# 如何做 Android 应用流量测试

## 前言

我们经常手机应用有这样的困惑:想知道应用费不费流量;想知道某几款同类应用,做同样的事儿,哪个更省流量;更深入的,想知道一款应用为什么这么费流量,流量都消耗在哪了;想知道在大 4G 时代,一觉醒来怎么房子车子就变成别人的了。。本文将介绍给您,解答上述困惑的简单方法。

## 工具

**GT**(中文产品名称: 随身调): 是腾讯出品的开源调试工具,本次测试中用其进行手机的流量统计和抓包。请在 Android 手机上安装 GT 应用(可以通过官网或应用宝下载)。

Wireshark: 抓包的分析工具,也提供了 Android 手机的抓包实现,GT 中抓包的功能 就是在其提供的实现基础上的易用性封装,本次测试中用 Wireshark 进行抓包的分析。请在 PC 上安装 Wireshark。

## 正文

其实想知道一款应用费不费流量,大部分 Android4.x 版本系统已经可以简单的查看了:



关注流量比较粗的话,看一下上面这里也就够了,但从测试的需求看,这里只能观察到 宏观的流量情况,到1天的流量消耗就没法再细化了,如果想知道具体一个业务操作或一段 时间内的流量消耗呢?如果想知道应用一次启动的流量消耗呢?这时就该使用前面介绍的 工具了。

比前面稍微深入一些,我们可能需要知道一个业务操作过程内,消耗的流量,及发出 请求的流量、收到响应结果的流量各有多少,并且流量的消耗曲线是怎样一个走势。这时 就该使用 GT,关于 GT 的基本使用和为什么用 GT, GT 网站有详细的说明,这里只介绍和 流量相关的部分。GT 提供了一种简单的测试方式,也提供了一个严谨但麻烦的测试方式。

### ● 首先我们来看简单的方式:

1. 先将应用运行起来,然后启动 GT 并在 GT 上选中被测应用及被测项 NET (流量)。

👒 🔀 🖬 🏈 🛜		🖬 🕯 6:27		📽 😭 🛛 🗳 🗳		র 🖬 🖬 6:51
	Main				Main	
App Under Test :	Reference of the second s	- 文档 入	1. 选中被测应 用	App Under Test :	北京大学 navsns	esh running
MemoryInfo PSS	Mem Assist	ant: off		MemoryInfo PSS	Mem Assist	ant: off
OtherInfo Cpu Net		EIZA	2. 选中被测项 NET	OtherInfo Cpu	Jiffes MARX III	
e乐充公交卡(- {,,,}				立器稿商业(15 = 下二新新家田 - ○ - 」 - 「東庄路		
Main Para.	Profiler Lo	g Plugin		Main Para.	Profiler Lo	g Plugin

2. 业务操作前,启动数据采集,将会记录选中应用的流量的变化,为了方便统计,可以先 把业务操作前发生的流量记录归零。

😪 😭 🔇	> Ⅲ		👒 🖬 🛜 🗳 🖩	😪 🖬 🖬 7:20
	in <b>out</b> edit		in out	edit
Outpara	Gather & Waming (G&W)	启动数据采集	Outparam items	& Warning (G&W)
Show of the second seco	on Air-console(<=3 emply 可归零 al Parameters		Show on Air-console(<=3) empty Optional Parameters	= 虚凯原楼 :学
PNET wifi:t17.01	Pnet:com.t.ent.navsns KBJr69.36KB 3G:10KBJr0KB 2G:10K.		PNET Pnet:com.t.ent.navsns wifi:t0KBjr0KB 3G:t0KBjr0KB 2G:t0KBjr0KB	
CPU 18.09%	СРИ		CPU CPU 15.94%	
Disabl	ed Parameters # A 大阪		Disabled Parameters 副科大师	
MEM	МЕМ		МЕМ МЕМ	
NET	NET		NET NET	
SIG	Signal		SIG Signal	
FPS	FPS		FPS FPS	
Main	Para. Profiler Log Plugin		Main <b>Para</b> . Profiler L	og Plugin

3. 退到应用界面,执行需测试的业务操作。

4. 业务操作后,回到 GT 界面,停止流量数据的采集,查看本次业务操作流量的变化。



到这里,从前面一张图我们已经可以知道一个业务操作过程中消耗的流量,包括发出请求的流量、收到响应结果的流量、流量消耗曲线是怎样一个走势了。

5. 我们可以保存本次测试结果到文件,以备后面更深入的分析。



### 参观一下这个文件:

	A	В	С	D		
1	key	Pnet:com.	tencent.r	avsns		
2	alias	PNET				
3	unit	(KB)				
4	begin dat	#######				
5	end date	#######				
6	count	15				
7						
8		transmitt	received			
9	min	2	3			
10	max	5	10			
11	avg	3	4			
12						
13	28:41.3	0	0			
14	28:42.3	2	0			
15	28:56.6	2	3			
16	29:11.1	3	4			
17	29:12.1	4	4			
18	29:13.1	4	4			
19	29:14.1	4	5			
20	29:23.4	4	5			
21	29:24.4	4	5			
22	29:27.6	4	6			
23	29:31.6	5	6			
24	29:42.9	5	6			
25	29:43.9	5	6			
26	31:56.3	5	6			
27	31:57.3	5	10			
28						

用强大的 Excel 把 GT 应用里显示的趋势图还原出来不是难事。



### ● 我们再来看看麻烦而严谨的方式:

如果只是纯粹测测流量,上面的方式也足够了,那我们为什么需要麻烦而严谨的方式 呢?这里有两个原因,一个是**仅仅知道流量的大小和趋势,还不足以对后续的流量优化进** 行明确的指导,即知道流量可能有点多,但不知道该如何着手优化。另一个是原因是弥补 上面方式的一个不足:有的应用,使用了本地 socket 和手机里其他进程产生交互,有时候 Android 系统会把这种手机内部的 socket 传输的数据量也计算到应用消耗的流量里(比如常见的视频应用不少都有这个问题),此时上面的方式就显得不够准确了,要获得真是网卡上发生的流量,就需要抓包这种终极方法了。注意掌握这种方法的前提是您得先掌握基础的TCP和HTTP网络知识。

手机抓包是针对手机的网卡,所以这种方式无法单独抓一个应用的包,需要后续将归属于应用的包分析出来,而为了后续分析减少工作量,测试时候应尽量把其他能消耗流量的应用都关了。Android 手机的抓包是 Wireshark 提供的实现,GT 上面做了封装,使手机可以不必连着 PC 即可抓包,方便在室外测试的场景。

1. 先从 GT 启动抓包。



抓包功能的入口在这里

🎸 🛜 👒 III	🖉 🗐 🗐		🗳 🏈 😭 🖌	<b>"// 🖬 11:21</b>
Back GTPcap			Back	GTPcap
Foldername			Foldername	
Capture 保存的目录			Capture	
Argument		开始抓包	Argument	Off
Start			>	Stop
GTMemFill Fill Memory GTSdcardFill SdCard fill			tcpdump: listening Got 33 Got 71 Got 126 Got 130 Got 131 Got 150 Got 199 Got 222 Got 243	g on rmnet0, link-type 动消耗流量的业务操 后, 抓包的个数会不 刷新
Description <sup>.</sup>			Description:	
这是一个抓包插件,直接点击Start即可 击Stop后保存已抓包的内容到文件。抓 用PC上的Wireshark软件分析。	抓包,点 包文件可 Plugin		这是一个抓包插件 击Stop后保存已抓 用PC上的Wireshar	,直接点击Start即可抓包,点 包的内容到文件。抓包文件可 k软件分析。

- 2. 之后还是执行测试的业务操作。
- 3. 被测业务操作结束后,点击 stop,即停止抓包,并把抓包文件保存在对应的目录中。

$\triangleright$	DCIM	▲ 名称	▲ 大小	类型
	🔰 Download	Capture 1, pcap	432.00K	Wireshark ca
⊿	🕌 GT			
	D 📙 GW	抓包保存的文件		
	🖻 퉲 Log			
	📗 Profiler			
	Icpdump			
	📗 Capture			
	TLog			
$\triangleright$	🗼 iFlyIME			
	📗 LazyList			
	legacy			
	🚺 mfcache			

将抓到包文件导入到 PC,用 Wireshark 即可分析抓包文件。关于 Wireshark 的使用,和 PC 上的使用没有区别,请大家自行在网上搜索,这里仅对使用 Wireshark 的要点提示下:

1 我们最先需要知道我们的应用发出了哪些请求,对应了上行流量,可以在 Wireshark 左 上角【过滤】框输入"http"或"tcp"(如果确认过被测应用都是 http 请求,就只需要按 http 过滤),确认测试场景 GET 和 POST 的请求类型和个数(过滤结果可按【Info】分类更 方便统计)。

📕 Ca	pture1.pcap	Wireshark	: 1.10.5 (SVN	NRev 5426	2 from /trur	nk-1.10)]								
File	<u>E</u> dit <u>V</u> iew	<u>G</u> o <u>C</u> aptu	re <u>A</u> nalyze	Statistics	Telephony	<u>T</u> ools	Interna	als <u>H</u> elp	0					
0	•	<i>i</i> E	🗅 🗙 🔁	୍ ୍ 🍦	🔹 🏟 🔻	F 🕹		]  €	Q (	Q 🖭   i	¥ 🗹 🖲	<b>-</b> %	Ø	
Filter	http	过滤	<b>框,先按</b>	http 🗉	或 tcp 过	北滤	▼ Exp	ression	Clea	Apply	Save	按谐	ė I	
No.	Time		Source	Destinatio	on .	Protoc	ol	Length	Info	一 出 L	11110,	194月7	<u>ت</u>	
22	68 12:39	:23.49326	0 10.25	. 140. 206	5.160.215	HTTP		277	' Cont	· 类型	分组。	-HTTP t	raff	ic
23	59 12:43	:02.35325	5 10.25	. 101. 227	.160.23	HTTP		374	Cont	inuatior	or non	-HTTP t	raff	ic
23	60 12:43	:04.61078	8 42.12	010.25.1	.198	HTTP		74	Cont	inuation	n or non	-HTTP t	raff	ic
6	09 11:27	:52.68257	8 10.25	. 42.120.	176.10	HTTP		291	GET ,	/activei	p/?agoo	_apn=3G	INET&	app_
1	97 11:21	:43.46492	1 10.25	. 42.120.	176.10	HTTP		291	GET ,	/activei	ip/?agoo	_apn=3G	INET&	app_
4	22 11:22	:37.38544	2 10.25	. 42.120.	176.10	HTTP		291	GET ,	/activei	ip/?agoo	_apn=3G	INET&	app_
18	96 12:25	:40.12491	4 10.25	.124.160	.175.165	HTTP		68	GET ,	/cgi-bir	n/microm	sg-bin/	'newg	etdn
7	13 11:31	:16.08160	2 10.25	. 140.206	5.160.191	HTTP		1,338	GET ,	/cgi-bir	n/mmsupp	ort-bin	ı∕qrc	ode1
18	26 12:25	:23.72534	8 10.25	. 58.68.2				339	GET ,	/gslb/gs	lb/getb	ucket.a	ısp?v	er=3
20	18 12:26	:27.26778	3 10.25	. 118. 🔀	些 卻 定 1	<b>有</b> 水,	大壮	358	GET ,	/gs1b/gs	1b/getb	ucket.a	ısp?v	er=3
23	17 12:40	:44.26813	6 10.25	. 118. 194	-***	<b>☆₩</b>	*	358	GET ,	/gs1b/gs	1b/getb	ucket.a	sp?v	er=3
19	05 12:25	:40.27055	6 10.25	. 125. 🏨	24 关的	「奴,	开田	413	GET ,	/mmsns/C	M4v0FU2	h0u3n00	)UuoF	xMib
19	02 12:25	:40.24232	8 10.25	. 125	$\frac{240.68}{4}$	福礼	是由	423	GET	/mmsns/F	piajxSqB	RaEL2rB	gDqO	etQ3
19	57 12:25	:50.94/89	4 10.25	. 125 00	urce p	CHURC	ΖШ	423	GET ,	/mmsns/F	'iajxSqB	RaEL2rB	gDqO	etQ4
2	16 11:21	:43.90738	2 10.25	· 42.1	机发出的	KITTP		922	GET ,	/rest/ap	013.do?t	t1d=/03	304%	40ta
4	29 11:22	:37.64205	2 10.25	. 42. 120.		HTTP		922	GET ,	/rest/ap	013.d0?t	t1d=/03	304%	40ta
	1/ 11:2/	:53.20615	5 10.25	. 42. 120.	111.20	HITP		922	GEI	/rest/ap	013.00?t	T10=/03	304%	40ta
17	88 11:22	:11.05810	2 10.25	. 54.230.	127.189	HITP		123	GET,	/rtistat	us.dat	HTTP/1.	0	
1 1/	79 12:22	:12.60234	5 10.25	. 54.230.	127.80	HITP		123	GET,	/rtistat	listat	10£.	0	Deba
1	30 11:21	.30.33330	0 10.20	. 01.133. 61 135	191.9	HITP		212	GET	/server_	list?fm lict?fm	1 0 F v - 2	aop=	200a
	/0 11:21 /0 11:21	.42.0104/	0 10.25 7 10.25	61 125	101.0			212	GET ,	/server_ /server	list?fm	=1&fv=2	aop=	200a
2	78 11 . 22	16 58005	1 10.25	1/0 200	160 101	нттр		1386	GET ,	/zh cw/k		ion/ima	aop=	Copp
	20 11.31	.16 44170	1 10.25	140.200	160 101	нттр		1367	GET ,	/zh_CN/k	tmlodit	ion/sty	le/a	cod
18	02 12.25	.10.441/9	7 124 1	610 25 1	108	нттр		735	ULT D	/1 0 200		ion/sty	re/q	cou
10	2 11.21	.35 58482	4 140 2	010 25 1	108	нттр		483		/1 1 200				
	48 11:21	:36.83647	3 140.2	010.25.1	.198	HTTP		558	B HTTP	/1.1 200	) ок			

2 在具体请求上可以右键"follow tcp stream",等同于过滤条件 tcp.stream eq xx,这样可以 过滤出和它在同一个 TCP 流的消息。

	Capture1.pcap	Wireshark 1.	10.5 (SVN Rev 54262	from /trunk	-1.10)]						
<u> </u>	e <u>E</u> dit <u>V</u> iew	<u>G</u> o <u>C</u> apture	<u>Analyze</u> <u>S</u> tatistics	Telephon <u>y</u>	<u>T</u> ools <u>I</u> nt	ternals <u>H</u> elp					
0	•	<u> </u>	💥 🛃 🔍 🖕	🏟 🌍 ዥ	<u> </u>		0,00	<b>**</b>	ı 🗹 🖣	8 %	Ø
Fil	Filter tcp.stream eq 51   Expression Clear Apply Save										
No.	Time		Source Destinatio	n í	Protocol	Lenath	Info				
	608 11:27	:52.679169	10.25.42.120.	176.10	тср	56	57006	> http	[ACK]	Seq=1	Ack=1 Win
	613 11:27	:52.899552	10.25.42.120.	176.10	тср	56	57006	> http	[ACK]	Seq=23	36 Ack=15
	641 11:28	:52.994211	10.25.42.120.	176.10	тср	56	57006	> http	[ACK]	Seq=23	36 Ack=154
	606 11:27	:52.522108	10.25.42.120.	176.10	тср	76	57006	> http	[SYN]	Seq=0	Win=14600
	609 11:27	:52.682578	10.25.42.120.	176.10	НТТР	291	GET /a	ctivei	o/?agoo	o_apn=3	GNET&app_
	612 11:27	:52.899003	42.12010.25.1	.198	нттр	208	HTTP/1	.1 200	ок		
	611 11:27	:52.879302	42.12010.25.1	.198	тср	56	http >	► 57006	[ACK]	Seq=1	Ack=236 \
	640 11:28	:52.958452	42.12010.25.1	.198	тср	56	http >	► 57006	[FIN,	ACK] S	Seq=153 A
	607 11:27	:52.678863	42.12010.25.1	.198	тср	76	http >	► 57006	[SYN,	ACK] 5	5eq=0 Ack⊧
	<b>沙水和</b>		ませた 同一 ん	rcp 法由	的游自	白红木	- )/~ 法-	Hote Afr	CD VK	自 117	בי מידי
1	必雪仰	定与 GEI W	月水江问一个		叫们心	,也怕冲	·八庄:	<b>按</b> 凹 I	UL 拍	図、 11	111月
	求及其	应答 200(	DK,熟悉 TCP <sup>,</sup>	协议的同	学可以	很容易判	间断出	这是-	一个短	连接。	<b>b</b>

3 过滤条件出来后再点击【统计】->【概要】,对应 Bytes 栏【显示】列的数据即为流量。

🚄 Ca	Capture1.pcap [Wireshark 1.10.5 (SVN Rev 54262 from /trunk-1.10)]													
<u>F</u> ile	<u>E</u> dit	<u>V</u> iew	<u>G</u> o	<u>C</u> apture	<u>A</u> nalyze	<u>S</u> t	atistics	Tel	ephon <u>y</u>	<u>T</u> ools	Interna	als <u>H</u> elp		
0	۲		Ø.		🗙 🔁	1	<u>S</u> umm	nary	-				$\Theta$	0 🗗
Filter	: tc	p.strean	n eq	51			Comm Show	addr	Summar ess resol	y ution		ression.	. Cle	ar Aj
No.	509	Time	50	670160	Source	Ra	Proto Conve	col Hi ersatio	ierarchy			Length	Info	16 .
	613	11:27:	52.	899552	10.25	0	Endpo	pints	5115			56	5700	)6 >
	541 506	11:28: 11:27:	52. 52.	994211 522108	10.25		Packe	t Ler	ngths			56	5700 5700	)6 > )6 >
	609 512	11:27	52.	682578 899003	10.25	lest.	IO Gra	apn 				291 208	GET	/act 2/1_1
e	611	11:27:	52.	879302	42.12		Conversation List				- + 	56	http	> 5
(	540 507	11:28: 11:27:	52.	958452 678863	42.12		Servic	e <u>R</u> e	sponse T	ïme	•	56	http	) > 5 ) > 5
							ANCP							

🧲 Wireshark: Summary		_		_	- 0 X
File Name: Length: Format: Encapsulation: Packet size limit: Time First packet: Last packet: Elapsed: Capture Capture file comments Interface Dropped Packets	C:\L 442 Wire Linu 655 201 201 201 01:2 Capture Filter Lir	Jsers\yoyoc 299 bytes eshark/tcpc ix cooked-r 35 bytes 4-04-24 11 4-04-24 12 22:18 ik type	in\Desktop\Capt lump/ pcap node capture :21:35 :43:54	ure1.pcap Packet size lir	nit
unknown unknown <b>Display</b> Display filter:	unknown Lir	nux cooked	d-mode capture p.stream eq 51	65535 bytes	
Ignored packets:		0	(0.000%)		
Traffic Packets Between first and last packet Avg. packets/sec	Captured 4 2378 t 4938.887 sec 0.481	Displayed 9 60.472 sec 0 149	<ul> <li>Displayed 9</li> <li>0.378%</li> </ul>	6 ◀ Marked ◀ 0	Marked % 4 0.000%
Avg. packet size Bytes Avg. bytes/sec Avg. MBit/sec	169.986 bytes 404227 81.846 0.001	103.444 b 931 15.396 0.000	ytes 0.230% 包的总大小叔 出的消息大小	0 ē 404227Byt 、是 931Byte	0.000% te,本次过滤
<u>H</u> elp				ОК	<u>C</u> ancel

4 通过对包的过滤分析,我们自然就可以得到<mark>流量的大小,产生流量的类型和原因,请</mark> 求的频率,这样就能够对后续的流量优化进行指导了。

3G-环境限定1- 环境限定2	流量总消耗 (bytes)	类型	命令字	请求次数	流量消耗 (bytes)	过滤详情	备注
		GET	qt=rtt	7	30095	tcp.stream eq 1 or tcp.stream eq 18 or tcp.stream eq 27 or tcp.stream eq 33 or	3分钟一个
		GET	navtemp	4	18777	tcp.stream eq 0 or tcp.stream eq 51 or tcp.stream eq 47 or	15:27 <sup>~</sup> 15:28重复 三次才得到成功结果
		POST	wup				有连续1s俩的同一消  息
某应用场景1	50394	POST	lbsi?c=1& mars=1			http.request.uri contains "lbsi"	[TCP Retransmission] POST /lbsi?c=1&mars=1 HTTP/1.1 (application/x- www-form- urlencoded)带着的 指数回退重传的,我 们不需要管,和被重 传消息是有关系的。 有重复消息
		POST	analytics /upload	1	1522		数据上报,很小
竞品应用−场景1	0						

## 5 更谨慎的, 抓包和 GT 采集流量数据可以相互对照, 避免分析时有所遗漏。

<u>Eile Edit View Go Capture Analyze Statistics Telephony Tools Internals Help</u>	开始	台 插入	页面布局	公式 数
◎ ◎ 🚄 🔳 🙇   🖻 🛗 💥 😂    🗢 🌳 🗛 👍 👱   🗐 🕞   ① Q ℚ 🕾	e 📂 🕯	- L -	(°   = 💼	PNET_201420
Filter: http contains "GET" or http contains "POST"		A13	▼ 🖳 fx	19:07:57
No. Time Source Destination Protocol Length Info		A	В	С
86 2014-04-30 19:07:56 4 <del>20196 10.1.9112.64.23HTTP 270 GET /?ot</del> =r11&	13	-07:57.	1	4
187 2014-04-30 19:10:56.579664 10.1.9112.64.23HTTP 270 GET /?qt=rtt&	<b>-</b> 14	.08:08.	1	5
195 2014-04-30 19:10:58.763697 10.1.9112.64.23HTTP 270 GET /?qt=rtt&	C= 15	09:56.	2	5
302 2014-04-30 19:13:58.948607 10.1.9112.64.23HTTP 270 GET /?qt=rtt&	16	09:57.	2	5
307 2014-04-30 19:14:01.785267 10.1.9112.64.23HTTP 270 GET /?qt=rtt&	17	10:06	2	5
315 2014-04-30 19:14:01.957738 10.1.9112.64.23HTTP 270 GET 72d-11t&		10.00.	2	<u>ہ</u>
444 2014-04-30 19:23:39.944821 10.1.9512.64.23HTTP 2/0 GEL / (qt=rtts	10	10.11.	2	5
402 2014-04-30 19:23:40.21 059 10.1.9112-04.21HTP 2/0 GET / (0L=1140)	19	10:51.	2	0
752 2014-04-30 19:37:14 285003 10 1 9112 64 23HTTP 276 GET /201=rtt&	20	<u> </u>	2	8
945 2014-04-30 19:40:14.574696 10.1.911 64 23HTTP 270 SET /201=0158			3	9
950 2014-04-30 19:40:14.749963 10 1.9112.64.23HTTP 270 GET / Cotents	22	11:13.	3	9
172 2014-04-30 19:10:13.9601 10.1.959.151.11HTTP 642 POST /diagnos	23	13:59.	3	9
102 2014-04-30 19:08:06.844380 10.1.9112.64.23HTTP 524 POST /index.w	up 24	<u>∼</u> 14:02.	3	12
146 2014-04-30 19:09:56.12043 10.1.9112 64.23HTTP 454 POST /index.w	ur 25-	14:09.	4	13
152 2014-04-30 19:10:05.352287 10.1.9112.64.23HTTP 454 POST /Index.w	ur 26	-14:33.	4	13
218 2014-04-30 19:11:07.67 6659 10.1.9112.64.23HTTR 524 POST /index.w	ur 27	- 15:55.	4	13
328 2014-04-30 19:14:08.340859 10.1.9112 64.25HTTP 524 POST Andex.w	28	15:58	5	14
351 2014-04-30 19:14:31.550090 10.1.9118 60.23HTTP 454 POST / PROCK.W	29	20:36	5	14
488 2014-04-30 19:23:52 17 520 10 1 9112 64 23 HTTP 524 POST / Index w		20.00.	5	14
840 2014-04-30 19:32:32:172030-10,1:3112.04.23HTTP 454 POST / index.w	200	20.00.	5	14
879 2014-04-30 19:37:27.26616 19.1.9112.64.23HTTR 524 POST /index.w	10 00	23:41.	5	19
911 2014-04-30 19:39:26.749926 10 1.9112.64.23HTTR 454 POST /index.w	32	23:47.	5	18
915 2014-04-30 19:39:27.731-30 10.1.9112.64.23HTTP 454 POST (index.w	33	23:53.	5	19
979 2014-04-30 19:40:28.010699 10.1.9112.64.23HTTP 524 POST /index.w	ur 34	25:18.	5	19
1 2014-04-30 19:05:33.408281 10.1.9111.161.4HTTP 519 POST / lbs 72c=	18 35	29:58.	5	19
18 2014-04-30 19:05:36.101594 10.1.9111 101 4HTTP 519 ROST /1bsi?c=	36	34:01.	6	19
591 2014-04-30 19:33:23.794248 10.1.961.135.18HTTP 351 ROST /xcloudb	37	▶34:35.	6	19
6// 2014-04-30 19:33:31.1065// 10.1.961.135.18HTP 1282 POST xcToudb	38	- 34:39.	6	19
210 2014-04-30 19:11:04.799068 10.1.9112.64 23HIP 270 LICP Retransm	- 39	37:15.	6	23
⊞ Frame 172: 642 bytes on wire (5136 bits), 642 bytes captured (5136 bits)	40	37.28	7	24
H Linux cooked capture	41	39.16	7	24
H Internet Protocol Version 4, Src: 10.1.9./4 (10.1.9./4), DST: 59.151.110.2	40	20.10	7	24
H Transmission Control Protocol, Src Port: 40083 (40083), DSt Port: http (80	44	39.10.		24
In the based text data: application/x-www-form-urlencoded	A ST	-39:27.	1 2	24
a cine based ceve data, apprication x-mm-rorm-or reneoded	44	~39:28.	8	24
	45	40:15.	8	24
	46	40:16.	8	27
10020 3b 97 6e fd b4 03 00 50 15 e6 2e 93 c8 20 ed fe 'n P	48	40:28.	8	27
0030 80 18 00 e5 42 5b 00 00 01 01 08 0a 00 44 5b 50B[D[P	48	40:29.	8	28
0040 f6 49 3d 82 50 4f 53 54 20 2f 64 69 61 67 6e 6f .I=.POST /diagno	14 4	► H PN	ET_20140430	194203 / 🕇

#### ● 如何判断一个应用的流量消耗偏高

如果看流量的绝对值看不出高低,那就找几个同类型的产品对比一下。如果完成同样的 事务,被测应用比同类产品高很多,那就是偏高了,可能有优化空间。

#### ● 如何找到有效的优化点

把分析的不同类数据包,按包占总流量大小的比例,和包的数量排序,占比多的,和消息数量多的,一个优化空间大,一个精简请求次数的机会大。

#### ● 常见的流量问题

最后简单例举几类可控的比较容易优化的流量问题给大家:

◆ 冗余内容

同类请求被间隔执行,请求的内容包含一些相对静态的信息,正确的处理是第一次请求 包括静态信息就好,后面的同类请求只包含必要的即时变化信息即可。错误的处理方式是每 次请求服务器都返回一次静态信息。

◆ 冗余请求

有的时候会发现应用短时间内发出多个同样的请求,收到结果也都几乎一样,这种情况 应该尽量减少请求次数,同时注意排查程序逻辑错误,也许问题不像表面看起来那么简单。

◆ 无用请求

有的请求,你会发现谁也不知道它是干嘛的,很可能是以前版本遗留下来的无用请求, 或者是引用的其他代码包偷偷发出的,甚至是间谍请求,请收集一切证据后,毫不犹豫的干 掉它。

◆ 永远无法得到回应的请求

如果见到某类请求永远的连接失败或被返回 404 之类的失败结果,那它不是历史遗留的 多余请求,就是某个不易察觉的功能已经失效了。

◆ 过多的失败请求

有见过一类或一组请求,n个成功之中夹着m个失败的吗?举个简单的场景,某类请求,间隔1分钟后连续发两次,总是先有一次失败的请求,1s后马上再次发出一次同样的请求就成功了(这里1s后发出的请求是指业务逻辑层判断前面请求失败后延迟1s后重传的)。很好奇为什么第一次总失败是吧,就有这么种情况,客户端两次请求乐观的想要复用同一个TCP连接(长连接半长连接),但是服务端不这么想,也许是客户端发起两次请求的间隔,超出了服务端长设置的长连接无响应时限。。如何判断呢?看看失败的那次请求,是否和前一次成功的请求复用了同一个TCP连接(体现在Wireshark的streamId)。

◆ 非预期请求

比如一种常见的情况,应用退后台后,有些请求就没必要了,**观察下自己的产品,是 否在后台真的没有发出这些请求。**