justin Bieber's babbling is typically just 500ms in a quad–tree.

  Rule of thumb: *bandwidth* = *latency*, i.e. if it takes 20ms from hop to hop, each node should replicate data for 20ms — if we make the tree wider, the linear effort of replicating data will dominate transfer time, if we make the tree more narrow, the hop–to–hop time will dominate.

- The tree–like graph greatly reduces the number of nodes to know

Possible Performance

- Trees with a bigger base reduce latency. Example: To transfer a Justin Bieber tweet to 50 million followers, a binary tree needs 25.5 hops on average, a quad–tree 12.8 hops, and an oct–tree 8.5 hops.
- A typical domestic (inside e.g. Germany) hop–to–hop time is just 20ms. International hops can be in the order of 250ms. Assuming there is only one international hop in the chain, the latency to distribute Justin Bieber's babbling is typically just 500ms in a quad–tree.
- Rule of thumb: $\mathit{bandwidth} = \mathit{latency}$, i.e. if it takes 20ms from hop to hop, each node should replicate data for 20ms — if we make the tree wider, the linear effort of replicating data will dominate transfer time, if we make the tree more narrow, the hop–to–hop time will dominate.
- The tree–like graph greatly reduces the number of nodes to know

Possible Performance

- Trees with a bigger base reduce latency. Example: To transfer a Justin Bieber tweet to 50 million followers, a binary tree needs 25.5 hops on average, a quad–tree 12.8 hops, and an oct–tree 8.5 hops.
- A typical domestic (inside e.g. Germany) hop–to–hop time is just 20ms. International hops can be in the order of 250ms. Assuming there is only one international hop in the chain, the latency to distribute Justin Bieber's babbling is typically just 500ms in a quad–tree.
- Rule of thumb: $bandwidth = latency$, i.e. if it takes 20ms from hop to hop, each node should replicate data for 20ms — if we make the tree wider, the linear effort of replicating data will dominate transfer time, if we make the tree more narrow, the hop–to–hop time will dominate.
- The tree–like graph greatly reduces the number of nodes to know

Distributed Prefix Hash Tree

- Most DHT approaches have poor performanc
- Prefix Hash Trees use a quite large base
- Only a few queries necessary to query an extremely large data base
- Suggestion: Active instantaneous replication of all changed data using the avalanche tree mentioned above

Distributed Prefix Hash Tree

- Most DHT approaches have poor performanc
- Prefix Hash Trees use a quite large base
- Only a few queries necessary to query an extremely large data base
- Suggestion: Active instantaneous replication of all changed data using the avalanche tree mentioned above

Distributed Prefix Hash Tree

- Most DHT approaches have poor performanc
- Prefix Hash Trees use a quite large base
- Only a few queries necessary to query an extremely large data base
- Suggestion: Active instantaneous replication of all changed data using the avalanche tree mentioned above

Distributed Prefix Hash Tree

- Most DHT approaches have poor performanc
- Prefix Hash Trees use a quite large base
- Only a few queries necessary to query an extremely large data base
- Suggestion: Active instantaneous replication of all changed data using the avalanche tree mentioned above

Metadata Privacy

- A global distributed hash tree is 100% observable by anybody with enough money
- Private trees shared only between a group of people: "dark trees in a dark forrest"
- Use different identities for distinct groups (one for your friends, one for your work, one for sharing pr0n), each one only known to that group: "dark social graph"
- Your public ID is only for first contacts and the things you want to publish (i.e. PR)
- Use neighborhood relationships to limit spread of data — e.g. a node participating in a data distribution tree may only be known to the peer nodes in that tree

Metadata Privacy

- A global distributed hash tree is 100% observable by anybody with enough money
- Private trees shared only between a group of people: “dark trees in a dark forrest”
- Use different identities for distinct groups (one for your friends, one for your work, one for sharing pr0n), each one only known to that group: “dark social graph”
- Your public ID is only for first contacts and the things you want to publish (i.e. PR)
- Use neighborhood relationships to limit spread of data — e.g. a node participating in a data distribution tree may only be known to the peer nodes in that tree

Metadata Privacy

- A global distributed hash tree is 100% observable by anybody with enough money
- Private trees shared only between a group of people: “dark trees in a dark forrest”
- Use different identities for distinct groups (one for your friends, one for your work, one for sharing pr0n), each one only known to that group: “dark social graph”
- Your public ID is only for first contacts and the things you want to publish (i.e. PR)
- Use neighborhood relationships to limit spread of data — e.g. a node participating in a data distribution tree may only be known to the peer nodes in that tree

Metadata Privacy

- A global distributed hash tree is 100% observable by anybody with enough money
- Private trees shared only between a group of people: “dark trees in a dark forrest”
- Use different identities for distinct groups (one for your friends, one for your work, one for sharing pr0n), each one only known to that group: “dark social graph”
- Your public ID is only for first contacts and the things you want to publish (i.e. PR)
- Use neighborhood relationships to limit spread of data — e.g. a node participating in a data distribution tree may only be known to the peer nodes in that tree

Metadata Privacy

- A global distributed hash tree is 100% observable by anybody with enough money
- Private trees shared only between a group of people: “dark trees in a dark forrest”
- Use different identities for distinct groups (one for your friends, one for your work, one for sharing pr0n), each one only known to that group: “dark social graph”
- Your public ID is only for first contacts and the things you want to publish (i.e. PR)
- Use neighborhood relationships to limit spread of data — e.g. a node participating in a data distribution tree may only be known to the peer nodes in that tree

Content or Apps?

- The current web is defined by content — web apps (JavaScript) are an afterthough
- Therefore, the application logic is usually on the server side
- This doesn't work for a P2P network!

Content is structured text, images, videos, music, etc.

Content or Apps?

- The current web is defined by content — web apps (JavaScript) are an afterthough
- Therefore, the application logic is usually on the server side
- This doesn't work for a P2P network!

Content is structured text, images, videos, music, etc.

Content or Apps?

- The current web is defined by content — web apps (JavaScript) are an afterthough
- Therefore, the application logic is usually on the server side
- This doesn't work for a P2P network!

Content is structured text, images, videos, music, etc.

Content or Apps?

- The current web is defined by content — web apps (JavaScript) are an afterthough
- Therefore, the application logic is usually on the server side
- This doesn't work for a P2P network!
- Content is structured text, images, videos, music, etc.

App–Centric World

- There's a phenomenon I call "Turing creep": Every sufficiently complex system contains a user–accessible Turing–complete language
- Corollary: Every efficient sufficiently complex system can execute native machine code
- The application logic is to present the data; data itself is as above: structured text, images, videos, music, etc.
- Executing (especially efficient) code from the net raises obvious questions about security

App–Centric World

- There's a phenomenon I call "Turing creep": Every sufficiently complex system contains a user–accessible Turing–complete language
- Corollary: Every efficient sufficiently complex system can execute native machine code
- The application logic is to present the data; data itself is as above: structured text, images, videos, music, etc.
- Executing (especially efficient) code from the net raises obvious questions about security

App–Centric World

- There's a phenomenon I call "Turing creep": Every sufficiently complex system contains a user–accessible Turing–complete language
- Corollary: Every efficient sufficiently complex system can execute native machine code
- The application logic is to present the data; data itself is as above: structured text, images, videos, music, etc.
- Executing (especially efficient) code from the net raises obvious questions about security

App–Centric World

- There's a phenomenon I call "Turing creep": Every sufficiently complex system contains a user–accessible Turing–complete language
- Corollary: Every efficient sufficiently complex system can execute native machine code
- The application logic is to present the data; data itself is as above: structured text, images, videos, music, etc.
- Executing (especially efficient) code from the net raises obvious questions about security

How to securely execute code?

There are several options tried; as usual, things are broken:

1. Execute code in a controlled secure VM, see for example Java. This is broken by design, as securing something from the inside doesn't work.
2. Execute code in a sandbox. This has shown as more robust, depending on how complex the outside of the sandbox is.
3. Public inspection of code. This is how the open source world works, but the underhanded C contest shows that inspection is tricky.
4. Scan for known evil code. This is the security industry's approach, and it is not working.
5. Code signing can work together with public inspection — but using it for accountability doesn't work.

Therefore the choice is to sandbox public inspected code.

How to securely execute code?

There are several options tried; as usual, things are broken:

1. Execute code in a controlled secure VM, see for example Java. This is broken by design, as securing something from the inside doesn't work.
2. Execute code in a sandbox. This has shown as more robust, depending on how complex the outside of the sandbox is.
3. Public inspection of code. This is how the open source world works, but the underhanded C contest shows that inspection is tricky.
4. Scan for known evil code. This is the security industry's approach, and it is not working.
5. Code signing can work together with public inspection — but using it for accountability doesn't work.

Therefore the choice is to sandbox public inspected code.

How to securely execute code?

There are several options tried; as usual, things are broken:

1. Execute code in a controlled secure VM, see for example Java. This is broken by design, as securing something from the inside doesn't work.
2. Execute code in a sandbox. This has shown as more robust, depending on how complex the outside of the sandbox is.
3. Public inspection of code. This is how the open source world works, but the underhanded C contest shows that inspection is tricky.
4. Scan for known evil code. This is the security industry's approach, and it is not working.
5. Code signing can work together with public inspection — but using it for accountability doesn't work.

Therefore the choice is to sandbox public inspected code.

How to securely execute code?

There are several options tried; as usual, things are broken:

1. Execute code in a controlled secure VM, see for example Java. This is broken by design, as securing something from the inside doesn't work.
2. Execute code in a sandbox. This has shown as more robust, depending on how complex the outside of the sandbox is.
3. Public inspection of code. This is how the open source world works, but the underhanded C contest shows that inspection is tricky.
4. Scan for known evil code. This is the security industry's approach, and it is not working.
5. Code signing can work together with public inspection — but using it for accountability doesn't work.

Therefore the choice is to sandbox public inspected code.

How to securely execute code?

There are several options tried; as usual, things are broken:

1. Execute code in a controlled secure VM, see for example Java. This is broken by design, as securing something from the inside doesn't work.
2. Execute code in a sandbox. This has shown as more robust, depending on how complex the outside of the sandbox is.
3. Public inspection of code. This is how the open source world works, but the underhanded C contest shows that inspection is tricky.
4. Scan for known evil code. This is the security industry's approach, and it is not working.
5. Code signing can work together with public inspection — but using it for accountability doesn't work.

Therefore the choice is to sandbox public inspected code.

How to securely execute code?

There are several options tried; as usual, things are broken:

1. Execute code in a controlled secure VM, see for example Java. This is broken by design, as securing something from the inside doesn't work.
2. Execute code in a sandbox. This has shown as more robust, depending on how complex the outside of the sandbox is.
3. Public inspection of code. This is how the open source world works, but the underhanded C contest shows that inspection is tricky.
4. Scan for known evil code. This is the security industry's approach, and it is not working.
5. Code signing can work together with public inspection — but using it for accountability doesn't work

Therefore the choice is to sandbox public inspected code.

How to securely execute code?

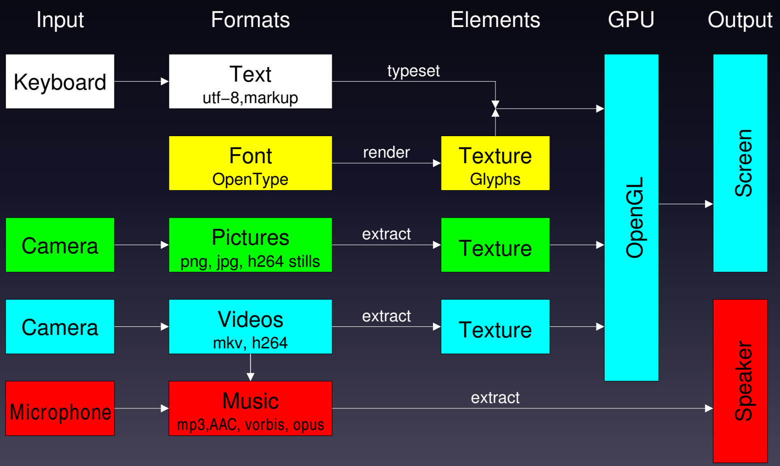
There are several options tried; as usual, things are broken:

1. Execute code in a controlled secure VM, see for example Java. This is broken by design, as securing something from the inside doesn't work.
2. Execute code in a sandbox. This has shown as more robust, depending on how complex the outside of the sandbox is.
3. Public inspection of code. This is how the open source world works, but the underhanded C contest shows that inspection is tricky.
4. Scan for known evil code. This is the security industry's approach, and it is not working.
5. Code signing can work together with public inspection — but using it for accountability doesn't work

Therefore the choice is to sandbox public inspected code.

Formats&Requirements

How to display things



Why OpenGL?

OpenGL can do everything

OpenGL renders:

1. Triangles, lines, points — simple components
2. Textures and gradients
3. and uses shader programs — the most powerful thing in OpenGL from 2.0.

Real requirement: visualization of *any* data. OpenGL can do that.

Why OpenGL?

OpenGL can do everything

OpenGL renders:

1. Triangles, lines, points — simple components
2. Textures and gradients
3. and uses shader programs — the most powerful thing in OpenGL from 2.0.

Real requirement: visualization of *any* data. OpenGL can do that.

Why OpenGL?

OpenGL can do everything

OpenGL renders:

1. Triangles, lines, points — simple components
2. Textures and gradients
3. and uses shader programs — the most powerful thing in OpenGL from 2.0.

Real requirement: visualization of *any* data. OpenGL can do that.

Why OpenGL?

OpenGL can do everything

OpenGL renders:

1. Triangles, lines, points — simple components
2. Textures and gradients
3. and uses shader programs — the most powerful thing in OpenGL from 2.0.

Real requirement: visualization of *any* data. OpenGL can do that.

Why OpenGL?

OpenGL can do everything

OpenGL renders:

1. Triangles, lines, points — simple components
2. Textures and gradients
3. and uses shader programs — the most powerful thing in OpenGL from 2.0.

Real requirement: visualization of *any* data. OpenGL can do that.

How to connect the media?

Lemma: every glue logic will become Turing complete

- currently used glue: HTML+CSS+JavaScript
- containers with Flash, Java, ActiveX, PDF, Google's NaCl...
- conclusion: use a powerful tool right from start!

browser: run–time and development tool for applications

How to connect the media?

Lemma: every glue logic will become Turing complete

- currently used glue: HTML+CSS+JavaScript
- containers with Flash, Java, ActiveX, PDF, Google's NaCl...
- conclusion: use a powerful tool right from start!

  browser: run–time and development tool for applications

How to connect the media?

Lemma: every glue logic will become Turing complete

- currently used glue: HTML+CSS+JavaScript
- containers with Flash, Java, ActiveX, PDF, Google's NaCl...
- conclusion: use a powerful tool right from start!

  browser: run–time and development tool for applications

How to connect the media?

Lemma: every glue logic will become Turing complete

- currently used glue: HTML+CSS+JavaScript
- containers with Flash, Java, ActiveX, PDF, Google's NaCl...
- conclusion: use a powerful tool right from start!
- browser: run–time and development tool for applications

Frameworks

- libsoil for images (PNG+JPEG loading into a texture)
- freetype-gl for fonts (TrueType/OpenType into a texture)
- OpenMAX on Android, gstreamer on Linux: videos into a texture
- MINOΣ2: Lightweight OpenGL–based widget library in Forth (still a lot of work in progress)

Frameworks

- libsoil for images (PNG+JPEG loading into a texture)
- freetype-gl for fonts (TrueType/OpenType into a texture)
- OpenMAX on Android, gstreamer on Linux: videos into a texture
- MINOΣ2: Lightweight OpenGL–based widget library in Forth (still a lot of work in progress)

Frameworks

- libsoil for images (PNG+JPEG loading into a texture)
- freetype-gl for fonts (TrueType/OpenType into a texture)
- OpenMAX on Android, gstreamer on Linux: videos into a texture
- MINOΣ2: Lightweight OpenGL–based widget library in Forth (still a lot of work in progress)

Frameworks

- libsoil for images (PNG+JPEG loading into a texture)
- freetype-gl for fonts (TrueType/OpenType into a texture)
- OpenMAX on Android, gstreamer on Linux: videos into a texture
- MINOΣ2: Lightweight OpenGL–based widget library in Forth (still a lot of work in progress)

Use Cases, Funding&Law

"Layer 8" is the human in front of the screen. What will people use this for?

1. Sharing photos and videos
2. Chat & video telephony
3. News, opinions, scientific papers, sharing knowledge
4. Online shopping

Use Cases, Funding&Law

"Layer 8" is the human in front of the screen. What will people use this for?

1. Sharing photos and videos
2. Chat & video telephony
3. News, opinions, scientific papers, sharing knowledge
4. Online shopping

Use Cases, Funding&Law

“Layer 8” is the human in front of the screen. What will people use this for?

1. Sharing photos and videos
2. Chat & video telephony
3. News, opinions, scientific papers, sharing knowledge
4. Online shopping

Use Cases, Funding&Law

"Layer 8" is the human in front of the screen. What will people use this for?

1. Sharing photos and videos
2. Chat & video telephony
3. News, opinions, scientific papers, sharing knowledge
4. Online shopping

Use Cases, Funding&Law

"Layer 8" is the human in front of the screen. What will people use this for?

1. Sharing photos and videos
2. Chat & video telephony
3. News, opinions, scientific papers, sharing knowledge
4. Online shopping

What’s the problem with those?

From the point of view of Hans–Peter Uhl

1. “Pirated” videos and music (Hollywood will sue me), child porn+terrorism
2. Molested children
3. Dissident opinions, leaks
4. Sex, Drugs & Weapons (Rock’n’Roll see 1.)

What's the problem with those?

From the point of view of Hans–Peter Uhl

1. "Pirated" videos and music (Hollywood will sue me), child porn+terrorism
2. Molested children
3. Dissident opinions, leaks
4. Sex, Drugs & Weapons (Rock'n'Roll see 1.)

What's the problem with those?

From the point of view of Hans–Peter Uhl

1. "Pirated" videos and music (Hollywood will sue me), child porn+terrorism
2. Molested children
3. Dissident opinions, leaks
4. Sex, Drugs & Weapons (Rock'n'Roll see 1.)

What's the problem with those?

From the point of view of Hans–Peter Uhl

1. "Pirated" videos and music (Hollywood will sue me), child porn+terrorism
2. Molested children
3. Dissident opinions, leaks
4. Sex, Drugs & Weapons (Rock'n'Roll see 1.)

What's the problem with those?

From the point of view of Hans–Peter Uhl

1. "Pirated" videos and music (Hollywood will sue me), child porn+terrorism
2. Molested children
3. Dissident opinions, leaks
4. Sex, Drugs & Weapons (Rock'n'Roll see 1.)

What to do about it?

- I don't want to know what my users want to do, nor do they want me to know
- Public shared stuff is possible to track down — copyright is a political problem, the technology we build is there for making copies, primarily for cat videos and duck–face selfies
- net2o is not primarily targeted at people who have "something to hide", it is intended to offer state–of–the–art privacy protection to everybody without performance and usability drawbacks
- Normal criminal investigation has still a very good chance to catch criminals

What to do about it?

- I don't want to know what my users want to do, nor do they want me to know
- Public shared stuff is possible to track down — copyright is a political problem, the technology we build is there for making copies, primarily for cat videos and duck–face selfies
- net2o is not primarily targeted at people who have "something to hide", it is intended to offer state–of–the–art privacy protection to everybody without performance and usability drawbacks
- Normal criminal investigation has still a very good chance to catch criminals

What to do about it?

- I don't want to know what my users want to do, nor do they want me to know
- Public shared stuff is possible to track down — copyright is a political problem, the technology we build is there for making copies, primarily for cat videos and duck–face selfies
- net2o is not primarily targeted at people who have "something to hide", it is intended to offer state–of–the–art privacy protection to everybody without performance and usability drawbacks
- Normal criminal investigation has still a very good chance to catch criminals

What to do about it?

- I don't want to know what my users want to do, nor do they want me to know
- Public shared stuff is possible to track down — copyright is a political problem, the technology we build is there for making copies, primarily for cat videos and duck–face selfies
- net2o is not primarily targeted at people who have "something to hide", it is intended to offer state–of–the–art privacy protection to everybody without performance and usability drawbacks
- Normal criminal investigation has still a very good chance to catch criminals

How to fund it

- Companies are not very trustworthy: If the NSA pays the bill, they do whatever the NSA wants. However, this problem also exists for FOSS projects to some extent (e.g. Dual_EC_DRBG was implemented in OpenSSL after receiving funding from an unnamed company).
- Kickstarter funding looks a lot more interesting, and can work for FOSS projects, too
- Ad–based funding is pretty problematic if you don't want to sell customer's data one way or another
- Storage space "in the cloud" comes with the responsibility to take copyright violations down
- The whole economy behind such a network is huge, the cost for developing are tiny compared to that

How to fund it

- Companies are not very trustworthy: If the NSA pays the bill, they do whatever the NSA wants. However, this problem also exists for FOSS projects to some extent (e.g. Dual_EC_DRBG was implemented in OpenSSL after receiving funding from an unnamed company).
- Kickstarter funding looks a lot more interesting, and can work for FOSS projects, too
- Ad–based funding is pretty problematic if you don't want to sell customer's data one way or another
- Storage space "in the cloud" comes with the responsibility to take copyright violations down
- The whole economy behind such a network is huge, the cost for developing is tiny compared to that

How to fund it

- Companies are not very trustworthy: If the NSA pays the bill, they do whatever the NSA wants. However, this problem also exists for FOSS projects to some extent (e.g. Dual_EC_DRBG was implemented in OpenSSL after receiving funding from an unnamed company).
- Kickstarter funding looks a lot more interesting, and can work for FOSS projects, too
- Ad–based funding is pretty problematic if you don't want to sell customer's data one way or another
- Storage space "in the cloud" comes with the responsibility to take copyright violations down
- [illegible] tiny compared to that

How to fund it

- Companies are not very trustworthy: If the NSA pays the bill, they do whatever the NSA wants. However, this problem also exists for FOSS projects to some extent (e.g. Dual_EC_DRBG was implemented in OpenSSL after receiving funding from an unnamed company).
- Kickstarter funding looks a lot more interesting, and can work for FOSS projects, too
- Ad–based funding is pretty problematic if you don't want to sell customer's data one way or another
- Storage space "in the cloud" comes with the responsibility to take copyright violations down
- [illegible] tiny compared to that

How to fund it

- Companies are not very trustworthy: If the NSA pays the bill, they do whatever the NSA wants. However, this problem also exists for FOSS projects to some extent (e.g. Dual_EC_DRBG was implemented in OpenSSL after receiving funding from an unnamed company).
- Kickstarter funding looks a lot more interesting, and can work for FOSS projects, too
- Ad–based funding is pretty problematic if you don't want to sell customer's data one way or another
- Storage space "in the cloud" comes with the responsibility to take copyright violations down
- The whole economy behind such a network is huge; the cost for developing are tiny compared to that

Adoption

- People have nothing to hide, so security is *not* a primary feature
- Ease of use is a key for success
- Adoption rate usually is exponential with a quite constant replication factor, i.e. people will complain about “empty wasteland”
- People like to feel good — that’s why Facebook has only a “like” button
- Censorship is not liked: Platforms like Facebook&Co. take down sexual content and copyrighted stuff. I won’t (because I can’t, by design)
- Filter bubble instead of censorship: Don’t be friend with people who share things you don’t like

Adoption

- People have nothing to hide, so security is *not* a primary feature
- Ease of use is a key for success
- Adoption rate usually is exponential with a quite constant replication factor, i.e. people will complain about “empty wasteland”
- People like to feel good — that’s why Facebook has only a “like” button
- Censorship is not liked: Platforms like Facebook&Co. take down sexual content and copyrighted stuff. I won’t (because I can’t, by design)
- Filter bubble instead of censorship: Don’t be friend with people who share things you don’t like

Adoption

- People have nothing to hide, so security is *not* a primary feature
- Ease of use is a key for success
- Adoption rate usually is exponential with a quite constant replication factor, i.e. people will complain about “empty wasteland”
- People like to feel good — that’s why Facebook has only a “like” button
- Censorship is not liked: Platforms like Facebook&Co. take down sexual content and copyrighted stuff. I won’t (because I can’t, by design)
- Filter bubble instead of censorship: Don’t be friend with people who share things you don’t like

Adoption

- People have nothing to hide, so security is *not* a primary feature
- Ease of use is a key for success
- Adoption rate usually is exponential with a quite constant replication factor, i.e. people will complain about "empty wasteland"
- People like to feel good — that's why Facebook has only a "like" button
- Censorship is not liked: Platforms like Facebook&Co. take down sexual content and copyrighted stuff. I won't (because I can't, by design)
- Filter bubble instead of censorship: Don't be friend with people who share things you don't like

Adoption

- People have nothing to hide, so security is *not* a primary feature
- Ease of use is a key for success
- Adoption rate usually is exponential with a quite constant replication factor, i.e. people will complain about "empty wasteland"
- People like to feel good — that's why Facebook has only a "like" button
- Censorship is not liked: Platforms like Facebook&Co. take down sexual content and copyrighted stuff. I won't (because I can't, by design)
- Filter bubble instead of censorship: Don't be friend with people who share things you don't like

Adoption

- People have nothing to hide, so security is *not* a primary feature
- Ease of use is a key for success
- Adoption rate usually is exponential with a quite constant replication factor, i.e. people will complain about “empty wasteland”
- People like to feel good — that’s why Facebook has only a “like” button
- Censorship is not liked: Platforms like Facebook&Co. take down sexual content and copyrighted stuff. I won’t (because I can’t, by design)
- Filter bubble instead of censorship: Don’t be friend with people who share things you don’t like

For Further Reading I

- BERND PAYSAN
  net2o source repository
  http://fossil.net2o.de/net2o
- SHAY GUERON, VLAD KRASNOV
  The fragility of AES-GCM authentication algorithm
  http://eprint.iacr.org/2013/157.pdf
- MARKKU-JUHANI O. SAARINEN
  GCM, GHASH and Weak Keys
  http://www.ecrypt.eu.org/hash2011/proceedings/hash2011_03.pdf