Ein BitTorrent Analyzer für das Bro NIDS

Nadi Sarrar
nadi@cs.tu-berlin.de

Hauptstudiumsprojekt
Betreuer: Bernhard Ager

Technische Universität Berlin
Deutsche Telekom Laboratories, INET
Research Group Prof. Anja Feldmann, Ph.D.

1. Februar 2008

Zusammenfassung

BitTorrent ist ein populäres Peer-to-Peer-Protokoll zum Austausch von Dateien. Es ermöglicht eine breite Verteilung von großen Datenmengen, wobei weder eine hohe Bandbreite noch kostenintensives Serverhosting notwendig sind. Diese Arbeit erklärt die Funktionsweise von BitTorrent im Detail. BitTorrent besteht aus zwei Protokollen, einem für Peer-to-Peer-Verbindungen und einem zur Kommunikation zwischen Peer und Tracker. Für beide Protokolle haben wir Analyzer für das Bro NIDS entwickelt, mit welchem wir zwei Mitschnitte des Münchner Hochschulnetzes, einem von 2005 und einem von 2007, analysiert haben. Die Ergebnisse dieser Analysen zeigen Ähnlichkeiten sowie Unterschiede.

Inhaltsverzeichnis

1 Einleitung

BitTorrent wurde im Jahr 2001 von Bram Cohen entworfen und in einer ersten Implementierung veröffentlicht. Seitdem steigt die Popularität von BitTorrent stark an. Mittlerweile sind diverse freie und kommerzielle Implementierungen des Protokolls verfügbar. Einer Analyse des Unternehmens Ipoque zufolge sind etwa 30% des gesamten Internetverkehrs Peer-to-Peer-Daten, in der Nacht sogar 70%. BitTorrent hat mit etwa 53% den größten Anteil daran [1]. Da das Interesse an Statistiken über BitTorrent-Verkehr groß ist, entwickeln wir im Rahmen dieser Arbeit Software, die BitTorrent-Verkehr erfasst und analysiert. Dafür erweitern wir das Bro NIDS um entsprechende Analyzer. Wir analysieren Daten des Münchner Hochschulnetzes und präsentieren die Ergebnisse.

In Abschnitt 2 beschreiben wir BitTorrent im Detail. Eine Übersicht über die Funktionsweise und die besonderen Features von Bro geben wir in Abschnitt 3, anschliessend beschreiben wir die Implementierung unseres BitTorrent-Analyzers. Die Ergenisse unserer Analysen stellen wir in Abschnitt 5 dar. Begriffe, die kursiv geschrieben sind, werden im Glossar am Ende dieses Textes erläutert.

2 BitTorrent

BitTorrent besteht aus vier Komponenten. *Peers* sind die Endbenutzer, sie laden und verteilen Dateien. *Tracker* machen Peers miteinander bekannt, die am selben *Torrent* interessiert sind. In der so genannten Metainfodatei (Abschnitt 2.1) sind alle notwendigen Information über ein Torrent enthalten, die ein Peer benötigt, um an dessen Verteilung teilzunehmen. Über eine Metainfodatei wird ein Torrent identifiziert. Spezielle Suchmaschinen haben sich zur Aufgabe gemacht, Metainfodateien zu indizieren und verfügbar zu machen.

Zunächst soll anhand eines Beispiels der typische Ablauf eines Dateidownloads per BitTorrent veranschaulicht werden:

1. Eine Metainfodatei wird bezogen, üblicherweise von einer Suchmaschine per HTTP.
2. Durch eine Clientsoftware wird der in der Metainfodatei referenzierte Tracker kontaktiert, und eine Liste von Peers wird angefordert.
3. Verbindungen zu einigen Peers werden aufgebaut, Verbindungsanfragen von anderen Peers werden angenommen. Es kommt zum Austausch von *Pieces*.
4. Nach Beendigung des Downloads aller Pieces werden solange andere, interessierte Peers versorgt, bis die Clientsoftware beendet wird.

Bei Verbindungen zwischen einem Peer und einem Tracker kommt das Trackerprotokoll (Abschnitt 2.2) zum Einsatz. Der Austausch von den eigentlichen Daten findet ausschließlich in Peer-to-Peer-Verbindungen statt, in welchen das Peerprotokoll (Abschnitt 2.3) verwendet wird.

Typ	Syntax	Beispiel
String	Dem eigentlichen String wird die Stringlänge und ein Doppelpunkt vorangestellt.	`3:bro` ⇒ bro
Integer	Die Zahl wird mit dem Präfix `i` und dem Postfix `e` umschlossen.	`i23e` ⇒ 23 `i-1e` ⇒ -1
List	Listen beginnen mit dem Zeichen `l` und enden mit `e`, die Werte sind ebenfalls bencodet.	`l3:val4:val2` ⇒ [val,val2]
Dictionary	Verzeichnisse beginnen mit dem Zeichen `d` und enden mit `e`, die Key-Value Paare sind ebenfalls bencodet.	`d3:key5:valuee` ⇒ {key:value}

Tabelle 1: Bencoding

2.1 Metainfodatei

In der Metainfodatei [2] (typische Dateiendung: `.torrent`) steht neben der URL eines Trackers auch der Name der verteilten Datei (es können auch mehrere sein) und Informationen über die Pieces, sowie weitere optionale Metainformationen. Der Inhalt der Metainfodatei ist ein Verzeichnis in *Bencoding* [3] (Tabelle 1) mit mindestens den im folgenden aufgeführten Einträgen, auf optionale Metainformationen gehen wir nicht ein. Der announce-Eintrag enthält eine oder mehrere Verweise auf Tracker. Der info-Eintrag ist wiederum ein Verzeichnis in Bencoding und enthält Informationen über die Daten, die mit diesem Torrent verteilt werden. Die Struktur des Wertes des info-Eintrages unterscheidet sich, wenn anstelle einer einzelnen Datei (Single-File-Mode), mehrere Dateien (Multiple-File-Mode) mit nur einem Torrent verteilt werden.

`info`-Verzeichnisinhalt im Single-File-Mode:

Schlüssel	Typ	Wert
`piece length`	*Integer*	Größe der Pieces in Bytes
`pieces`	*String*	Konkatenierung von 20-Byte SHA1 Hashwerten, ein Wert pro Piece
`name`	*String*	Name der Datei
`length`	*Integer*	Größe der Datei in Bytes

`info`-Verzeichnisinhalt im Multiple-File-Mode:

Schlüssel	Typ	Wert
`piece length`	*Integer*	Größe der Pieces in Bytes
`pieces`	*String*	Konkatenierung von 20-Byte SHA1 Hashwerten, ein Wert pro Piece
`name`	*String*	Name des Verzeichnisses, in welchem die Dateien abzulegen sind
`files`	*List of Dictionaries*	Infos zu den Dateien, jeweils mit folgenden Schlüsseln: `length` (*Integer*) Größe der Datei in Bytes `path` (*List of Strings*) Pfadelemente und Dateiname

2.2 Trackerprotokoll

Der Tracker verwaltet Informationen über BitTorrent-Netze und deren Peers. Er stellt eine zentrale Instanz dar, demnach ist BitTorrent kein reines Peer-to-Peer-System. Trackerloses BitTorrent [4] wird im folgenden nicht betrachtet. Die Netze werden anhand eines Hashwertes (`info_hash`), der aus einem Teil der Metainfodatei berechnet wird, unterschieden. So kann der Tracker die Peers miteinander bekannt machen, die an dem selben Torrent interessiert sind.

Das Trackerprotokoll [2] verwendet HTTP [5]. Der Tracker antwortet auf HTTP GET-Anfragen. Die URL für die Anfragen ist in der Metainfodatei hinterlegt. Um eine Anfrage an den Tracker zu stellen, muss ein Peer zunächst eine beliebige `peer_id` fester Länge (20 Byte) generieren und den Wert des `info_hash` anhand der Metainfodatei berechnen. Die Parameter der Anfrage werden in der Form `<Announce URL>?param1=value1&param2=value2&...` an den Tracker übergeben. Notwendige Parameter:

Parameter	**Beschreibung**
`info_hash`	20-Byte SHA1-Hash des Wertes des info-Schlüssels aus der Metainfodatei.
`peer_id`	20-Byte String zur eindeutigen Identifizierung des Peers, generiert durch die Client Software.
`port`	Port, an dem der Peer Verbindungen von anderen Peers annimmt (Standardports: 6881 bis 6889).
`uploaded`	Hochgeladene Datenmenge in Bytes, seit dem das started-Event an den Tracker gesendet wurde.
`downloaded`	Heruntergeladene Datenmenge in Bytes, seit dem das started-Event an den Tracker gesendet wurde.
`left`	Datenmenge in Bytes die noch fehlt, um die Daten vollständig zu haben.
`compact`	Die Peers-Liste wird im *Compact-Modus* akzeptiert.
`event`	Entweder started, stopped, completed oder empty.

Die Antwort des Trackers enthält im erfolgreichen Fall eine Liste von Peers, welche am selben Torrent interessiert sind. Im Fehlerfall enthält die Antwort eine Fehlernotiz. Der Tracker überträgt seine Antwort in Form eines text/plain-Dokuments, welches ein Verzeichnis in Bencoding mit folgendem Inhalt enthält:

Schlüssel	**Typ**	**Wert**
`failure reason`	*String*	Fehlerbeschreibung. Wenn vorhanden, dann ist dies normalerweise der einzige Schlüssel.
`warning message`	*String*	Warnhinweis. Die Anfrage war dennoch erfolgreich.
`interval`	*Integer*	Minimale Zeit in Sekunden, die der Peer zwischen Trackeranfragen warten muss.
`complete`	*Integer*	Anzahl an *Seedern*.
`incomplete`	*Integer*	Anzahl an *Leechern*.
`peers`	*String*	Compact-Modus: Jeweils 6 Bytes pro Peer (IP-Adresse und Port). Ursprüngliche Form: Bencodierte Liste von Dictionaries, jeweils mit den Schlüsseln peer id, ip und port.

2.3 Peerprotokoll

Für den Austausch der Pieces, so wie sie in der Metainfodatei beschrieben sind, wird ein eigenes Protokoll verwendet. Das Peerprotokoll [2] (auch PeerWire-Protokoll genannt) kommt bei BitTorrent Peer-to-Peer Verbindungen zum Einsatz. Es ist ein binäres, bidirektional funktionierendes Protokoll. Nach dem Aufbau einer Verbindung wird keine Unterscheidung zwischen den beiden beteiligten Peers gemacht, beide agieren gleichwertig.

Eine Übersicht der Pakete und deren Aufbau geben Tabellen 2 und 3. Mit Ausnahme der Handshake-Nachricht haben alle Pakete ihre Länge als Präfix. Es gibt drei Pakete mit Variabler Länge: Handshake, Bitfield, sowie Piece. Nach dem Aufbau einer TCP-Verbindung zwischen zwei Peers tauschen diese Handshake-Pakete aus. Hat der Wert des info hash-Parameters nicht den erwarteten Wert, nämlich den selben der auch schon für die Trackeranfrage berechnet wurde, wird die Verbindung abgebrochen. Direkt nach dem Handshaking besteht die einzige Möglichkeit, dem Gegenüber ein Bitfield-Paket zu senden. In dem Bitfeld steht jedes Bit für ein Piece, ist der Wert 1, so hat der sendende Peer das Piece, ansonsten nicht.

Im Laufe der Peer-to-Peer Verbindung werden Pakete der weiteren Typen aus Tabelle 2 ausgetauscht. Die Piece-Pakete stellen dabei den tatsächlichen Austausch von Teilen der durch dieses BitTorrent-Netz verteilten Daten dar. Pieces werden nicht komplett, sondern in so genannten *Chunks* übertragen. Ein besonderes Augenmerk sollte auf die Choke und Unchoke Mitteilungen fallen, denn mit deren Hilfe wird ein Tit-for-Tat-Algorithmus umgesetzt: Erhält ein Peer ein Choke-Paket, so ist dieser Peer im choked-Status. Dies bedeutet, dass keine neuen Requests gestellt werden dürfen, und alle noch nicht beantworteten Requests verworfen werden. Erhält dieser Peer ein Unchoke-Paket, können wieder Requests gestellt werden. Ist ein Peer im choked-Status und hat Interesse an Pieces von dem verbundenen Peer, so kann ein Interested-Paket gesendet werden, um dem Gegenüber mitzuteilen, dass Requests gestellt werden, sobald ein Unchoke-Paket empfangen wird. Der initiale Status eines Peers nach dem Aufbau einer Peer-to-Peer Verbindung ist choked.

Tabelle 4 veranschaulicht eine beispielhafte BitTorrent Peer-to-Peer-Verbindung. Die Quelle ist ein Mitschnitt von realem BitTorrent-Verkehr (deutlich gekürzt), zur Analyse wurde Bro mit unserem BitTorrent-Analyzer eingesetzt.

3 Bro NIDS

Bro [7, 8] ist ein Network Intrusion Detection System (*NIDS*). Es unterliegt einer BSD Lizenz und läuft auf Unix-artigen Systemen wie *BSD und Linux. Bro ist ein Projekt der University of California, Lawrence Berkeley National Laboratory, initiiert von Vern Paxson. Die Entwicklung von Bro begann 1995. Ein herausstechendes Feature von Bro ist die Möglichkeit, Protokolle unter Mitführung von Statusinformationen zu analysieren.

3.1 Aufbau und Funktionsweise

Um Angriffe zu erkennen, wird der Netzwerkverkehr bis hin zum Applikationsprotokoll geparst. Die dafür verwendeten Protokollparser werden entweder per Hand in C++ geschrieben, oder durch Binpac (Abschnitt 3.2) generiert. Letzterer ist der empfohlene Weg

Nachricht	**Kodierung**

Handshake	`| pstrlen | pstr | rsvd | info hash | peer id |`
	`0 ------- 1 ---- 20 --- 28 -------- 48 ----- 68`
	
	`pstrlen`: Länge von `pstr` (Wert: 19)
	`pstr`: Protokollidentifier (Wert: „BitTorrent protocol“)
	`rsvd`: Reserviert, laut Spezifikation [2, 6] gleich Null.
	`info hash`: SHA1 des Torrents
	`peer id`: Peer Identifier

Keep Alive	`| len |`
	`0 --- 3`
	
	`len`: Länge des *PDU*-Payload (Wert: 0)

Choke	`| len | id |`
	`0 --- 3 -- 4`
	
	`len`: Länge des PDU-Payload (Wert: 1)
	`id`: Message-ID (Wert: 0)

Unchoke	`| len | id |`
	`0 --- 3 -- 4`
	
	`len`: Länge des PDU-Payload (Wert: 1)
	`id`: Message-ID (Wert: 1)

Interested	`| len | id |`
	`0 --- 3 -- 4`
	
	`len`: Länge des PDU-Payload (Wert: 1)
	`id`: Message-ID (Wert: 2)

Not Interested	`| len | id |`
	`0 --- 3 -- 4`
	
	`len`: Länge des PDU-Payload (Wert: 1)
	`id`: Message-ID (Wert: 3)

Have	`| len | id | piece index |`
	`0 --- 3 -- 4 ---------- 8`
	
	`len`: Länge des PDU-Payload (Wert: 5)
	`id`: Message-ID (Wert: 4)
	`piece index`: Index des neu verfügbaren Pieces

Tabelle 2: Peerprotokoll: Nachrichten (Wert: 1)

Bitfield	`| len | id | bitfield |`
	`0 --- 3 -- 4 -------- x`
	
	`len`: Länge des PDU-Payload (Wert: 1 + Länge von `bitfield`)
	`id`: Message-ID (Wert: 5)
	`bitfield`: Jedes Bit in dem String steht für ein Piece, bei gesetztem Bit ist das Piece verfügbar.

Request	`| len | id | index | begin | length |`
	`0 --- 3 -- 4 ----- 8 ----- 12 ---- 16`
	
	`len`: Länge des PDU-Payload (Wert: 13)
	`id`: Message-ID (Wert: 6)
	`index`: Index des gewünschten Pieces
	`begin`: Anfangsadresse des angeforderten Chunks relativ zum Piecebeginn
	`length`: Chunklänge

Pieces	`| len | id | index | begin | block |`
	`0 --- 3 -- 4 ----- 8 ----- 12 ---- x`
	
	`len`: Länge des PDU-Payload (Wert: 9 + Chunklänge)
	`id`: Message-ID (Wert: 7)
	`index`: Index des gewünschten Pieces
	`begin`: Anfangsadresse des angeforderten Chunks relativ zum Piecebeginn
	`block`: Chunkdaten

Cancel	`| len | id | index | begin | length |`
	`0 --- 3 -- 4 ----- 8 ----- 12 ---- 16`
	
	`len`: Länge des PDU-Payload (Wert: 13)
	`id`: Message-ID (Wert: 8)
	`index`: Index des gewünschten Pieces
	`begin`: Anfangsadresse des angeforderten Chunks relativ zum Piecebeginn
	`length`: Chunklänge

Port	`| len | id | listen-port |`
	`0 --- 3 -- 4 ---------- 6`
	
	`len`: Länge des PDU-Payload (Wert: 3)
	`id`: Message-ID (Wert: 9)
	`listen-port`: Port, an dem Verbindungen angenommen werden

Tabelle 3: Peerprotokoll: Nachrichten (Wert: 2)

Mitteilungstyp	Peer 1 (orig)		Peer 2 (resp)	Parameter
handshake	10.1.1.1:6881	<	10.2.2.2:5555	info hash, peer id, ...
handshake	10.1.1.1:6881	>	10.2.2.2:5555	info hash, peer id, ...
bitfield	10.1.1.1:6881	<	10.2.2.2:5555	bitfield
bitfield	10.1.1.1:6881	>	10.2.2.2:5555	bitfield
interested	10.1.1.1:6881	<	10.2.2.2:5555	
unchoke	10.1.1.1:6881	>	10.2.2.2:5555	
request	10.1.1.1:6881	<	10.2.2.2:5555	piece index, begin, length
piece	10.1.1.1:6881	>	10.2.2.2:5555	piece index, begin, block
request	10.1.1.1:6881	<	10.2.2.2:5555	piece index, begin, length
have	10.1.1.1:6881	<	10.2.2.2:5555	piece index
...				
keep-alive	10.1.1.1:6881	<	10.2.2.2:5555	
...				

Tabelle 4: Peer-to-Peer Verbindung (Ausschnitt)

für neue Analyzer. Die Protokollparser erzeugen dann geeignete Ereignisse, welche wiederum durch ereignisorientierte Analyzerskripte, geschrieben in der Bro-Policysprache, behandelt werden. Darin können detaillierte Logeinträge erzeugt oder Systemkommandos ausgeführt werden. Es lassen sich auch wesentlich komplexere Probleme mit der Bro-Policysprache lösen, beispielsweise können Statusinformationen in Form von Variablen über Events hinweg gespeichert, oder aus einem Eventhandler heraus neue Events generiert werden.

3.2 Binpac

Binpac [17, 9] ist ein Generator für Protokollparser. Ein Teil der Analyzer, die in Bro enthalten sind, verwenden Binpac. Obwohl Binpac ein Teil der Bro Distribution ist, kann es grundsätzlich auch ohne Bro verwendet werden. Einige in Bro enthaltene Analyzer wie CIFS/SMB, DNS, Sun/RPC und HTTP verwenden einen durch Binpac generierten Parser. Für den Parser des BitTorrent Peerprotokolls kommt Binpac ebenfalls zum Einsatz.

Das zu parsende Protokoll wird in einer Binpac-eigenen hohen Beschreibungssprache definiert. Dabei besteht die Möglichkeit, direkt C++ Code einzubetten. Des weiteren besteht die Möglichkeit, Bro-Ereignisse auszulösen, wenn beispielsweise eine *PDU* vollständig analysiert ist. Aus dieser Beschreibung erzeugt Binpac anschließend einen Parser in C++.

4 Implementierung des BitTorrent Analyzers

Da BitTorrent zwei Protokolle vereint, das Trackerprotokoll (Abschnitt 2.2) und das Peerprotokoll (Abschnitt 2.3), werden auch zwei separate Analyzer benötigt. Der Analyzer für das Trackerprotokoll (Abschnitt 4.1) besteht aus einem HTTP-Parser und einem Decoder für Bencoding, sowie einem Skript in der Bro-Policysprache, um unter anderem Logfileeinträge zu generieren. Der Analyzer für das Peerprotokoll (Abschnitt 4.2) besteht aus einem durch Binpac generierten Protokollparser und ebenfalls einem Skript in der Bro-Policysprache.

Es stellt sich das Problem, wie BitTorrent in realem Netzwerkverkehr erkannt werden kann, schließlich gibt es weder für die Peers, noch für die Tracker feste Ports, an denen Verbindungen angenommen werden. Bro implementiert dafür eine Technik mit der Bezeichnung Dynamic Protocol Detection (DPD) [16]. Mit Hilfe von DPD werden Datenpakete auf beliebigen (in dem hier benötigten Fall auf allen) Ports mit Hilfe von Signaturen untersucht, um im Trefferfall den gewünschten Analyzer zu starten. Wir haben für das Tracker- und das Peerprotokoll geeignete Signaturen entwickelt.

Eine weiteres Feature von Bro ist, dass man Verbindungen mit beliebigen Endpunkten erwarten, und einen beliebigen Analyzer dafür delegieren kann. Die entsprechende Funktion heißt `expect_connection`. Die Informationen, welche das Trackerprotokoll bietet, reichen aus, um Endpunkte für potentielle Peerverbindungen zu vorherzusagen. Was in dem Policyskript für das Trackerprotokoll zu tun bleibt, ist das Anmelden des Peerprotokoll-Analyzers für Verbindungen, die entweder an den aktuellen Peer, oder an irgendeinen Peer aus der vom Tracker erhaltenen Liste geht.

Allein das Zutreffen einer Signatur oder das Vermuten von Verbindungen anhand der Informationen aus Trackeranfragen gibt noch keine sichere Auskunft darüber, ob es sich tatsächlich um das erwartete Protokoll handelt oder nicht. Ein fälschlich gestarteter Analyzer kann sich selbst wieder stoppen und den Grund dafür mit Hilfe der Funktion `ProtocolViolation` melden, im positiven Fall sollten Analyzer Aufrufe der Funktion `ProtocolConfirmation` tätigen.

4.1 Analyzer für das Trackerprotokoll

Das Trackerprotokoll verwendet HTTP für den Transport der Informationen. Bro's DPD aktiviert anhand der folgenden Signatur unseren Trackeranalyzer:

```
^.*\/announce\?.*info_hash
```

Für den Fall eines Treffers nehmen wir an, dass es sich um eine Trackerverbindung handelt und starten unseren HTTP-Parser. Dieser analysiert die angefragte URL, speichert die Parameter in geeigneter Form in Variablen und reicht diese durch Auslösen eines Ereignisses an ein Policyskript weiter. Neben der Generierung von Logeinträgen erfolgt darin der erste Aufruf der `expect_connection`-Funktion. Denn wir erwarten von Verbindungen, die als Ziel die IP-Adresse des Clients und den in der Trackeranfrage mitgeteilten Port verwenden, dass es sich um das Peerprotokoll handelt, und melden auf diese Weise unseren Analyzer für das Peerprotokoll an.

Die Antwort des Trackers wird ebenfalls durch unseren HTTP-Parser geschickt, um in erster Linie den bencodeten Inhalt zu ermitteln. Dieser wird von unserem Bencoding-Parser eingelesen, und in passender Variablenform per Event an das Policyskript weitergereicht. Darin erfolgen neben der Generierung von Logeinträgen weitere Aufrufe der `expect_connection`-Funktion, jeweils einen für jeden Peer, der in der Trackerantwort übermittelt wurde. Wir erwarten wieder das Peerprotokoll in Verbindungen, die vom Client ausgehend an Peers aus der Liste vom Tracker gerichtet sind.

Jetzt haben wir einen Analyzer für das Trackerprotokoll entwickelt. Trackeranfragen und -antworten werden als solche erkannt und analysiert, außerdem wird der Peerprotokoll-Analyzer für erwartete Peer-to-Peer-Verbindungen aktiviert.

	2005	**2007**
Verbindungen	24630	20413
Anzahl verschiedener Tracker	141	136
Ports	6969: 32% 2710: 19% 3389: 10%	2710: 39% 6969: 20% 8000: 5%
Persistent Connection	17 (0,07%)	33 (0,06%)

Tabelle 5: Trackerstatistik

4.2 Analyzer für das Peerprotokoll

Das Peerprotokoll verwendet TCP. Der TCP-Analyzer von Bro kann verwendet werden, um das TCP-Payload als Datenstrom abzugreifen. Dieser Datenstrom wird durch einen per Binpac generierten Parser geleitet. Unser Parser erlaubt Lücken in den Netzwerkdaten, so genannte *Content Gaps*, falls sie innerhalb der `block`-Daten der piece-*PDU* (Tabelle 3) liegen. Für jede geparste PDU wird ein Bro-Ereignis ausgelöst. Behandelt werden diese wieder durch ein Policyskript. Dort geschieht nicht viel mehr als die Generierung von Einträgen in einer Logdatei.

5 Ergebnisse

Im Rahmen dieser Arbeit untersuchen wir mit unserem BitTorrent-Analyzer realen Netzwerkverkehr auf BitTorrent. Dabei betrachten wir die Ergebnisse unserer Peer- und Trackeranalysen separat und suchen in den Auswertungen nach Entsprechungen und Wiedersprüchen.

5.1 Datenquelle

Zu Analysezwecken liegen uns zwei Traces der Verbindung des Münchner Hochschulnetzes (MHN) und des Internets vor. Der eine beinhaltet den Mitschnitt von 24 Stunden Netzwerkverkehr (ca. 3,1 TB), begonnen am Dienstag, den 11. Oktober 2005 um 13:43 Uhr. Der andere geht über 11 Stunden (ca. 2,1 TB), begonnen am Dienstag, den 29. Mai 2007 um 23:10 Uhr. Im folgenden betrachten wir lediglich TCP-Payloads.

5.2 Ergebnisse der Trackeranalysen

Einen Überblick über die erkannten Trackerverbindungen gibt Tabelle 5. Rechnet man die Länge der Traces in die Statistik mit ein, dann hat sich die Anzahl der Trackerverbindungen von 2005 auf 2007 etwa verdoppelt. Die Anzahl der verschiedenen Tracker ist in etwa gleich geblieben. Ebenso die Tatsache, dass nur wenige Tracker, nämlich 14 (2005) bzw. 18 (2007) etwa 70% der Anfragen abarbeiten. Bei einem Blick auf die Ports, an welchen die Tracker Anfragen annehmen, wird deutlich, dass die Standardports 6969 und 2710 zwar die Prominentesten sind, jedoch 49% (2005) bzw. 41% (2007) aller Anfragen über Nichtstandardports angenommen werden. Das HTTP-Feature „Persistent Connection“ kommt bei Trackerverbindungen nicht bzw. in nicht erwähnenswertem Maße zum Einsatz.

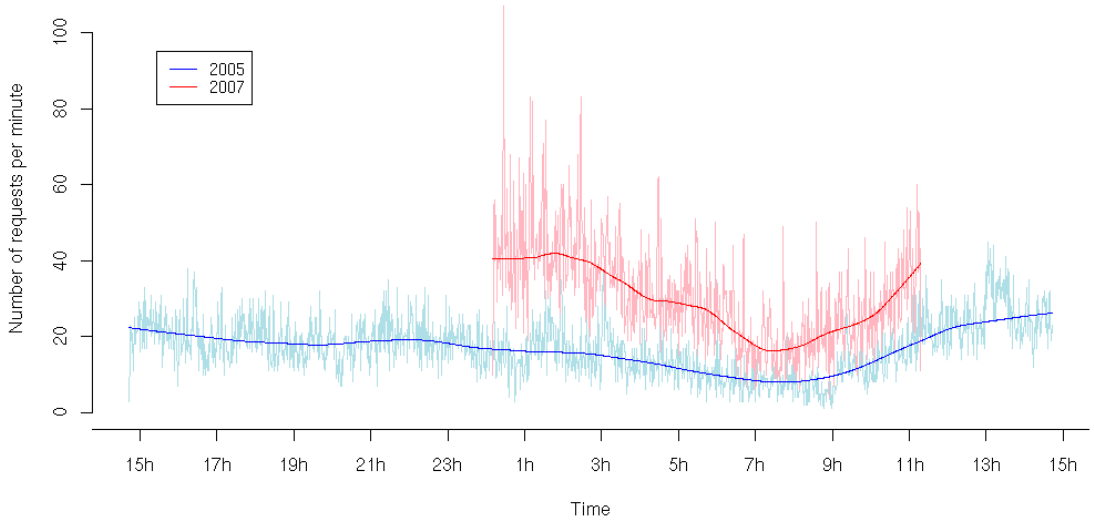

Abbildung 1: Trackeranfragen

Abbildung 1 stellt die Anzahl an Trackeranfragen pro Minute über die Zeit dar. Die im Vordergrund deutlich sichtbaren Kurven sind geglättete Version der schwach gedruckten Linien, welche die unveränderten Messdaten wiedergeben. In beiden Jahren ist der Tageszeiteffekt deutlich zu erkennen. An diesem Diagramm ist zu erkennen, dass im Jahr 2007 deutlich mehr Trackeranfragen stattfanden als 2005. Zu sehen ist außerdem eine leicht gestiegene Anzahl an Anfragen, wenn man für das Jahr 2005 den Stand der Kurve um 15h an Beginn, und um 15h am Ende des Traces vergleicht. Einen dazu passenden Effekt erkennen wir im folgenden auch bei der Analyse der Peerverbindungen.

Die Liste von Peers, die ein Tracker in seiner Antwort zurück gibt, ist längenmäßig variabel. Aufschluss über die Länge dieser Liste gibt Abbildung 2. Die Standardlänge dieser Liste ist 50, was in beiden Jahren auch am häufigsten auftritt. Als ein weiterer Standardwert scheint sich die 100 zu etablieren. Im Jahr 2007 ist im Vergleich zu 2005 der Anteil an Trackerantworten gestiegen, die weniger als 50 Peers bekannt geben. Dies passiert genau dann, wenn dem Tracker keine, oder zumindest weniger als 50 Peers für den jeweiligen Torrent bekannt sind, oder wenn in der Anfrage weniger als 50 Peers verlangt werden.

5.3 Ergebnisse der Peeranalysen

Im Gegensatz zu den Trackerverbindungen gibt es in der Anzahl der Peerverbindungen von 2005 auf 2007 einen Abfall um etwa 20% (Tabelle 6). Das Volumen ist auch deutlich gefallen, die Anzahl an verschiedenen Hosts ist in beiden Jahren etwa gleich. In der Statistik fällt auf, dass es im Jahr 2005 noch sehr viel populärer war, den ersten Standardport für Peerverbindungen zu verwenden, nämlich 6881. Sollte dieser Port bereits in Verwendung sein, so sieht es die Protokollspezifikation vor, werden aufsteigend alle Ports bis 6889 versucht. Anhand der `peer id` lässt sich in vielen Fällen auf die verwendete Clientsoft-

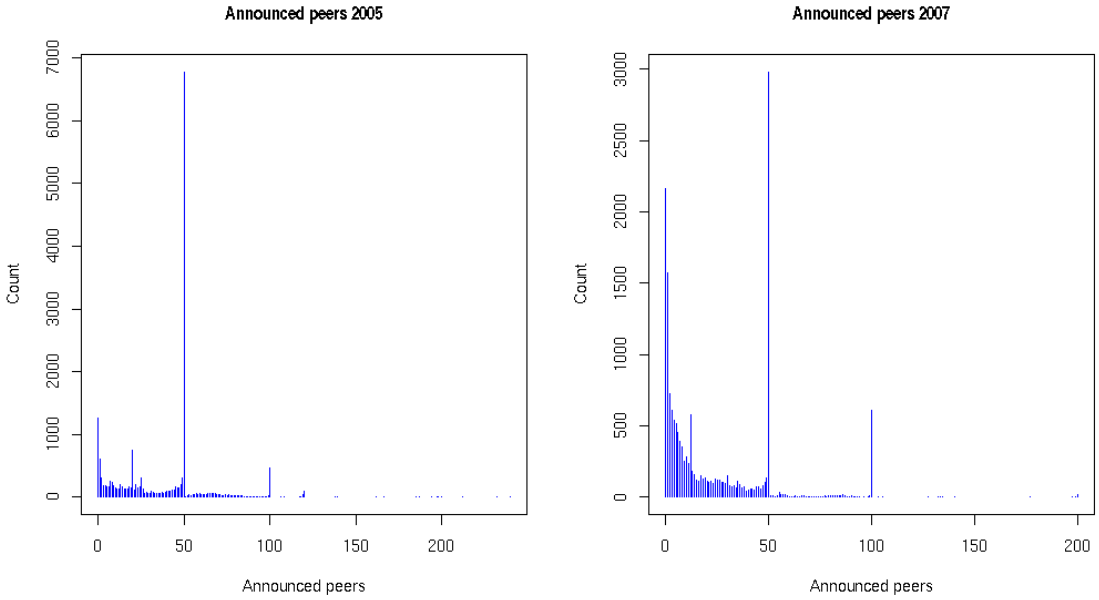

Abbildung 2: Anzahl an Peers in Trackerantworten

	2005	**2007**
Verbindungen	640402	244134
Volumen	ca. 142 GB	ca. 36 GB
Hosts	105142	55078
Ports	6881: 37,3% 6882: 4,9% 16881: 2,7%	7000: 6,8% 6881: 4,1% 60003: 2,7%
Software	Azureus: 29,8% BitComet/BitLord: 26,2% BitComet: 22,0% Bram's client: 3,9%	uTorrent: 32,7% Azureus: 23,4% BitComet: 21,3% Bram's client: 14,7%

Tabelle 6: Peerstatistik

ware schließen. In Tabelle 6 stellen wir die daraus resultierenden Top-4 der verwendeten Clientprogramme dar. Azureus [10] war 2005 am häufigsten Vertreten, im Jahr 2007 war es uTorrent [11]. Bram's Client [12] ist die Software von Bram Cohen, dem Erfinder von BitTorrent.

Abbildung 3 stellt die Anzahl an simultanen Peerverbindungen über die Zeit dar. Ähnlich wie in Abbildung 1 ist auch hier der Effekt der Tageszeit in beiden Jahren deutlich erkennbar. Auch der Abfall an Peerverbindungen beim Vergleich der Jahre 2005 und 2007 ist deutlich zu sehen. Eine mögliche Ursache dafür könnte ein erhöhter Einsatz von verschlüsseltem bzw. verschleiertem BitTorrent [13, 14] sein, da wir diese Protokollvarianten mit unserem Analyzer nicht erkennen. Im Rahmen der Deutung von Abbildung 1 haben wir einen leichten Zuwachs an Trackeranfragen beobachtet, wenn man die Werte von 15h am Beginn und 15h am Ende des 2005er Traces vergleicht. Dieser Effekt kommt in Abbildung 3, sowie bei der im folgenden betrachteten Abbildung 4, erneut und sogar etwas deutlicher zum Vorschein.

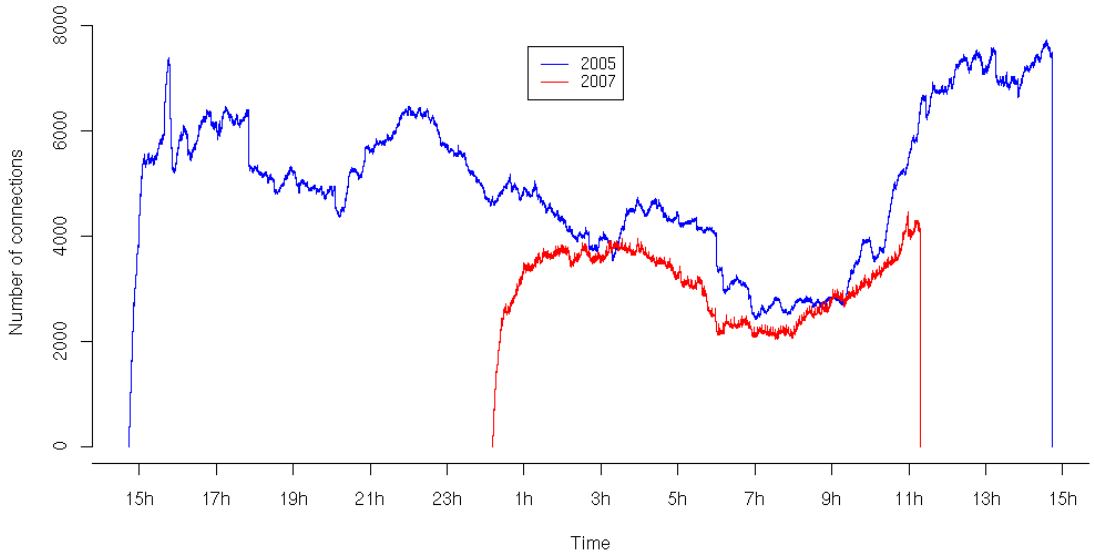

Abbildung 3: simultane Peerverbindungen

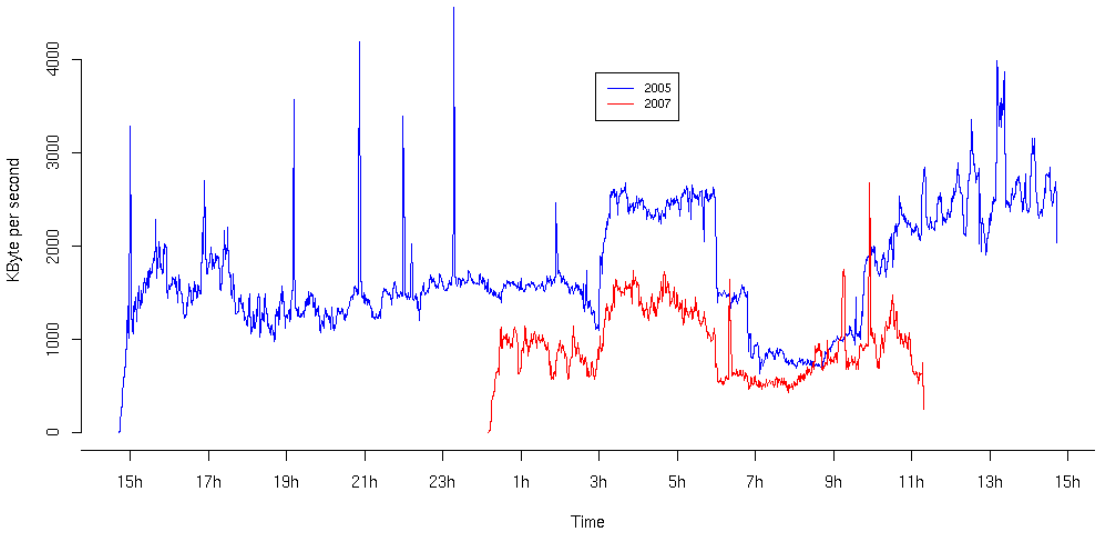

Abbildung 4: Bandbreitenverbrauch durch Peerverbindungen

Der Bandbreitengraph (Abbildung 4) stellt den durch das BitTorrent Peerprotokoll erzeugten Netzwerkverkehr in Kilobyte pro Sekunde über die Zeit dar. Dieser bestätigt den in Tabelle 6 angegebenen Abfall des Volumens von 2005 auf 2007. Davon abgesehen ist der Verlauf der Kurven beider Jahre sehr ähnlich. Interessant ist der plötzliche und deutliche Anstieg beider Kurven von etwa 3 bis 6 Uhr nachts. Als Ursache für diesen Effekt vermuten wir die durch das DFN (Deutsches Forschungsnetz) bereitgestellte Flatrate für die Nachtstunden. Aus daraus resultierenden Kosteneinsparungen motiviert das LRZ (Leibnitz Rechenzentrum) seine Benutzer dazu, Netzlast vom Tag in die Stunden der Nacht zu verlagern. Belegen können wir diese Vermutung jedoch nicht.

6 Fazit

Mit unserem BitTorrent-Analyzer für das Bro NIDS haben wir die Möglichkeit geschaffen, BitTorrent-Verbindungen zu erkennen und Statistiken über BitTorrent-Verkehr zu erstellen, entweder aus archivierten Mitschnitten oder aus Livedaten eines Netzwerkes. Wir haben zwei größere Mitschnitte eines Hochschulnetzwerkes analysiert. Die so gewonnenen Statistiken zeigen einige erwartete, aber auch unerwartete Eigenschaften auf. Den Effekt der Tageszeit haben wir erwartet, jedoch haben wir insgesamt weniger BitTorrent-Verkehr gemessen als angenommen, im Jahr 2007 sogar weniger als in 2005. Interessant wären weitere Analysen anderer großer Netzwerke, sowie eine Erweiterung des Analyzers um bisher nicht unterstützte Protokollvarianten wie dem HTTPS und dem UDP Trackerprotokoll [15], der trackerlosen Protokollerweiterung [4], welche verteilte Hashtabellen verwendet, als auch dem verschlüsselten Peerprotokoll [13, 14].

Glossar

Bencoding ASCII-basiertes infix Kodierungsverfahren für komplexe, baumartige Datenstrukturen (Tabelle 1).

Chunk Teilstück eines Pieces.

Compact-Modus Bytestring mit 6 Byte pro Peer (IP-Adresse und Port).

Content-Gap Fehlende Pakete beim Mitschnitt von Netzwerkverkehr.

Leecher Peer, der die Daten noch gar nicht, oder nur teilweise besitzt.

NIDS Network Intrusion Detection System.

Peer Teilnehmer eines BitTorrent-Netzes, entweder Seeder oder Leecher.

PDU Protocol Data Unit.

Piece Teilstück an Daten, die per BitTorrent verteilt werden.

Seeder Peer, der die Daten komplett hat und anderen Peers zur Verfügung stellt.

Torrent Ein Datensatz, der per BitTorrent verteilt wird.

Tracker Zentrale Einheit eines BitTorrent-Netzes. Macht Peers untereinander bekannt, die am selben Torrent interessiert sind.

Literatur

[1] Ipoque GmbH Pressemitteilung: http://www.ipoque.com/media/news/pressrelease_ipoque_241006.html.

[2] Official Protocol Specifications v1.0: http://bittorrent.org/protocol.html.

[3] Bencoding: http://en.wikipedia.org/wiki/Bencoding.

[4] DHT-Protokoll (trackerloses BitTorrent): http://www.bittorrent.org/DHT_protocol.html.

[5] HTTP-Protokoll: http://de.wikipedia.org/wiki/HTTP.

[6] theory.org BitTorrent Specification: http://wiki.theory.org/BitTorrentSpecification.

[7] Bro Website: http://www.bro-ids.org/.

[8] Bro Wiki: http://www.bro-ids.org/wiki/.

[9] Binpac Wiki: http://bro-ids.org/wiki/index.php/BinPAC.

[10] Azureus: http://azureus.sourceforge.net/.

[11] uTorrent: http://www.utorrent.com/.

[12] Bram's Client (bis Version 5): http://download.bittorrent.com/dl/.

[13] BitTorrent Protocol Encryption: http://en.wikipedia.org/wiki/BitTorrent_protocol_encryption.

[14] Bram Cohen: Obfuscating BitTorrent: http://bramcohen.livejournal.com/29886.html.

[15] UDP-Trackerprotokoll: http://xbtt.sourceforge.net/udp_tracker_protocol.html.

[16] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic application-layer protocol analysis for network intrusion detection. Proc. USENIX Security Symposium, 2006.

[17] R. Pang, V. Paxson, R. Sommer, and L. Peterson. binpac: a yacc for writing application protocol parsers. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 289–300, New York, NY, USA, 2006. ACM Press.