Undocumented server behavior considered harmful

Rob ten Berge, Sjors Gielen, Hans Harmannij

14 juli 2010

Inhoudsopgave

1 Inleiding

Tijdens de groei van het internet kwam "als vanzelf"de wens om via internet te communiceren; dit manifesteerde zich al snel in het ontstaan van e-mail en het Simple Mail Transfer Protocol (SMTP) [1]. SMTP wordt tot de dag van vandaag gebruikt voor het versturen van e-mail, een bericht dat van één "e-mailadres"naar het andere wordt verstuurd. Wanneer een antwoord wordt verstuurd op een e-mail, is dat antwoord zelf ook weer een e-mail, waarin de oude e-mail vaak ook wordt geciteerd.

Tegenwoordig heeft bijna ieder bedrijf zelf een e-mailserver staan, een soort postkamer: een machine die alle post van buitenaf ontvangt, en ervoor zorgt dat het in de goede vakjes terechtkomt. Er is een enorme verzameling software voor deze e-mailservers; het lijstje echt populaire software is zoals vaak klein, en omvat bijvoorbeeld Postfix en Microsoft IIS. E-mail is decentraal opgezet, wat wil zeggen dat er niet één "hoofdcomputer" is die alle e-mail regelt. Bedrijven hebben vaak één of meerdere e-mailservers of maken gebruik van de servers van andere bedrijven. Het is één van de grote speerpunten van e-mail dat niemand er "de baas over is" en het zeer simpel in te stellen is: zet een e-mailserver

op en verbind er een domeinnaam aan, en je bent deel van het wereldwijde e-mailnetwerk.

In mei 2009 maakte het grote zoek- en advertentiebedrijf Google, ook al bekend om Google Mail en Google Talk, bekend dat zij een alternatief op het dertig jaar oude e-mail en SMTP hadden ontwikkeld, en het zou Google Wave heten. Google heeft zelf software ontwikkeld voor Google Wave, en om optimale spreiding teweeg te brengen, hebben zij hun software open-source gemaakt en alle specificaties vrijgegeven. Het systeem is daarnaast net als e-mail decentraal opgezet. De zogenaamde "Waves"kunnen, nadat zij verzonden zijn, gewijzigd worden; antwoorden worden "toegevoegd" aan de Wave, in plaats van als een nieuwe Wave verstuurd.

Omdat wij met z'n drieën zeer gecharmeerd waren van het idee van Google Wave, en vinden dat juist één van de grote speerpunten van e-mail de aanwezigheid van vele e-mailserver-software is, hebben wij besloten om zelf een Google Wave-server te schrijven. Daarvoor hebben wij onderzoek moeten doen en experimenten moeten uitvoeren, omdat de documentatie nog te onaf en vaag was om direct te kunnen implementeren. Soms was de documentatie die er was zelfs incorrect of zichzelf tegensprekend.

In deze publicatie hopen wij

- te documenteren welke problemen wij tegengekomen zijn,
- te documenteren welke oplossingen wij daarvoor gekozen hebben en waarom,
- en te laten zien dat wij nu een goede server hebben die deel kan worden van het Google Wave-netwerk.

Als andere producten na ons onderzoeksproject, stellen wij twee verzamelingen software beschikbaar voor vrij gebruik onder de GNU GPL2-licentie [2]. De eerste verzameling is een set veranderingen (patches) aan de codebibliotheek 'QXmpp' waar wij gebruik van hebben gemaakt, hierover later meer. De tweede verzameling is Google Wave-specifieke code in een los programma "ruwaved", de RUWave Google Wave-server.

Onderzoeksvraag:

Hoe maken we een waveserver die kan communiceren met andere waveservers?

Onderdelen:

- Hoe communiceren wave-servers met elkaar?
- Wat moet een waveserver verder kunnen?(Nadruk ligt op het communiceren in het onderzoek)
- Hoe implementeren wij dat precies?

2 Theoretisch kader

2.1 XMPP

Het Extensible Messaging and Presence Protocol (XMPP) is een generiek protocol voor het uitwisselen van data tussen twee partijen dat gebruik maakt van XML. De eerste versies waren rond 1999 uitgewerkt in de Jabber opensource-gemeenschap, daarna werd in 2002 de "XMPP Working Group" belast met het verder uitwerken van het protocol voor het bruikbaar maken ervan, tot in october 2004 de specificatie van het Core gedeelte van XMPP [3] en de specificatie van het instant messaging gedeelte [4] werden gepubliceerd. Ten tijde van schrijven is de negende draft van RFC3920bis [5] beschikbaar gesteld, wat aangeeft dat XMPP nog steeds vol in ontwikkeling is.

Het XMPP protocol onderscheidt een client partij en een server partij door bij het beginnen van een verbinding een verschillende namespace binnen het stream element te definieren. Daarna wordt een features element uitgewisseld, aangenomen dat beide partijen de huidige versie van 1.0 ondersteunen. Aan de hand van het features element, dat de andere partij laat weten wat de versturende kant ondersteunt qua versleuteling, identificatie en authenticatie, wordt bepaalt welke features overeenkomen bij beide partijen voor alle drie. Daarna onderneemt de ontvangende partij actie om voor zover mogelijk versleuteling van de onderlinge communicatie samen met de andere partij te regelen.

In het geval dat twee servers met elkaar verbinden vindt daarna een zwakke vorm van identificatie plaats met behulp van een algoritme genaamd Dialback. Dialback maakt gebruik van het feit dat wanneer servers beginnen met een stream ze verplicht zijn een domein te vermelden, en dat de specificatie van Dialback eist dat XMPP servers op hetzelfde domein een gedeeld geheim hebben dat versleuteld uitgewisseld kan worden met XMPP. In de algemene situatie zijn er vier servers die bij een complete cyclus van Dialback betrokken zijn, in eerste instantie zijn er de verbindende server en de ontvangende server. Beide hebben een domein waarvan ze beweren vandaan te komen, zeg ruwave.org voor de verbindende server en koekelweef.com voor de ontvangende server. Dan is er een server die antwoordt wanneer er naar ruwave.org verbonden worden, vanaf nu de ruwave.org controleserver genoemd, en net zo voor koekelweef.com. Om te beginnen initialiseert de verbindende server een verbinding naar de ontvangende server, en stuurt daarbij zijn domein en gehashde sleutel naar de ontvangende server. Deze verbindt vervolgens naar de ruwave.org controleserver en stuurt de gehashde sleutel daarheen voor de controlestap. De controleserver kijkt of de sleutel behoort tot het domein van ruwave.org en stuurt het resultaat terug naar de ontvangende server, welke dat resultaat doorstuurt naar de verbindende server. In het geval dat het resultaat positief is, worden deze stappen herhaald met de rollen van de ontvangende en de verbindende server omgedraaid, en is het de koekelwave.com controleserver die het resultaat bepaalt. Indien het bevestigen van de sleutel in beide keren successvol is verlopen is Dialback gelukt, en weten de ontvangende server en de verbindende server dat de ander echt van het domein komt waarvan ze beweren vandaan te komen.

De authenticatiestap is voornamelijk van toepassing op clients die bij servers verbinden, of bij servers die meer zekerheid nodig hebben dan slechts de bevestiging van Dialback dat ze met een server op het juiste domein praten. Deze stap bestaat uit het uitwisselen van ondersteunde authenticatiemethoden, waarop een overeenkomstige methode gekozen wordt door de partij die begon met de stream. Daarna vindt authenticatie plaats op de specifieke manier van de gekozen methode, wat alles kan zijn tussen een plain-text wachtwoord sturen tot interactie met externe partijen.

Na deze stappen doorgelopen te hebben, is de stream klaar voor het uitwisselen van stanzas. Stanzas zijn gedefinieerde XML elementen die binnen de stream mogen staan. Een belangrijk detail is dat server naar server verbindingen uit twee losse streams bestaan, een voor het versturen van stanzas, de ander voor het ontvangen van stanzas. Een client en een server praten met elkaar over een enkele stream.

2.2 PubSub

PubSub [6] is kort voor Publish-Subscribe, en is een uitbreiding op XMPP met het doel een algemeen framework te zijn voor het maken van publicaties op een daarvoor bestemde server en de daarbij horende notificaties naar de geabonneerde partijen. Een stanza behorende aan PubSub is te herkennen aan een pubsub element dat direct onder een iq stanza hangt, of een event element dat direct onder een message stanza hangt. De reden dat hier kort PubSub vermeldt wordt is dat het Federation Protocol van Google gebruiktmaakt van PubSub om Federationspecifieke elementen te versturen en te ontvangen. [7]

2.3 Federation Protocol

Het Federation Protocol [7] maakt gebruik van XMPP, PubSub en Operational Transformation om Waves en delen van Waves uit te wisselen tussen partijen, zodat alle deelnemers aan een Wave de Wave kunnen aanpassen, en dat de veranderingen bijna realtime zichtbaar zijn voor elke deelnemer. De specificatie noteert twee deelcomponenten van een Wave-server: Federation Host en Federation Remote.

Het Federation Host component is verantwoordelijk voor:

- het versturen van wavelet operaties gemaakt op de lokale Wave aan andere Wave servers met deelnemers van dezelfde Wave
- het reageren op verzoeken van andere Wave servers voor wavelet operaties
- Het verwerken van nieuwe wavelet operaties, mogelijk van buitenaf

Het Federation Remote component is verantwoordelijk voor:

- het ontvangen van nieuwe wavelet operaties
- het verzoeken van oude wavelet operaties van andere Wave servers

- het versturen van nieuwe wavelet operaties aan het Federation Host deel van andere servers

De belangrijkste elementen van het Federation Protocol:

- wavelet-update wordt gebruikt binnen een PubSub event message stanza om een of meer applied-delta elementen te versturen naar Wave servers met deelnemers van een Wave
- history-request is bedoelt om een deel of de gehele geschiedenis van een Wave op te vragen bij een Wave server, en staat binnen een PubSub request iq stanza.
- submit-request staat binnen een PubSub set stanza, en is het publiceren van een nieuwe delta aan een een Federation Host.
- submit-response is de reactie op een submit request, en geeft aan wat de Wave server met de ontvangen delta heeft gedaan

2.4 Delta's

Updates worden in de vorm van operaties gebundeld in delta's gestuurd, aldus Bekman [8]. Uit de specificatie van het federationprotocol [7] blijkt dat deze base64-gecodeerd worden verstuurd. Base64 wordt beschreven in RFC 1421 [9]. Met base64 wordt ruwe data gecodeerd naar leesbare tekens. Deze ruwe data moet ook gedecodeerd worden naar daadwerkelijke delta's die de server begrijpt. De documentatie vertelt echter niet hoe delta's in deze ruwe data worden opgeslagen.

3 Methoden

Omdat documentatie redelijk vaak ontbrak of incorrect was, hebben wij snel geleerd om functionaliteit eerst zo simpel mogelijk werkend te krijgen, daarna uit te breiden, vervolgens een korte samenvatting van de bevindingen aan elkaar te vertellen, en dat dan vervolgens daadwerkelijk uit te programmeren op een goede manier.

Omdat wij dit vooral mondeling hebben gedaan, is niet alles meer even makkelijk terug te zoeken. We willen in ieder geval de rode draad en enkele voorbeelden geven in dit gedeelte.

In het begin van het experiment hebben wij een complex stuk code, dat allerlei taken uitvoert die nodig zijn voor het experiment. Vaak moeten wij dus deze code aanpassen om verdere experimenten te doen. Zo kunnen wij in de code zetten dat een bepaald blok tekst wordt verstuurd, zodra er iets gebeurt. Bij bijvoorbeeld het ophalen van Wave-geschiedenis, sturen we een bepaald XMPP-stanza zodra een Wave-update wordt ontvangen. Dit stanza betekende alleen iets in onze specifieke situatie, voor één specifieke Wave, waarmee we op dat

moment aan het testen waren. Dit stanza passen we in elk experiment steeds aan totdat we de respons krijgen die we hadden verwacht.

Vervolgens controleren we alles tegen de specificatie tot op dat punt, als dat aanwezig was. Hier bleek vaak dat de documentatie incompleet of onduidelijk was, of dat sommige servers erg vrij omgaan met de regels. Uit de ervaringen tot dan toe opgedaan, en met de eventuele specificatie bij de hand, gaan we het verstuurde blok tekst veralgemeniseren om te werken met alle, of in ieder geval meer, situaties. Dit nog steeds op dezelfde manier, in een blok code waar het uiteindelijk niet thuishoort. Zo hebben wij bijvoorbeeld geprobeerd het History Request-stanza aan te passen om de volledige geschiedenis voor een arbitraire Wave op te kunnen vragen. Hierbij liepen we weer andere problemen tegen het lijf, gegevens die wij niet hadden, enzovoorts. Uiteindelijk hebben we die problemen steeds opgelost door de specificatie na te slaan, te experimenteren en bestaande open-source code te bekijken.

Wanneer we uiteindelijk op het punt zijn gekomen dat we het stanza zo kunnen versturen dat het begrepen wordt door zoveel mogelijk servers en werkt in zoveel mogelijk situaties, gaan we nadenken over de daadwerkelijke implementatie. Dit kan vaak heel snel: we maken één of meer methoden in de juiste klassen, roepen die methoden op de juiste plekken aan, en het is klaar.

Een ander voorbeeld hiervan naast Wave History is Dialback. Dialback was in de RFC die we aanvankelijk gebruikten, RFC3920 [3], erg vaag uitgelegd. Daarin stond bovendien dat Dialback niet per se nodig was, echter als wij het niet gebruikten, kregen we vage "geen toestemming-fouten. Pas na veel proberen kwamen wij erachter dat Dialback in RFC3920bis [5] en XEP0200 [10] en op andere plekken beter werd uitgelegd, waarna wij het snel konden implementeren.

3.1 Delta's

Er moet onderzocht worden hoe waveservers delta's coderen, zodat de ruwaveserver deze kan decoderen en toepassen. Hiervoor is onderzocht hoe een specifieke waveserver-implementatie dat doet, namelijk FedOne. Deze implementatie is de goede om te bekijken aangezien dit de door Google zelf geleverde implementatie is, zie Peterson [11].

Deze server is open source, dus de broncode kan bekeken worden om te ontdekken hoe het coderen van delta's gebeurt.

4 Resultaten

Zie voor resultaten de bijgevoegde code.

4.1 Delta's

Uit het bekijken van de FedOne-broncode is gebleken dat er de codering werkt met basistypen en messages die bestaan uit andere messages en basistypen. Messages en basistypen beginnen met een tag bestaande uit een getal dat aangeeft

wat voor soort data volgt en een cijfer dat aangeeft welk veld van een message het is. Afhankelijk van het soort data kan er nog worden opgegeven hoeveel bytes het basistype of de message inneemt. Hierna volgt de daadwerkelijke data van de message of het basistype.

De complete uitwerking staat in bijlage 1.

5 Discussie

Onze resultaten staan veelal verwerkt in de implementaties. Wij hebben in de loop van de maanden veel problemen opgelost, en alle problemen behandelen zou deze publicatie tientallen pagina's lang maken. Vandaar dat we enkele grote problemen die we getackled of uitgelegd hebben, hier behandelen.

5.1 Algeheel project

Terwijl we bezig waren met informatie lezen over XMPP tijdens het project, bedachten we ons dat de doelstelling van ons project misschien scheefligt. We bedachten ons dat een "Google Wave"-server an sich niet echt bestaat. Een domein heeft een XMPP-server, en die heeft een Google Wave-component.

Wij zijn toch doorgegaan met ons project, maar hebben ons daarbij goed gehouden aan de scheiding van een XMPP-server en een Google Wave-server. Door middel van de code die wij nu hebben, zou het vrij gemakkelijk moeten zijn om een Google Wave-component aan een bestaande XMPP-server toe te voegen. Voor vervolgprojecten is het belangrijk om aan te houden dat de Wave-code eigenlijk maar een component van de hele XMPP-server wordt.

5.2 Dubbele serverstreams

Zoals in de theorie uitgelegd, zijn er voor servers twee TCP-verbindingen nodig. Dit wisten wij pas later.

6 Conclusie

Een Google Wave-server bestaat niet echt. Dit is in feite in XMPP-server die Google Wave aankan. Hiervoor is meer nodig dan in de Google Wave documentatie staat, aangezien die incompleet is. Door onderzoek is het gelukt een werkende server te bouwen. Deze server kan communiceren met andere servers en Waves uitwisselen. Het resultaat is te zien in de broncode.

Referenties

[1] J. Klensin. rfc5321 - simple mail transfer protocol. http://www.ietf.org/rfc/rfc5321.txt, October 2008.

[2] Free Software Foundation. Gnu general public license. http://www.gnu.org/licenses/gpl-2.0.html, June 1991.

[3] Ed. P. Saint-Andre. Rfc 1421 - extensible messaging and presence protocol (xmpp): Core. http://www.ietf.org/rfc/rfc3920.txt, October 2004.

[4] Ed. P. Saint-Andre. Rfc 3921 - extensible messaging and presence protocol (xmpp): Instant messaging and presence. http://www.ietf.org/rfc/rfc3921.txt, October 2004.

[5] Ed. P. Saint-Andre. Rfc 1421bis - extensible messaging and presence protocol (xmpp): Core. http://tools.ietf.org/html/draft-saintandre-rfc3920bis-09, March 2009.

[6] Ralph Meijer Peter Millard, Peter Saint-Andre. Xep-0060: Publish-subscribe. http://xmpp.org/extensions/xep-0060.html, October 2009.

[7] D. Berlin J. Gregorio S. Lassen S. Thorogood A. Baxter, J. Bekmann. Google wave federation protocol over xmpp. http://wave-protocol.googlecode.com/hg/spec/federation/wavespec.html, July 2009.

[8] Jochen Bekmann. Google wave federation. http://www.waveprotocol.org/presentations/wave_federation_pub.pdf, August 2009.

[9] J. Linn. Rfc 1421 - privacy enhancement for internet electronic mail. http://www.ietf.org/rfc/rfc1421.txt, February 1992.

[10] Ian Paterson. Xep-0200: Stanza encryption. http://xmpp.org/extensions/xep-0200.html, May 2007.

[11] Dan Peterson. Google wave federation day. http://docs.google.com/present/view?id=dggjrx3s_112gfkbmsg3, July 2009.

7 Bijlage 1 - Deltacodering

7.1 Standaardtypes

Dit zijn de standaarttypes die voor het coderen van delta's gebruikt worden.

7.1.1 Wiretypes

- WIRETYPE_VARINT = 0
- WIRETYPE_LENGTH_DELIMITED = 2

7.1.2 tag

arguments: int fieldnumber, int wiretype

- varint32 ((fieldnumber << 3) | wiretype)

7.1.3 message

arguments: int fieldnumber, object

- tag(fieldnumber, WIRETYPE_LENGTH_DELIMITED)
- varint32 size
- object
- unknownfields(object)

7.1.4 string

arguments: int fieldnumber, string value

- tag(fieldnumber, WIRETYPE_LENGTH_DELIMITED)
- varint32 size
- rawbyte[] utf8(value)

7.1.5 int32

arguments: int fieldnumber, int value

- tag(fieldnumber, WIRETYPE_VARINT)
- value >= 0?
  - varint32 value
- else
  - varint64 value

7.1.6 int64

arguments: int fieldnumber, long value

- tag(fieldnumber, WIRETYPE_VARINT)
- varint64 value

7.1.7 bool

arguments: int fieldnumber, bool value

- tag(fieldnumber, WIRETYPE_VARINT)
- rawbyte value ? 1 : 0;

7.1.8 bytes

arguments: int fieldnumber, byte[] value

- tag(fieldnumber, WIRETYPE_LENGTH_DELIMITED)
- varint32 size
- rawbyte[] value

7.1.9 enum

arguments: int fieldnumber, int value

- tag(fieldnumber, WIRETYPE_VARINT)
- varint value

7.1.10 varint32/64

arguments: int/long value

- 1bit moreDataFollowing
- 7bit data

  ...
- 1bit moreDataFollowing = 0
- 7bit data

7.2 Messages

Dit zijn de door FedOne geïmplementeerde messages

7.2.1 appliedDelta

1. message signedOriginalDelta
2. message hashedVersion
3. int32 operationsApplied
4. int64 timestamp

7.2.2 signedDelta

1. bytes delta
2. message[] protocolSignature

7.2.3 protocolSignature

1. bytes signatureBytes
2. bytes signerId
3. enum signatureAlgorithm *//alleen SHA1_RSA = 1*

7.2.4 delta

1. message hashedVersion
2. string author
3. message[] operation
4. string[] address

7.2.5 hashedVersion

1. int64 version
2. bytes historyHash

7.2.6 operation

1. string addParticipant
2. string removeParticipant
3. message mutateDocument
4. bool noOp

7.2.7 mutateDocument

1. string documentId
2. message documentOperation

7.2.8 documentOperation

1. message[] component

7.2.9 component

1. message annotationBoundary
2. string characters
3. message elementStart;
4. bool elementEnd
5. int32 retainItemCount
6. string deleteCharacters
7. message elementStart deleteElementStart
8. bool deleteElementEnd
9. message replaceAttributes
10. message updateAttributes

7.2.10 annotationBoundary

1. bool empty
2. string[] element
3. message[] keyValueUpdate changeList

7.2.11 elementStart

1. string type
2. message[] keyvaluepair attributes

7.2.12 replaceAttributes

1. bool empty
2. message[] keyvaluepair oldAttributes
3. message[] keyvaluepair newAttributes

7.2.13 updateAttributes

1. bool empty
2. message[] keyValueUpdate attributeUpdates

7.2.14 keyValueUpdate

1. string key
2. string oldValue
3. string newValue

7.2.15 keyValuePair

1. string key
2. string value

8 Bijlage 2 - Stanza's serveronderzoek

Hier zijn delen van serverlogs ten tijde van de ontwikkeling van ruwaved, waar "PROT IN" staat voor een ontvangen stanza, en "PROT OUT" voor een verstuurd stanza.

```
OPZETTEN VAN EEN VERBINDING NAAR EEN ANDERE SERVER
17:08:48.278 PROT OUT >>
<stream:stream xmlns="jabber:server" xmlns:stream="http://etherx.jabber.org/streams" xml
<<

17:08:48.322 PROT IN >>
<stream:stream xmlns:db="jabber:server:dialback" xmlns:stream="http://etherx.jabber.org/
<<

17:08:48.360 PROT IN >>
<stream:features>
<starttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"></starttls>
<dialback xmlns="urn:xmpp:features:dialback"/>
<mechanisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl"></mechanisms>
</stream:features>
<<

17:08:48.363 PROT OUT >>
<db:result from='dazjorz.com' to='ruwave.org'>foobar</db:result>
<<

17:08:48.556 PROT IN >>
<db:result from="ruwave.org" to="dazjorz.com" type="valid"/>
<<


DIALBACK stanzas
16:27:59.224 PROT IN >>
<db:result to="dazjorz.com" from="gmail.com">CAESBxC8nu2KnSUaEN+h2T3YKXnyvaZz8nV9Re4=</d
<<

16:27:59.226 PROT OUT >>
<db:result from="dazjorz.com" to="gmail.com" type="valid"/>
<<


PRESENCE stanza
07:35:16.325 PROT IN >>
<presence from="<snip>@gmail.com/46BA5C9C" type="unavailable" to="dazjorz@dazjorz.com"/>
<<
```

```
WAVELET-UPDATE binnen PUBSUB stanza
17:40:25.203 PROT IN >>
<message type="normal" from="wave.ruwave.org" to="wave.dazjorz.com" id="ERIv/SWT9kMAAAAA
<request xmlns="urn:xmpp:receipts"/>
<event xmlns="http://jabber.org/protocol/pubsub#event">
<items>
<item>
<wavelet-update xmlns="http://waveprotocol.org/protocol/0.2/waveserver" wavelet-name="wa
<applied-delta>
<![CDATA[CvECCkUKGAgCEhQieqoM1doyKpcnL/xQtrKwRAL+xBISZGF6am9yekBydXdhdmUub3JnGhUKE2Rhemp
</applied-delta>
</wavelet-update>
</item>
</items>
</event>
</message>
<<


COMMIT-NOTICE binnen PUBSUB stanza
17:40:25.204 DEBUG starting to parse message stanza

17:40:25.278 PROT IN >>
<message type="normal" from="wave.ruwave.org" to="wave.dazjorz.com" id="+SOCiaAznvYAAAAA
<request xmlns="urn:xmpp:receipts"/>
<event xmlns="http://jabber.org/protocol/pubsub#event">
<items>
<item>
<wavelet-update xmlns="http://waveprotocol.org/protocol/0.2/waveserver" wavelet-name="wa
</wavelet-update>
</item>
</items>
</event>
</message>
<<


HISTORY REQUEST binnen PUBSUB REQUEST
17:08:48.558 PROT OUT >>
<iq id="qxmpp1" to="wave.ruwave.org" from="wave.dazjorz.com" type="get">
<pubsub xmlns="http://jabber.org/protocol/pubsub">
<items node="wavelet">
<delta-history end-version="5" end-version-hash="PcgedNDUL7E6GuQZeMmmc2yhM60=" start-vers
</delta-history>
```

```
</items>
</pubsub>
</iq>
<<


RESULTAAT VAN HISTORY REQUEST
17:08:48.684 PROT IN >>
<iq type="result" id="qxmpp1" from="wave.ruwave.org" to="wave.dazjorz.com">
<pubsub xmlns="http://jabber.org/protocol/pubsub">
<items>
<item>
<applied-delta mlns="http://waveprotocol.org/protocol/0.2/waveserver">
<![CDATA[Cq0DCoABCi4IABIqd2F2ZTovL3J1d2F2ZS5vcmcvdysxcFN1MDdWaXUtaXcvY29udityb290EhBzam9
</applied-delta>
</item>
<item>
<applied-delta mlns="http://waveprotocol.org/protocol/0.2/waveserver">
<![CDATA[Cu8CCkMKGAgCEhSGzEJwlax0jx/zW1eY+ondXDz8QRIQc2pvcnNAcnV3YXZlLm9yZxoVChNkYXpqb3J
<<

17:08:48.685 DEBUG starting to parse iq stanza
```

```
17:08:48.689 PROT IN >>
epwSQVOQwVDpek0es86x81lDQYARIYCAISFIbMQnCVrHSPH/NbV5j6id1cPPxBGAEg/NXu85gl]]>
</applied-delta>
</item>
<item>
<applied-delta mlns="http://waveprotocol.org/protocol/0.2/waveserver">
<![CDATA[CvACCkQKGAgDEhSAy6HpMvOTf66FCeVywZC2FKv5oxIQc2pvcnNAcnV3YXZlLm9yZxoWChRkYXpqb3J
</applied-delta>
</item>
<item>
<applied-delta mlns="http://waveprotocol.org/protocol/0.2/waveserver">
<![CDATA[CvACCkQKGAgEEhT++65piQ7iKgOQirLPYFTj5n7amRIQc2pvcnNAcnV3YXZlLm9yZxoWChRkYXpqb3J
</applied-delta>
</item>
<item>
<commit-notice xmlns="http://waveprotocol.org/protocol/0.2/waveserver" version="5"/>
</item>
<item>
<history-truncated xmlns="http://waveprotocol.org/protocol/0.2/waveserver" version="5"/>
</item>
</items>
</pubsub>
</iq>
<<
```